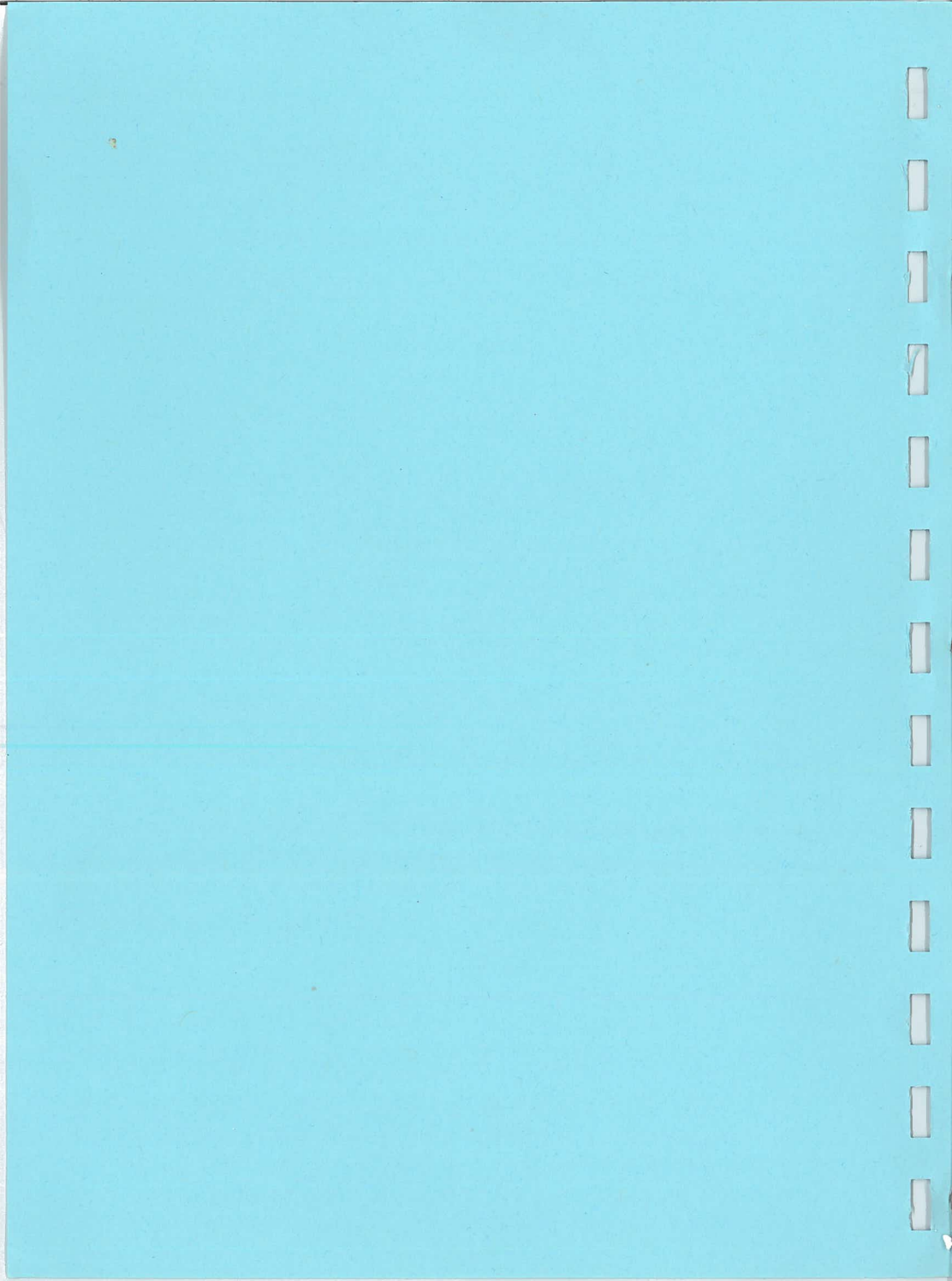




**OSA90/hope™**  
**User's Manual**

*Optimization Systems Associates Inc.*



**OSA90/hope™**  
**User's Manual**

**Version 4.0**

**August, 1997**

*Optimization Systems Associates Inc.*

## LIABILITY AND WARRANTY

NEITHER OPTIMIZATION SYSTEMS ASSOCIATES INC. NOR ITS EMPLOYEES, OFFICERS, DIRECTORS OR ANY OTHER PERSON, COMPANY, AGENCY OR INSTITUTION: (1) MAKES ANY WARRANTY, EXPRESS OR IMPLIED AS TO ANY MATTER WHATSOEVER REGARDING THIS MATERIAL, INCLUDING BUT NOT LIMITED TO THE GENERALITY THEREOF, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, OR THOSE ARISING BY STATUTE OR OTHERWISE IN LAW OR FROM THE COURSE OF DEALING OR USAGE OF TRADE HAVE BEEN AND ARE HEREBY EXPRESSLY EXCLUDED; OR (2) ASSUMES ANY LEGAL RESPONSIBILITY WHATSOEVER FOR THE ACCURACY, COMPLETENESS OR USEFULNESS OF THIS MATERIAL; OR (3) REPRESENTS THAT ITS USE WOULD NOT INFRINGE UPON PRIVATELY OWNED RIGHTS OF THIRD PARTIES. IT IS EXPRESSLY UNDERSTOOD AND AGREED THAT ANY RISKS, LIABILITIES OR LOSSES ARISING OUT OF ANY USE, TRANSFER OR LEASE OF THIS MATERIAL WILL NOT BE ATTRIBUTED TO OPTIMIZATION SYSTEMS ASSOCIATES INC. OR ANY INDIVIDUAL ASSOCIATED WITH THE COMPANY. ACCURACY, COMPLETENESS OR USEFULNESS FOR ANY APPLICATION SHALL BE DETERMINED INDEPENDENTLY BY THE PARTY UNDERTAKING SUCH AN APPLICATION.

IN NO EVENT WHATSOEVER WILL OPTIMIZATION SYSTEMS ASSOCIATES INC., ITS EMPLOYEES, OFFICERS, DIRECTORS, OR AGENTS BE LIABLE FOR ANY DAMAGES, INCLUDING, BUT WITHOUT LIMITATION, DIRECT, INDIRECT, INCIDENTAL AND CONSEQUENTIAL DAMAGES AND DAMAGES FOR LOST DATA OR PROFITS, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS MATERIAL.

CONTENTS ARE SUBJECT TO CHANGE WITHOUT NOTICE.

## Copyright

Copyright © 1997 Optimization Systems Associates Inc.

All rights reserved. The information contained in this document is the proprietary and confidential information of Optimization Systems Associates Inc. Reproduction of this document in whole or in part, or the use or disclosure of any of the information contained herein, without the prior express written authorization of Optimization Systems Associates Inc. is prohibited.

OSA90/hope User's Manual Version 4.0 first published in 1997. Printed in Canada.

Optimization Systems Associates Inc.  
P.O. Box 8083, Dundas, Ontario  
Canada L9H 5E7

Tel 905 628 8228  
Fax 905 628 8225

Email [osa@osacad.com](mailto:osa@osacad.com)

## Trademarks of Optimization Systems Associates Inc.

Datapipe  
Empipe  
Empipe3D  
FAST  
Geometry Capture  
HarPE  
OSA90  
OSA90/hope  
Space Mapping  
Spicepipe

## Other Trademarks

Windows95 and WindowsNT are registered trademarks of the Microsoft Corporation.  
Touchstone is a trademark of the Hewlett-Packard Company.  
*em* and *xgeom* are trademarks of Sonnet Software, Inc.

# OSA90/hope™ User's Manual

## Table of Contents

### 1 Installation

1.1	Introduction	1-1
1.2	Installing the Software	1-2
1.3	Installing the Hardware Lock	1-3
1.4	OSA90 Example Directory	1-4
1.5	Uninstalling the software	1-5

### 2 Technical Overview

2.1	Overview	2-1
2.2	OSA90 Window	2-5
2.3	OSA90 User Interface	2-6
2.4	OSA90 Input File and File Editor	2-10
2.5	Simulation and Optimization	2-11
2.6	A Practice Session	2-12

### 3 Input File

3.1	Overview	3-1
3.2	Keywords	3-1
3.3	Input File Blocks	3-2
3.4	Input File Templates	3-3
3.5	Statements	3-5
3.6	Preprocessor Macros	3-6
3.7	Include Files	3-12
3.8	Control Block	3-13
3.9	Physical Units	3-26

### 4 Expressions

4.1	Overview	4-1
4.2	Label Names	4-2
4.3	Constant Labels	4-3
4.4	Predefined Labels	4-4

## Table of Contents

4.5	Optimization Variables	4-6
4.6	Expressions	4-8
4.7	Conditional Expressions	4-10
4.8	Arrays	4-13
4.9	Array Expressions	4-17
4.10	Array Functions	4-19
4.11	String Labels	4-41
4.12	Transformations	4-43
4.13	Cubic Spline Functions	4-49
4.14	Piece-Wise Linear Interpolation	4-52

## 5 Datapipe

5.1	Overview	5-1
5.2	Datapipe Server	5-3
5.3	Datapipe Protocols	5-6
5.4	SIM Protocol	5-9
5.5	COM and COMD Protocols	5-15
5.6	FUN Protocol	5-24
5.7	FDF Protocol	5-25
5.8	LINEAR Protocol	5-29
5.9	Application Notes	5-32

## 6 Circuit Models

6.1	Overview	6-1
6.2	Nodes	6-3
6.3	Elements	6-4
6.4	Ideal Sources	6-6
6.5	Ports	6-11
6.6	Voltage and Current Labels	6-16
6.7	CIRCUIT Statement	6-18
6.8	DC Responses	6-19
6.9	Small-Signal Responses	6-22
6.10	Large-Signal Responses	6-26
6.11	Postprocessing Responses	6-36
6.12	Linear Subcircuits	6-38
6.13	User-Defined Linear Models	6-41
6.14	User-Defined Nonlinear Models	6-44
6.15	ImportData Block	6-48
6.16	LINEAR Datapipe	6-56
6.17	Macros and Symbolic Subcircuits	6-58
6.18	Oscillator Port	6-61

## 7 Nonlinear Elements

<b>BJTU</b>	user-definable bipolar junction transistor model	7-2
<b>DIODE</b>	SPICE diode model	7-3
<b>DIODE0</b>	semiconductor diode model	7-7
<b>DIODEU</b>	user-definable diode model	7-9
<b>FETC</b>	Curtice and Ettenberg FET model	7-10
<b>FETCA</b>	Curtice asymmetrical FET model	7-12
<b>FETM</b>	Materka and Kacprzak FET model	7-14
<b>FETR</b>	Raytheon FET model	7-17
<b>FETT</b>	physics-based Khatibzadeh and Trew FET model	7-19
<b>FETT1</b>	modified physics-based FET model	7-23
<b>FETU1</b>	user-definable FET model 1	7-26
<b>FETU2</b>	user-definable FET model 2	7-27
<b>HBT</b>	heterojunction bipolar transistor model	7-28
<b>HEMTAC</b>	advanced Curtice HEMT model	7-31
<b>HEMTC</b>	Curtice HEMT model	7-34
<b>HEMTG1</b>	high order beta degradation HEMT model	7-37
<b>HEMTG2</b>	double parabolic HEMT model	7-40
<b>KTL</b>	physics-based bias-dependent small-signal FET model	7-43
<b>NLCCS</b>	nonlinear controlled current source	7-48
<b>NLCQ</b>	nonlinear controlled charge source	7-49
<b>NPN</b>	NPN Gummel and Poon bipolar transistor model	7-50
<b>PNP</b>	PNP Gummel and Poon bipolar transistor model	7-52

## 8 Linear Elements

<b>CAP</b>	capacitor	8-2
<b>CCCS</b>	current controlled current source	8-3
<b>CCVS</b>	current controlled voltage source	8-4
<b>CIR3</b>	ideal three-port circulator	8-5
<b>CLIN</b>	ideal coupled transmission lines	8-6
<b>CLINP</b>	coupled transmission lines, physical model	8-7
<b>EXTRINSIC1</b>	extrinsic parasitic model 1	8-8
<b>EXTRINSIC2</b>	extrinsic parasitic model 2	8-9
<b>EXTRINSIC3</b>	extrinsic parasitic model 3	8-10
<b>EXTRINSIC4</b>	extrinsic parasitic model 4	8-11
<b>IND</b>	inductor	8-12
<b>MAGAP</b>	microstrip asymmetric gap	8-13
<b>MBEND1</b>	microstrip 90 degree bend	8-14
<b>MBEND2</b>	microstrip chamfered 90 degree bend	8-15
<b>MBEND3</b>	microstrip optimally-chamfered 90 degree bend	8-16
<b>MBEND3A</b>	microstrip optimally-chamfered 90 degree bend	8-17
<b>MCROSS</b>	microstrip cross-junction	8-18
<b>MGAP</b>	microstrip symmetric gap	8-19
<b>MLANG4</b>	four-finger microstrip Lange coupler	8-20
<b>MLANG6</b>	six-finger microstrip Lange coupler	8-22
<b>MLANG8</b>	eight-finger microstrip Lange coupler	8-24
<b>MOPEN</b>	microstrip open stub	8-26
<b>MRSTUB</b>	microstrip radial stub	8-27

## Table of Contents

<b>MSACL</b>	asymmetrical coupled microstrip lines . . . . .	8-28
<b>MSCL</b>	two-conductor symmetrical coupled microstrip lines . . . . .	8-29
<b>MSHORT</b>	microstrip short stub . . . . .	8-30
<b>MSL</b>	microstrip line . . . . .	8-31
<b>MSLIT</b>	narrow transverse slit in microstrip . . . . .	8-33
<b>MSTEP</b>	microstrip step . . . . .	8-34
<b>MSUB</b>	microstrip substrate definition . . . . .	8-35
<b>MTEE</b>	microstrip T-junction . . . . .	8-36
<b>OPEN</b>	open circuit . . . . .	8-37
<b>PRC</b>	parallel connection of resistor and capacitor . . . . .	8-38
<b>RES</b>	resistor . . . . .	8-39
<b>SRC</b>	series connection of resistor and capacitor . . . . .	8-40
<b>SRL</b>	series connection of resistor and inductor . . . . .	8-41
<b>SRLC</b>	series connection of resistor, inductor and capacitor . . . . .	8-42
<b>TEM</b>	ideal transmission line . . . . .	8-43
<b>TRL</b>	transmission line, physical model . . . . .	8-44
<b>VCCS</b>	voltage controlled current source . . . . .	8-45
<b>VCVS</b>	voltage controlled voltage source . . . . .	8-46
<b>DATAPORT</b>	imported n-port subcircuit . . . . .	8-47
<b>SPORT</b>	n-port subcircuit defined by S matrix . . . . .	8-48
<b>YPORT</b>	n-port subcircuit defined by Y matrix . . . . .	8-54
<b>ZPORT</b>	n-port subcircuit defined by Z matrix . . . . .	8-60

## 9 Simulation

<b>9.1</b>	<b>Overview</b> . . . . .	9-1
<b>9.2</b>	<b>Sweep Block</b> . . . . .	9-1
<b>9.3</b>	<b>Simulation Types</b> . . . . .	9-3
<b>9.4</b>	<b>Parameter Sweeps</b> . . . . .	9-7
<b>9.5</b>	<b>Parameter Labels</b> . . . . .	9-13
<b>9.6</b>	<b>Output Labels</b> . . . . .	9-14
<b>9.7</b>	<b>OSA90 Display Menu</b> . . . . .	9-15
<b>9.8</b>	<b>Xsweep Display</b> . . . . .	9-17
<b>9.9</b>	<b>Numerical Display</b> . . . . .	9-22
<b>9.10</b>	<b>Graphics Zoom</b> . . . . .	9-24
<b>9.11</b>	<b>Draw Types</b> . . . . .	9-27
<b>9.12</b>	<b>Parametric Display</b> . . . . .	9-28
<b>9.13</b>	<b>Array Display</b> . . . . .	9-31
<b>9.14</b>	<b>Visualization and Contour Plotting</b> . . . . .	9-32
<b>9.15</b>	<b>Host Acceleration Files</b> . . . . .	9-43
<b>9.16</b>	<b>Report Generation</b> . . . . .	9-44

## 10 Graphical Views

<b>10.1</b>	<b>Overview</b> . . . . .	10-1
<b>10.2</b>	<b>Xsweep Views</b> . . . . .	10-2
<b>10.3</b>	<b>Specifications Shown In Views</b> . . . . .	10-14
<b>10.4</b>	<b>Select Views for Display</b> . . . . .	10-17



10.5	Parametric Views	10-18
10.6	Waveform Views	10-21
10.7	Smith Chart Views	10-26
10.8	Polar Plot Views	10-29
10.9	Visualization Views	10-32

## 11 Optimization

11.1	Overview	11-1
11.2	Specification Block	11-2
11.3	Simulation Types	11-5
11.4	Parameter Sweeps	11-9
11.5	Parameter Labels	11-15
11.6	Output Labels and Goals	11-16
11.7	Error Functions and Weights	11-19
11.8	FUN Datapipe	11-21
11.9	FDF Datapipe	11-24
11.10	Objective Functions	11-27
11.11	OSA90 Optimize Menu	11-32
11.12	Trace of Optimization Variables	11-45
11.13	Sensitivity Analysis	11-46
11.14	Space Mapping Optimization	11-51
11.15	Technical References	11-58

## 12 Statistical Analysis

12.1	Overview	12-1
12.2	Statistical Parameters	12-2
12.3	Uniform Distribution	12-3
12.4	Exponential Distribution	12-4
12.5	Lognormal Distribution	12-5
12.6	Normal Distribution	12-6
12.7	Sample Distribution	12-8
12.8	Statistics Block and Correlations	12-11
12.9	User-Created Hybrid Distributions	12-14
12.10	MonteCarlo Block	12-15
12.11	Simulation Types	12-18
12.12	Parameter Sweeps	12-22
12.13	Parameter Labels	12-28
12.14	Output Labels and Goals	12-29
12.15	OSA90 MonteCarlo Menu	12-32
12.16	Monte Carlo Display Options	12-35
12.17	Monte Carlo Xsweep Display	12-37
12.18	Monte Carlo Parametric Display	12-39
12.19	Histograms	12-41
12.20	Run Charts	12-43
12.21	Yield	12-44
12.22	Yield Sensitivity	12-45

## Table of Contents

12.23	Display of Maximum Errors	12-49
12.24	Scatter Diagrams	12-51
12.25	Views for Monte Carlo Displays	12-53
12.26	Histogram Views	12-56
12.27	Run Chart Views	12-58
12.28	Scatter Views	12-60

## 13 Yield Optimization

13.1	Overview	13-1
13.2	Yield Optimization Options	13-2
13.3	Restart Yield Optimization	13-7
13.4	Technical References	13-8

## 14 OSA90 as Datapipe Child

## 15 Auxiliary Files

15.1	User's Manual Files	15-1
15.2	Log Files	15-2

## 16 Examples

16.1	Overview	16-1
16.2	Example demo01	16-8
16.3	Example demo02	16-11
16.4	Example demo03	16-14
16.5	Example demo11	16-19

## Appendix A Datapipe Server

## Appendix B Diagnostic Messages

## Index

# 1

## Installation

<b>1.1 Introduction</b> .....	1-1
<b>1.2 Installing the Software</b> .....	1-2
<b>1.3 Installing the Hardware Lock</b> .....	1-3
<b>1.4 OSA90 Example Directory</b> .....	1-4
<b>1.5 Uninstalling the software</b> .....	1-5



# 1

## Installation

### 1.1 Introduction

Throughout this Manual, OSA90/hope is referred to as OSA90.

The installation of OSA90 can be done by either a system administrator or a user.

In a Windows NT multi-user environment, the system administrator should install OSA90 so that it is accessible to all users. If, on the other hand, OSA90 is to be used by a single user, then that user can carry out the installation himself/herself.

Sections 1.2 and 1.3 describe the procedure of installing the software and hardware lock respectively. Only the person who performs the installation needs to follow this procedure.

If you experience difficulties in the installation of OSA90, you can contact our technical support staff at

Tel 905 628 8228  
Fax 905 628 8225

Email [osa@osacad.com](mailto:osa@osacad.com)

Home page <http://www.osacad.com>

## 1.2 Installing the Software



We recommend that you close all open applications before starting the installation. If there were additional installation instructions provided with the software, please follow them as well.

Insert the floppy disk labelled "Installation Disk 1" in the appropriate drive. Click on the "Start" menu (located on the taskbar) and choose "Run". A dialog box will appear, prompting you for a file location. Enter the following string:

```
a:\setup.exe
```

where "a:" is the drive letter of the 3.5" floppy drive.

The installation program will guide you through the actual installation of OSA90.

After setup has been successfully completed you will see a dialog box which provides directions for installing the Sentinel Pro drivers. OSA90 requires these drivers to access the information stored in the hardware lock to function properly.

The software portion of the setup procedure is now complete.

## 1.3 Installing the Hardware Lock

Provided with the software is a Sentinel Pro hardware lock. The hardware lock should be installed on the first parallel port of the computer (i.e., LPT1). If there are other locks already attached to the computer, be sure to place the Sentinel Pro at the end of the chain. This lock is completely transparent and should not interfere with normal operation of the computer, printers or other devices which may be attached to the computer's parallel port..

After installing the hardware lock, insert the floppy disk labelled either "Win95 Sentinel Setup" or "WinNT Sentinel Setup" (whichever is appropriate for your operating system) in the drive. Click on the "Start" menu (located on the taskbar) and choose "Run". A dialog box will appear, prompting you for a file location. Enter the following string:

```
a:\setup.exe
```

where "a:" is the drive letter of the 3.5" floppy drive.

You will be presented with a screen titled "Sentinel Driver Setup Program". You will see that there is only one menu item available, "Functions". From "Functions", choose "Install Sentinel Driver". You will be prompted for the location of the installation files. The location is the root directory of the installation floppy disk (for example, "a:\").

After a brief period of time, you will be presented with a dialog box informing you of the outcome of the installation and requesting that you reboot your machine. Prior to rebooting the machine, remove the installation floppy disk from the drive.

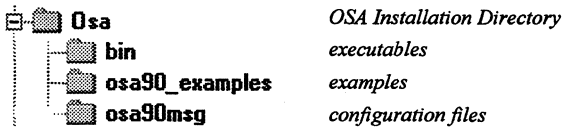
After rebooting, OSA90 is ready to run.



There have been problems reported involving hardware locks attached to enhanced or bidirectional parallel ports. If you experience problems with the hardware lock, please see the documentation for your computer and set the port to "standard" or "normal" mode.

## 1.4 OSA90 Example Directory

The directory hierarchy of the OSA90 installation is as follows.



We recommend that before using the software, you make a working copy of the examples to a suitable directory and work with that copy.

If you need more information on how to copy files and folders, select “Help” from the “Start” menu (located on the taskbar) and search for “copying”.



## 1.5 Uninstalling the Software

To uninstall OSA90 and its components, perform the following actions:

- Select “Control Panel” from the “Settings” sub-menu of the “Start” menu.
- Open “Add/Remove Programs”.
- Select the software you wish to uninstall.
- Click the “Add/Remove” button.
- Click “Yes” when the “Confirm File Deletion” dialog box appears.

A message box will appear, informing you whether or not the software was successfully removed.



# 2

## Technical Overview

<b>2.1 Overview</b> .....	2-1
<b>2.2 OSA90 Window</b> .....	2-5
<b>2.3 OSA90 User Interface</b> .....	2-6
<b>2.4 OSA90 Input File and File Editor</b> .....	2-10
<b>2.5 Simulation and Optimization</b> .....	2-11
<b>2.6 A Practice Session</b> .....	2-12



## 2

## Technical Overview

### 2.1 Overview

**OSA90/hope™** is a state-of-the-art, general purpose CAD software system offering simulation, modeling, statistical analysis, nominal and yield optimization, and data visualization for linear and nonlinear RF/microwave analog circuits. Its open architecture allows you to create fully optimizable interconnections of components, subcircuits, simulators and mathematical functions. OSA90/hope is equipped with several protocols for connecting external programs through interprocess pipes. This facilitates high-speed data connections to external *executable* programs.

#### Powerful Circuit Simulation

- ▶ linear and nonlinear simulation of general n-port analog circuits with an arbitrary number of nonlinear devices and excitations
- ▶ analytically unified DC, small-signal AC and large-signal harmonic balance analyses
- ▶ comprehensive nonlinear device and linear element model library
- ▶ customized models created by the user from nonlinear controlled current and charge sources with arbitrary controlling voltages
- ▶ linear subcircuits and symbolic nonlinear subcircuits
- ▶ user-created linear element models as fully parameterized subcircuits
- ▶ calculation of DC and AC voltages and currents, small-signal scattering parameters, large-signal harmonic distortion, compression, intercept points, intermodulation products, insertion loss, stability factor, group delay and more
- ▶ design of small-signal amplifiers, power amplifiers, filters, mixers, frequency multipliers and oscillators

## State-of-the-Art Optimization

- ▶ gradient-based minimax,  $\ell_1$ ,  $\ell_2$ , quasi-Newton, conjugate gradient and Huber optimizers
- ▶ non-gradient simplex, random and simulated annealing optimizers
- ▶ Space Mapping™ optimization
- ▶ statistical yield optimization
- ▶ efficient FAST™ sensitivities for nonlinear circuits
- ▶ display of large-change sensitivities, including yield sensitivities
- ▶ arbitrary variables, functions, specifications and weights

## Sophisticated Mathematics

- ▶ formulation of functions and equations by arbitrary combinations of constants, variables, labels, algebraic operations and mathematical functions
- ▶ vector and matrix definitions and expressions, including built-in functions for matrix algebra, inverse, LU factorization, eigenvalues and eigenvectors
- ▶ built-in transformations for complex numbers: polar form to/from rectangular form
- ▶ built-in DFT transformations: time domain to/from frequency domain
- ▶ built-in cubic spline interpolation, including two-dimensional bicubic splines
- ▶ conditional expressions (if and else)
- ▶ uniform, normal, exponential, lognormal and sample statistical distributions with absolute or relative tolerances and user-definable correlation matrices
- ▶ quadratic approximations for efficient statistical optimization

## Intelligent Connectivity

- ▶ unique Datapipe™ for high-speed data communications to and from external programs
- ▶ functional integration of separate in-house software with flexible pre-, post- and inter-processing of inputs and outputs
- ▶ adding statistical and optimization capabilities to other CAD programs such as time-domain, field and structural simulators

- ▶ optimization using externally calculated gradients via Datapipe
- ▶ importing linear subcircuits from external simulators
- ▶ OSA90 calling OSA90 through Datapipe to create a virtual simulation hierarchy of unlimited depth

## EM Simulation and Optimization

- ▶ proprietary Empipe™ connection to *em* (by Sonnet Software, Inc.) for electromagnetic simulation and optimization of predominantly planar structures
- ▶ proprietary Empipe3D™ connection to Ansoft HFSS and HP HFSS (offered by Ansoft Corporation, and Hewlett-Packard Co., respectively) for electromagnetic simulation and optimization of 3D structures
- ▶ Geometry Capture™ for optimization of arbitrary structures
- ▶ automatic discretization and interpolation of geometrical parameters, and efficient database management to minimize the number of calls to the EM solver
- ▶ accurate EM simulation results seamlessly embedded into OSA90 circuit definition

## Friendly Environment

- ▶ logically designed menus and windows, easy to learn and use
- ▶ integrated screen editor and input file parser
- ▶ arbitrary parameter sweeps, parametric plots, waveform displays, Smith chart and polar plots, Monte Carlo sweeps, histograms and run charts
- ▶ quality graphics with zooming
- ▶ 3D visualization and contour plotting with rotating, scaling and smoothing
- ▶ view definitions allowing customized graphical displays
- ▶ user-formatted report generation embedding simulation results in arbitrary text

## Comprehensive Documentation

- ▶ on-line User's Manual
- ▶ more than 80 demo examples on a wide range of applications

## OSA Products and Services

Optimization Systems Associates Inc. has been developing state-of-the-art CAE software and technologies since 1983. OSA90 offers you the opportunity to take advantage of OSA's long-standing and continually evolving CAD/CAE technologies.

HarPE™ is a well established system dedicated to complete nonlinear device CAD, including device characterization, device simulation and optimization, as well as statistical modeling and analysis.

Empipe is a powerful and friendly software system for automated electromagnetic (EM) design optimization, driving the EM simulator *em*™ from Sonnet Software, Inc. Empipe allows you to designate geometrical and material parameters as candidate variables for optimization in an intuitive and friendly manner. Any arbitrary structures that you can simulate using *em* you can now optimize using Empipe!

Empipe3D is a powerful and friendly software system for automated EM optimization, driving the EM simulators Ansoft HFSS from Ansoft Corporation and HP HFSS offered by Hewlett-Packard Co. Empipe3D allows you to designate geometrical and material parameters as candidate variables for optimization in an intuitive and friendly manner. Any arbitrary structures that you can simulate using HFSS you can now optimize using Empipe3D!

OSA also provides consulting services for specialized CAE solutions.



For further information:

Tel 905 628 8228

Fax 905 628 8225

Email [osa@osacad.com](mailto:osa@osacad.com)

Home Page <http://www.osacad.com>



## 2.2 OSA90 Window

```

OSA90 V4.0.0 - bend1.ckt
File Edit Display Optimize NetworkCable Help
! Empipe3D user-defined structure BEND1
Model
#include "bend1_osa\bend1.inc";

BEND1_d: 70 0.2 0.357;

BEND1 1 2 0 model=7
d=(BEND1_d * 1in);

PORTS 1 0 2 0;

CIRCUIT;

MS_DB[2,2] = if (MS > 0) (20 + log10(MS)) else (NaN);
MS11_DB = MS_DB[1,1];
end

Sweep
AC: FREQ: from 9GHz to 15GHz step=1GHz MS11_dB
{XSWEPT title="MS11_dB and Spec" X-FREQ V=MS11_dB
SPEC=(from 9GHz to 15GHz, < -30) &
(from 9GHz to 15GHz, < -30)};

AC: FREQ: from 9GHz to 15GHz step=1GHz MS MS_DB PS
{Smith MP=(MS11,PS11).S11}
{Polar MP=(MS21,PS21).S21};
end

Spec
AC: FREQ: from 9GHz to 15GHz step=1GHz MS11_dB < -30;
AC: FREQ: from 9GHz to 15GHz step=1GHz MS11_dB < -30;
end

Control
Perturbation_Scale=1.0e-4;
Optimizer=Minimax;
File Parsing Completed

```

Fig. 2.1 OSA90 window.

The top of the window contains the menu bar and toolbar and the middle portion of the window is used to display graphics or ASCII files (the input file, or netlist is shown here). The status bar is located at the bottom of the window.

## 2.3 OSA90 User Interface

Near the top of the OSA90 window is the menu area, where the menu options are presented.

You can use the mouse to highlight the different menu options. As you do so, the status bar displays a brief comment on the function of the highlighted option.

To select a menu option, you can click the left-hand mouse button on the desired option.

Some menu options lead to another level of menu options. The OSA90 menu hierarchy is summarized in Table 2.1, described in greater detail in Table 2.2 and toolbar button associations are listed in Table 2.3.

TABLE 2.1 OSA90 MENU OPTIONS

Menu Option	Brief Description
File	reads, edits, parses and saves files
Edit	provides access to cut, copy and paste
Display	calculates and displays responses and functions
Optimize	initiates optimization
MonteCarlo	performs statistical (Monte Carlo) analysis
Help	where to go when you need help
















TABLE 2.2 MENU FUNCTIONS

Option	Brief Description
<b>FILE</b>	
➤Open	opens an existing file
➤Save	saves the current input file
➤Save As	saves the current input file under a different name
➤New	creates a new input file
➤Edit Input File	returns to editing mode
➤Compile Input File	compiles the input file and checks for syntax errors
➤Print	prints the current contents of the OSA90 main window
➤Online Manual	displays the online manual
➤Exit	exits OSA90
<b>EDIT</b>	
➤Undo	undoes the last edit or change to the input file
➤Cut	deletes selected text and places it on the clipboard
➤Copy	copies selected text to the clipboard
➤Paste	places the contents of the clipboard at the current cursor location
➤Select All	selects the text of the entire file
➤Find	searches the file for the specified text
➤Replace	replaces specified text with alternate text
<b>DISPLAY</b>	
➤Xsweep	displays responses versus parameter sweeps
➤Parameteric	displays parametric plots of responses
➤Array	displays elements of arrays
➤Waveform	displays time-domain waveforms
➤SmithChart	displays Smith charts and polar plots
➤Visual	displays 3D visualization and contours
➤Report	generates a report file

TABLE 2.2 (cont'd) MENU FUNCTIONS

Option	Brief Description
<b>OPTIMIZE</b>	selects and runs one of available optimization routines
↳ L1	
↳ L2	
↳ Minimax	
↳ Quasi-Newton	
↳ Conjugate Gradient	
↳ Huber	
↳ One-Sided L1	
↳ Minimax AG	
↳ Random	
↳ Simplex	
↳ Simulated Annealing	
↳ Space Mapping	
↳ Yield	
↳ Sensitivity Analysis	invokes sensitivity analysis
<b>MONTECARLO</b>	
↳ Xsweep	displays statistical sweep responses
↳ Parametric	displays statistical parametric plots
↳ Histogram	displays histogram of individual responses
↳ Run Chart	displays run chart of individual responses
↳ Yield	displays yield estimated by Monte Carlo analysis
↳ Sensitivity	displays yield versus parameter sweeps
↳ Max Error	displays histogram of the maximum errors
↳ Scatter Plot	displays scatter diagram between two responses
<b>HELP</b>	
↳ Help Topics	launches Online Manual
↳ About OSA90	displays license information and serial number

TABLE 2.3 TOOLBAR BUTTONS

Menu Option	Hot Key	Toolbar Button
Open	Ctrl + O	
Save	Ctrl + S	
New	Ctrl + N	
Edit Input File	Ctrl + E	
Compile Input File	F7	
Print	Ctrl + P	
Online Manual	F1	
Cut	Ctrl + X	
Copy	Ctrl + C	
Paste	Ctrl + V	
Find	Ctrl + F	
Display		
Optimize		
Sensitivity		
MonteCarlo		

## 2.4 OSA90 Input File and File Editor

OSA90 input files are ASCII text files. The default extension for OSA90 input files is ".ckt".

The input file contains definitions of parameters, variables, labels, models, responses, equations, frequency ranges, parameter sweeps, simulation outputs, optimization specifications, operation control options, statistical tolerances and distributions, etc.

The structure and syntax of OSA90 input files are discussed in Chapter 3.

OSA90 has a built-in full screen ASCII text file editor. It allows you to create, retrieve, modify and save input files for OSA90.

The screen editor features search and replace, cut and paste, undo, printing, and more. The editor is integrated with the file parser to make syntax error detection and correction convenient and friendly. Functions listed under the "File" and "Edit" menus above pertain to the editing and compilation of the input file.

## 2.5 Simulation and Optimization

OSA90 simulation is invoked by the “Display” menu option. The various options listed under the “Display” menu allow you to view the circuit responses in a variety of formats.

TABLE 2.4 DISPLAY FORMATS

Option	Brief Description
Xsweep	displays one or more responses versus frequency or a parameter
Parametric	displays one or more responses versus another response
Array	displays elements of an array
Waveform	displays time-domain waveforms
Smith	displays Smith charts and polar plots
Visual	displays 3D visualization and contours

OSA90 optimization is invoked by the “Optimize” menu option. The choices available for the “Optimize” menu option are listed in Table 2.2 and are further described in Chapter 11.

For more information on simulation and optimization, please see Chapters 9, 10 and 11.

## 2.6 A Practice Session

This practice session guides you, step by step, through the basic operations of OSA90 using a simple example.

### Copying the Tutorial Examples

A set of OSA90 examples are provided to you in the subdirectory

*<OSA Installation Directory>/osa90\_examples*

Before proceeding with this practice session, we strongly recommend that you make a copy of the example files to a suitable location as set out in Chapter 1.

### Starting OSA90

To start OSA90, click on the “OSA90” icon located in the “Osa” program group.

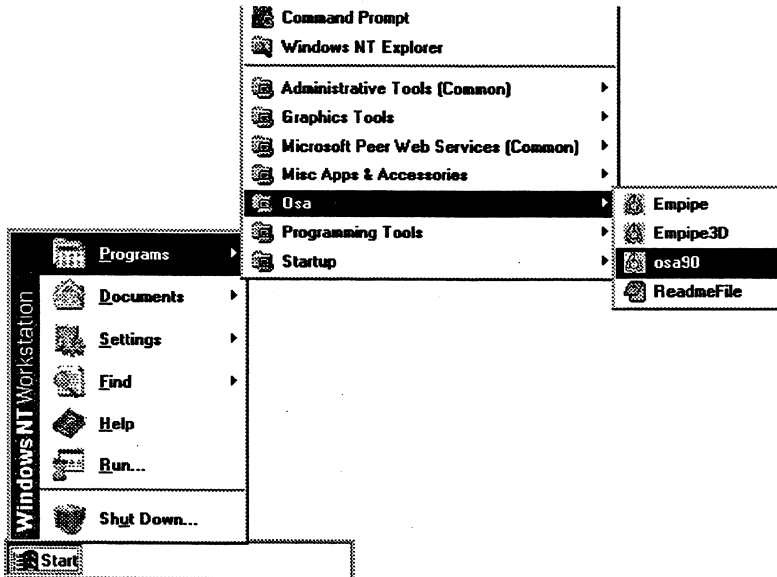


Fig. 2.2 Location of OSA90 on the Start Menu



The OSA90 main window will appear on screen, along with the “OSA90 Open File” dialog box.

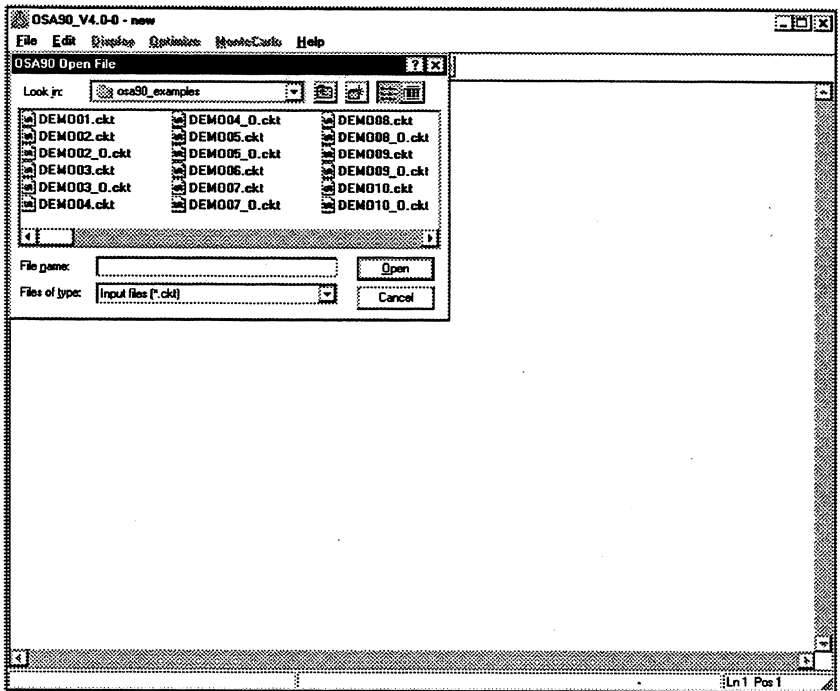


Fig. 2.3 The “OSA90 main window and Open File dialog”

For this practice session, we need to read in the file named "practice.ckt".

## OSA90 File Editor

The contents of the file "practice.ckt" are displayed in the OSA90 file editor screen, as depicted in Fig. 2.4. You can browse through the file, but be careful not to inadvertently alter the contents of "practice.ckt".

The screenshot shows a window titled "OSA90\_V4.0.0 - practice.ckt". The menu bar includes "File", "Edit", "Display", "Optimize", "MonteCarlo", and "Help". Below the menu is a toolbar with various icons. The main text area contains the following circuit description:

```

! Example practice.ckt
! key: minmax, optimization, small-signal, amplifier.

Model
TEM 1 0 2=70 E=?125? F=10GHZ;
TEM 2 0 2=50 E=?80? F=10GHZ;

TEM 1 3 0 4 2=100 E=65 F=10GHZ;
OPEN 3 4;

TEM 1 5 2=100 E=25 F=10GHZ;

CAP 5 6 C=10NF;
IND 6 7 L=100NH;

Extrinsic2 8 9 10 6 11
RG=3.5 LC=0.0306NH RD=0.5 LD=0.00792NH
RS=4.73 LS=0.0451NH GDS=0.00345 CDS=0.00738PF CX=10PF;

FETR 8 9 10
IS=5E-15 N=1 FC=0.5 GMIN=1.0E-07
UBI=0.8 UBR=20.0 ALPHA=2 THETA=0.003
BETA=0.029 UTO=-1.637 LAMBDA=0.04978 TAU=2.8PS
CGS0=0.4428PF CGD0=0.1066PF;

CAP 11 12 C=10NF;
IND 11 13 L=100NH;
    
```

The status bar at the bottom right of the window shows "Ln 1 Pos 1".

Fig. 2.4 OSA90 file editor screen.

## The Circuit Model

The OSA90 input file contains the circuit model, the frequency range for simulation, the responses of interest, the optimization specifications and other relevant information.

The file "practice.ckt" describes a small-signal amplifier, as depicted in Fig. 2.5.

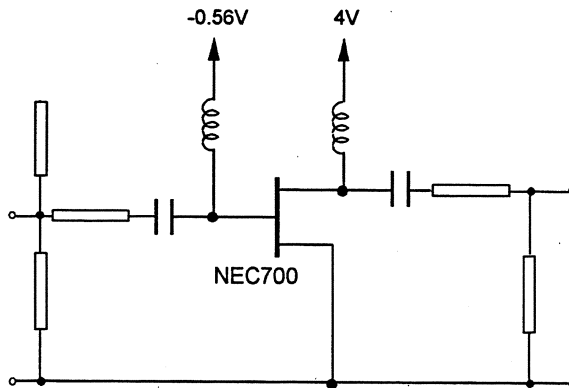


Fig. 2.5 The small-signal amplifier.

## Optimization Variables and Specifications

Near the beginning of the input file "practice.ckt", you can find these statements:

```
TEM 1 0 Z=70 E=?125? F=18GHZ;
TEM 2 0 Z=50 E=?80? F=18GHZ;
```

Each statement defines a transmission line model with the parameters Z (characteristic impedance), E (electrical length) and F (frequency at which E is given). Notice that the values for the parameter E are enclosed within question marks. This indicates that they are optimizable parameters. In other words, when we invoke optimization, the values enclosed within question marks will be adjusted by the program in an attempt to improve the circuit performance with respect to a set of specifications. Further discussions on optimization variables and bounds can be found in Chapter 4.

Near the end of the input file you can find these statements:

```
Spec
AC: FREQ: from 6 to 18 Step=1 MS21_DB > 7 MS21_DB < 9;
end
```

These statements represent the specifications on the small-signal gain of the amplifier:

$$7 \leq 20 \cdot \log_{10}(|S_{21}|) \leq 9 \quad \text{for } 6 \text{ GHz} \leq \text{frequency} \leq 18 \text{ GHz}$$

For details on defining specifications, see Chapter 11.

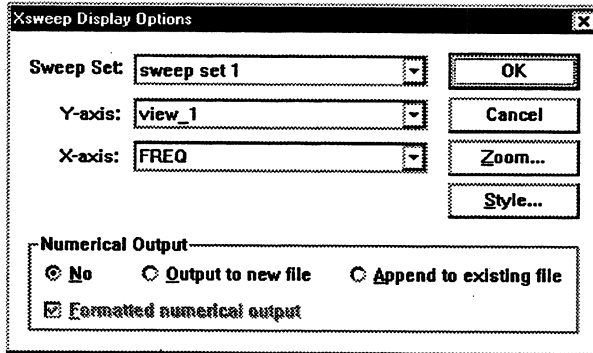
## Compiling the Input File

Before you can perform any display or optimization operations, you must first compile the input file.

When compiling the input file OSA90 checks the syntax of the file and will locate and highlight any errors it detects. To compile the input file, either choose "Compile Input File" from the "File" menu, hit the <F7> key, or click on the corresponding toolbar button.

## The Display Menu

The “Display” menu should now be active. Click on “Display” and select “Xsweep” from the list of options that drops down. A dialog box will appear:



Press <ENTER> to accept the default setting (the display options are explained in Chapter 9). You should see the display shown in Fig. 2.6. It shows both the response and the specifications. The specifications are not satisfied.

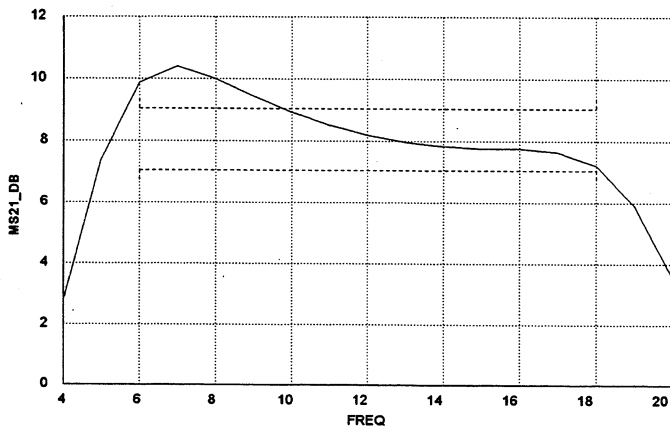


Fig. 2.6 The small-signal amplifier gain.

## Performing Minimax Optimization

Choose "Minimax" from the "Optimize" menu option. A dialog box appears:

**Minimax Optimization Options**

Number of iterations: 30

Accuracy of solution: 0.0001

Show downhill iterations only

OK

Cancel

Click on "OK" (alternatively you can press <ENTER>). This starts the optimization.

The progress of optimization is reported on the screen:

```

Iteration 1/30 Max Error=1.35546
Iteration 2/30 Max Error=1.27626
Iteration 3/30 Max Error=1.11658
Iteration 4/30 Max Error=0.792285
Iteration 5/30 Max Error=0.124942
Iteration 6/30 Max Error=-0.543871
Iteration 7/30 Max Error=-0.573292
Iteration 8/30 Max Error=-0.577194
Iteration 9/30 Max Error=-0.577254
Solution Max Error=-0.577254
    
```



The numerical values you actually see may be slightly different from those shown here, due to differences in the computer hardware and/or software versions.

## Simulating the Optimized Solution

After the optimization is finished, choose "Xsweep" from the "Display" menu. After the simulation is completed the "Xsweep Display Options" dialog box appears, press <ENTER> to accept the default setting.

The optimized amplifier response is shown in Fig. 2.7. Clearly, the specifications are satisfied after the optimization.

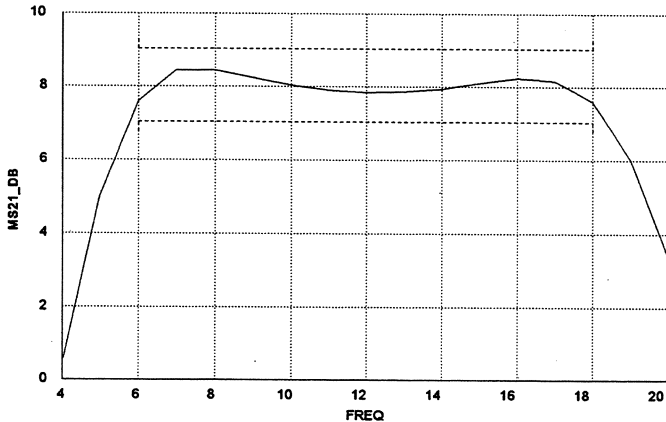


Fig. 2.7 The small-signal amplifier gain after optimization.

## Back Annotation of the Optimization Solution

The solution of the optimization is automatically back-annotated into the input file. To see it, choose "Edit Input File" from the "File" menu (or press <Ctrl + E>).

Within the file editor, you can locate near the beginning of the input file these statements:

```
TEM 1 0 Z=70 E=?126.958? F=18GHZ;
TEM 2 0 Z=50 E=?61.4634? F=18GHZ;
```

The optimizable parameter values within the question marks have been updated.



The numerical values you actually see may be slightly different from those shown here, due to differences in the computer hardware and/or software versions.

## Exiting the OSA90 Program

Choose "Exit" from the "File" menu. A message box will appear asking you whether you wish to save the changes. This alerts us to the fact that we did not save the input file with the optimized parameter values. Since this is only a practice session, we do not need to save the file. Click on "No".

Another message box will appear, requesting confirmation that you wish to exit the program. Click on "Yes".



# 3

## Input File

<b>3.1 Overview</b> . . . . .	3-1
<b>3.2 Keywords</b> . . . . .	3-1
<b>3.3 Input File Blocks</b> . . . . .	3-2
<b>3.4 Input File Templates</b> . . . . .	3-3
<b>3.5 Statements</b> . . . . .	3-5
<b>3.6 Preprocessor Macros</b> . . . . .	3-6
<b>3.7 Include Files</b> . . . . .	3-12
<b>3.8 Control Block</b> . . . . .	3-13
<b>3.9 Physical Units</b> . . . . .	3-26



## 3

# Input File

## 3.1 Overview

An input file is an ASCII text file prepared by the user to supply OSA90 with the essential information needed for simulation and optimization.

The input file defines parameters and variables, circuit topology and models, responses and functions, formulas and equations, simulation types and ranges, optimization goals and specifications, statistical tolerances and distributions, etc.

The contents of an input file are divided into sections called file blocks. Each file block is designated to supply information related to a particular aspect of the operation.

This chapter outlines the basic structure of the input file and describes the syntax rules that are applicable throughout the input file. Details of the various file blocks are given in subsequent chapters when the appropriate features are discussed.

**Extension .ckt is recommended** for all OSA90 input files. Some efficiency-enhancing features apply only to input files with the .ckt extension (see Chapters 9, 11 and 12).

The input file size is limited to 256K bytes by default. This limit can be extended if necessary (please contact the technical support at OSA).

## 3.2 Keywords

Keywords are names designated by OSA90 to denote built-in features and options.

For example, in the input file, a resistor is defined as

```
RES 1 2 R = 5.0;
```

where RES is the keyword representing the built-in resistor element model, and R is the keyword representing the resistance parameter of the resistor element model.

**Keywords are case insensitive.** For example, RES, res and Res are treated by the file parser as identical keywords.

### 3.3 Input File Blocks

The contents of an input file are divided into sections called file blocks. Each file block is designated to supply information related to a particular aspect of the operation.

#### Syntax

*Block\_Name*

...

End

where *Block\_Name* represents one of the keywords listed in Table 3.1.

TABLE 3.1 INPUT FILE BLOCKS

Block Name	Contains Definitions of	Required for	Chapter
CONTROL	user-controllable default options	-	3
SAMPLE	sample of statistical outcomes	-	12
STATISTICS	statistical correlation coefficients	-	12
IMPORTDATA	imported linear subcircuit data	-	6
EXPRESSION	labels, variables, functions	-	4,5
MODEL	circuit models, circuit responses	circuit simulation	6,7,8
SWEEP	simulation types, ranges, outputs	simulation/display	9,10
SPECIFICATION	design specifications, weights	optimization	11
MONTECARLO	statistical analysis ranges, outputs	statistical analysis	12
REPORT	text report template	-	9
TRACE	record of optimization variables	-	11

**Block names are case insensitive**, like all other keywords. For example, MODEL, Model and model all identify the same file block.

**Block names can be abbreviated** to the first four or more characters. For example, you can use Spec as the name for the Specification block.

**All blocks are optional.** You only need to define those file blocks that are necessary for the desired operations, as noted in Table 3.1 in the "Required for" column. For instance, you need to define the Specification block in the input file only if you wish to perform optimization.

**Order of entry is arbitrary.** The file blocks may appear in the input file in any order, although the file parser always processes the blocks in the order given in Table 3.1.

## 3.4 Input File Templates

This section contains a number of templates for creating OSA90 input files. The templates are designed for several typical combinations of features. In each case, the *minimal* configuration of file blocks necessary to set up a problem properly is shown.

The templates contain only an outline of the input file. Details of the relevant file blocks are discussed in the appropriate chapters as listed in Table 3.1.

☞ See also the demo examples in Chapter 16 and *OSA90/hope Applications Illustrated*.

### Generic Numerical Simulation

```

Expression
  define parameters and functions (Chapters 4,5)
End

Sweep
  define simulation ranges and outputs (Chapter 9)
End

```

### Generic Numerical Simulation and Optimization

```

Expression
  define parameters, variables and functions (Chapters 4,5)
End

Sweep
  define simulation ranges and outputs (Chapter 9)
End

Specification
  define optimization ranges, specifications and weights (Chapter 10)
End

```

**Circuit Simulation**

```

Model
    define circuit topology, models, excitations and responses (Chapters 6,7,8)
    define formulas, functions and postprocessed responses (Chapters 4,5)
End

Sweep
    define simulation types, ranges and outputs (Chapter 9)
End

```

**Circuit Simulation and Optimization**

```

Model
    define circuit topology, models, excitations and responses (Chapters 6,7,8)
    define formulas, functions and postprocessed responses (Chapters 4,5)
End

Sweep
    define simulation types, ranges and outputs (Chapter 9)
End

Specification
    define optimization types, ranges, specifications and weights (Chapter 10)
End

```

**Statistical Analysis and Optimization**

```

Statistics
    define statistical correlation coefficient matrices (Chapter 11)
End

Model
    define statistical parameters (Chapter 11)
    define circuit models, responses and other functions (Chapters 4,5,6,7,8)
End

MonteCarlo
    define statistical simulation types, ranges and outputs (Chapter 11)
End

Specification
    define optimization types, ranges, specifications and weights (Chapter 10)
End

```

## 3.5 Statements

The logical unit of the information within an input file block is called a statement.

Statements are delimited (separated) by a semicolon ";".

A statement may occupy one text line:

```
RES 1 2 R = 5.0;
```

Or, a long statement may occupy several text lines:

```
Value_DB:  if (Value > Value_Max) (20.0 * log10(Value_Max))
            else (if (Value < Value_Min) (20.0 * Log10(Value_Min))
            else (20.0 * log10(Value)));
```

On the other hand, several short statements may share a single text line:

```
V1 = 50;   V2 = ?1.5?;   V3 = (V1 + V2) / 2;
```



Since a statement does not always correspond to one text line, make sure it is properly terminated by a semicolon. Otherwise information from different lines can get mixed up.

**Comments** can be included in the input file following the exclamation mark "!". The file parser will skip over any text following an exclamation mark "!" until the end of the line.

You can have full lines of comments, such as

```
! File: demo01.ckt
! OSA90 input file for demonstration
```

or in-line comments, such as

```
d = sqrt(X * X + Y * Y);      ! Euclidean norm
```

**Blank Lines** can be used to space the input file text for clarity.

## 3.6 Preprocessor Macros

OSA90's file parser has a built-in preprocessor. As the first phase of file parsing, the input file is scanned for preprocessor directives which are marked by the number sign (hash) "#". The preprocessor directives include text macros and include files.

Text macros are created by the `#define` directive to represent lengthy, complex and/or repetitive portions of text. At parsing time, references to text macros are expanded into the full text the macros represent. The use of text macros can make the input file more readable and easier to modify.

The preprocessor macros in OSA90 are quite similar to those of the "C" language, except

- ▷ In OSA90, text macro names are case insensitive unless enclosed within quotation marks.
- ▷ In OSA90, macros apply to the whole input file regardless of the location where the macros are defined. In other words, the file parser recognizes references to a macro even if such references appear *before* the macro is defined. This is necessary because, as described in Section 3.3, the input file blocks may be rearranged by the file parser into a certain order.
- ▷ OSA90 allows two alternative styles for continuation lines.
- ▷ OSA90 implements a special "\$" argument.

### Macro Constants

#### Syntax

```
#define macro_name substitution_text
```

Example:

```
#define Boltzmann_Constant 1.38066E-23
```

During preprocessing, the file parser will replace all references to *macro\_name* in the input file with *substitution\_text*. In the example, the substitution text is "1.38066E-23". The file preprocessor will replace all references to `Boltzmann_Constant` with "1.38066E-23".

Macro names are case insensitive unless enclosed within double quotation marks. For example, `MACRO1`, `macro1` and `Macro1` are indistinguishable as macro names, whereas "macro1" and "Macro1" would be considered as different and distinct names.

Macros are useful for associating critical constants with meaningful names, such as the Boltzmann constant.



**Aliases for built-in keywords** can be created using macros. For instance, the macro

```
#define Resistor RES
```

creates an alias for the built-in resistor element keyword RES. Using this alias, you can write

```
Resistor 1 2 R = 5.0;
```

which the file parser will translate into the standard form of

```
RES 1 2 R = 5.0;
```

## Macro Expressions

### Syntax

```
#define macro_name substitution_text
```

Example:

```
#define AREA LENGTH * WIDTH
```

Here, the substitution text is an expression rather than a constant.

**Recursive references**, i.e., macros which contain references to other macros, are permissible. For example,

```
#define VOLUME AREA * HEIGHT
```

where the substitution text includes a reference to the macro AREA defined above.



Care must be exercised with recursive references to macro expressions. You may have to use parentheses to preserve the proper precedence of algebraic operations.

Example:

```
#define X_SUM X1 + X2
```

```
#define Y2 Y1 * X_SUM
```

Y2 will be expanded into

```
Y1 * X1 + X2
```

which is unlikely to be the intended result.

A proper definition of Y2 should be

```
#define Y2    Y1 * (X_SUM)
```



Remember that the preprocessor does not actually evaluate the macro expressions but merely carries out text substitutions.

## Macro Functions

Macro functions are the most sophisticated form of preprocessor macros. In addition to a name for identification, a macro function is defined with a set of arguments and the substitution text is typically defined with dependence on the arguments.

### Syntax

```
#define macro_name(arguments)  substitution_text
```

Example:

```
#define MAGNITUDE(REAL,IMAG)  sqrt(REAL*REAL + IMAG*IMAG)
```

The opening parenthesis of the argument list must immediately follow the macro name. No space can separate the macro name from the opening parenthesis, otherwise the parenthesis and the arguments would be mistaken as part of the substitution text. Multiple arguments must be separated by commas.

**Formal and actual arguments:** the arguments listed in the macro definition are called the formal arguments; each reference to a macro function must supply a set of actual arguments to match the list of formal arguments. The formal arguments which appear in the substitution text will be replaced by the actual arguments. In other words, for each reference, a specific version of the substitution text is generated according to the actual arguments.

For example, the following reference to the macro function shown above

```
MAGNITUDE(x1,x2)
```

will be expanded during preprocessing to

```
sqrt(x1*x1 + x2*x2)
```

**Recursive references in actual arguments**, i.e., the actual arguments for a macro function which contain references to other macros, are permissible. For example, consider a macro function defined as

```
#define MULTI(X1,X2)    X1 * X2
```

and the following reference to this macro function:

```
MULTI(MAGNITUDE(x1,x2), x3)
```

where the actual arguments refer to another macro function **MAGNITUDE** which is defined earlier. This reference will be expanded to

```
sqrt(x1*x1 + x2*x2) * x3
```

after preprocessing.



Care must be exercised with recursive macro references. You may have to use parentheses to preserve the proper precedence of algebraic operations.

Example:

```
#define SUM(X1,X2)    X1 + X2
#define MULTI(X1,X2) X1 * X2

MULTI(SUM(x1,x2), x3)
```

The macro reference will lead to erroneous results after preprocessing:

```
x1 + x2 * x3
```

You can avoid this kind of problem by using parentheses either in the macro definition, such as

```
#define SUM(X1,X2)  (X1 + X2)
```

or, in the actual arguments

```
MULTI((SUM(x1,x2)), x3)
```

or both.

**Formal arguments are local identifiers** within the macro functions, since they are merely placeholders for the actual arguments. For instance,

```
#define MAGNITUDE(REAL,IMAG)  sqrt(REAL*REAL + IMAG*IMAG)
```

the formal arguments **REAL** and **IMAG** are local identifiers recognized only within the macro function **MAGNITUDE**.

Hence, you can assign the same names to the formal arguments of different macro functions without causing conflicts.

## Continuation Lines for Long Substitution Text

The substitution text of some macros may be longer than one line. Similar to the "C" language, OSA90's preprocessor allows long text to be continued onto the next line by ending the current line with a backslash "\":

```
#define EXP_SAFE(X)  if (X > X_MAX) (exp(X_MAX))  \
                    else (if (X < X_MIN) (0))      \
                    else (exp(X))
```

Also, OSA90 implements an alternative style for continuation lines: using a pair of curly brackets to enclose the lines, such as

```
#define EXP_SAFE(X) { if (X > X_MAX) (exp(X_MAX))
                    else (if (X < X_MIN) (0))
                    else (exp(X))
                    }
```

This is especially useful for macros which contain multiple statements, such as

```
#define INITIALIZATION {
    X1 = 1.2;
    N_STEPS = 50;
    N_POINTS = 3 * N_STEPS;
    ....
}
```



The two styles for continuation lines can be mixed for different macros, but they cannot be mixed within the same macro definition.

## Null Substitution Text

A macro can be defined with null (empty) substitution text:

```
#define macro_name
```

The result is that all appearances of *macro\_name* are removed from the input file.

An interesting application is the design of a "compilation switch". You may wish to include or exclude part of the input file for testing purpose. A convenient way for you to do this is to mark (precede) the text you wish to include/exclude with a macro name, say, FLAG. Then you can "comment out" the marked text by having

```
#define FLAG  ''
```

or, you can "activate" the marked text by having

```
#define FLAG
```

which in effect removes FLAG from the input file.

## The \$ Argument

The "\$" argument is a special feature of OSA90 for macro functions. You can include "\$" as a formal argument of a macro function, and the \$ argument will be treated differently from any other arguments, in the sense that it can be concatenated with other arguments to form unique identifiers (names).

Example:

```
#define OFFSET($)    FACTOR$ * A$09 + J$
```

Then

```
OFFSET(3)
```

will be expanded into

```
FACTOR3 * A309 + J3
```

The \$ argument may appear anywhere in the formal argument list. The corresponding actual argument can be a number, a character string or even null (empty). For example,


```
#define CONCAT(A, $, B)    A$B
```

```
CONCAT(Drive_, Path, _File)
```

```
CONCAT(Spec, , ification)
```

The first reference will be expanded into Drive\_Path\_File and the second reference will be expanded into Specification.

## Macro Subcircuits

One of the most useful application of macro functions is the definition of symbolic subcircuits.  See Chapter 6 for details.

## 3.7 Include Files

**Syntax:**

```
#include file_name
```

**Examples:**

```
#include "equation.inc"
```

```
#include "/osa90/macros.inc"
```

```
#include "numbers.dat"
```

where "equation.inc", "/osa90/macros.inc" and "numbers.dat" are file names.

The preprocessor will incorporate the contents of the named file into the input file at the point where the `#include` directive appears.



The files to be included must already exist on the disk and must be accessible to OSA90 (i.e., the user must have the file read privilege).

Include files are useful for

- ▷ breaking up a very long file into several smaller files
- ▷ importing data from other programs and applications
- ▷ storing frequently used definitions so they can be shared among different input files

**Restrictions on include files:**

- ▷ The `#include` directive must appear within the body of a file block. The file block name (which begins the block) and the `END` keyword (which closes a file block) must be in the main input file and cannot be imported from an include file. In other words, a file block cannot begin or end within an include file.
- ▷ The input file may contain any number of `#include` directives, but an include file cannot contain any `#include` directives. In other words, you can have many files to be included into the input file, but you cannot have nested include files.
- ▷ An include file cannot contain definitions of optimization variables (see Chapter 4), because of back-annotation problems. After optimization, the input file text is updated with the new values of the optimizable variables. Since the text of an include file is utilized only during preprocessing, it cannot be updated after optimization.

## 3.8 Control Block

Many of the operation parameters of OSA90 are user-controllable by means of the Control block in the input file.

### Syntax

```
Control
    control_keyword = choice;

    control_keyword;
    ...
End
```

TABLE 3.2 CONTROL KEYWORDS

Keyword	Type	To Select
Accuracy	choice	solution accuracy required for optimization
Allow_Neg_Parameters	flag	permitting circuit model parameters to be negative
Alternative_Solver	flag	the alternative Newton nonlinear equation solver
Cooling_Ratio	value	a parameter for the simulated annealing optimizer
Disable_Adjoint	flag	no adjoint sensitivity analysis during optimization
Display_N_Digits	choice	number of decimal digits for numerical displays
Equality_Threshold	value	threshold value for comparing two numbers
Group_Delay_Freq_Step	value	frequency step size for group delay calculation
Huber_Threshold	value	threshold value for Huber optimization
Initial_Temperature	value	a parameter for the simulated annealing optimizer
Interpolate_Real_Imag	flag	interpolating imported data in rectangular form
Iterations_of_T	choice	a parameter for the simulated annealing optimizer
Jacobian_Perturbation	value	perturbation scale for nonlinear device Jacobians
NAN	value	a value representing "Not A Number"
Newton_EPS_Relative	value	solution accuracy required for Newton iteration
Non_Microwave_Units	flag	non-microwave physical unit system
No_Default_Bounds	flag	no default bounds for optimization variables
No_Q_Model	flag	no quadratic modeling for yield optimization
N_Iterations	choice	number of iterations for optimization
N_Yield_Outcomes	choice	number of random outcomes for yield optimization
Objective_Function	choice	choosing the objective function for optimization
One_Sided_Group_Delay	flag	one-sided perturbation for group delay calculation
Optimizer	choice	choosing the optimizer
Perturbation_Scale	value	perturbation scale for estimating gradients
Print_Best_Iterations	flag	printing only the best optimization iterations
Two_Sided_Jacobian	flag	two-sided perturbations for device Jacobians
Two_Sided_Perturbation	flag	two-sided perturbations for estimating gradients

## Control Keyword Types

In Table 3.2, the control keywords are categorized into three different types: choice, flag and value. For the "choice" type keywords, you select among a predefined set of multiple choices. The "flag" type keywords by their presence activate the options they represent. The "value" type keywords require to you specify a numerical value.

### **Accuracy:** *Solution Accuracy Required for Optimization*

When you invoke optimization, you can interactively select the accuracy required for the solution in the pop-up window (☞ Chapter 11).

You can also designate your choice in the Control block.

#### **Syntax:**

```
Accuracy = v;
```

where  $v$  is one of the choices: 0.01, 0.001, 1.0e-4, 1.0e-5, 1.0e-6.

Other values of  $v$  will be rounded to the closest choice which is smaller than  $v$ , e.g., 0.003 will be rounded to 0.001. If  $v < 1.0e-6$ , it will be set to 1.0e-6.

### **Allow\_Neg\_Parameters:** *Permitting Negative Circuit Model Parameters*

By default, OSA90 requires circuit model parameters (resistance, capacitance, inductance, dielectric constant, width, height, etc.) to be assigned positive values. If a negative circuit model parameter value is detected, the file parser will signal an error.

In some situations, such as in the development of a new empirical model, you may wish to allow negative values for the circuit model parameters.

#### **Syntax:**

```
Allow_Neg_Parameters;
```

This option affects only *circuit* parameters. Parameters and coefficients in generic equations and formulas can always be assigned positive or negative values as appropriate.



**Alternative\_Solver:** *Alternative Solver for Nonlinear Circuit Equations*

Solutions of nonlinear circuit equations are involved in harmonic and DC circuit analyses. Two different nonlinear solvers are implemented in OSA90.

The first one is an improved implementation of Powell's hybrid method (M.J.D. Powell, "A hybrid method for nonlinear equations", in *Numerical Methods for Nonlinear Algebraic Equations*, P. Rabinowitz Ed., New York: Gordon and Breach Science Publishers, 1970). This is chosen as the default nonlinear solver and it works well in most cases.

The alternative nonlinear solver is based on a norm-reducing Newton Method (H.R. Yeager and R.W. Dutton, "Improvement in norm-reducing Newton methods for circuit simulation", *IEEE Trans. Computer-Aided Design*, vol. 8, 1989). This method may provide better convergence in some situations. In the Control block, you can request that this alternative solver be used for nonlinear circuit simulation.

**Syntax:**

```
Alternative_Solver;
```

**Cooling\_Ratio:** *a Parameter for Simulated Annealing Optimization*

This is a parameter for the simulated annealing optimizer in OSA90. It controls the ratio for reducing the annealing temperature between iterations. As the temperature declines, exploratory moves are less likely to be accepted and the optimization is more likely to focus on the vicinity of the most promising local optimum.

You can specify a value for the cooling ratio either in the Control block or interactively in the optimization pop-up window (☞ Chapter 11).

**Syntax:**

```
Cooling_Ratio = v;
```

where  $v$  must be a constant and  $0.01 \leq v \leq 0.999999$ . The default value is 0.85.

**Disable\_Adjoint:** *Disabling Adjoint Sensitivity Analysis*

OSA90 incorporates circuit adjoint sensitivity analysis to compute gradients efficiently for optimization. This feature can be disabled in the Control block.

**Syntax:**

```
Disable_Adjoint;
```

This can be useful in studying the impact of gradient accuracy on optimization. Also, it may alter the sequence of simulation during optimization. For example, the adjoint analysis in OSA90 may require parameter perturbations at each frequency of circuit simulation, i.e., the parameter perturbation loop is within the frequency sweep loop (or power sweep, etc.). When the adjoint analysis is disabled, the parameter perturbation loop, if required, is outside of the frequency (power, etc.) sweep loop.

**Display\_N\_Digits:** *Number of Decimal Digits for Numerical Outputs*

OSA90 can display simulation results graphically or numerically (see Chapter 9). By default, numerical outputs are displayed with up to 4 significant decimal digits. You can change this default in the Control block.

**Syntax:**

```
Display_N_Digits = n;
```

where  $n$  is the number of decimal digits for numerical display and  $3 \leq n \leq 6$

For example, the numerical value 12.3456 is displayed as 12.35 by default. If you specify

```
Display_N_Digits = 5;
```

then the same value will be displayed as 12.346.

**Equality\_Threshold:** *Tolerance for Comparing Two Floating-Point Numbers*

OSA90 supports conditional expressions using If-Else structures (Chapter 4). One of the often used conditions is to compare two numbers to see if they are equal. For integers, such equality check can be exact. But, for floating-point numbers, such comparison is subject to truncation errors.

The control keyword `Equality_Threshold` allows you to prescribe a tolerance within which two floating-point numbers will be considered equal. More precisely, if

$$|X - Y| \leq \text{Equality\_Threshold} \cdot (|X| + |Y|)$$

then `X` and `Y` are considered equal.

OSA90 applies criterion to conditional expressions such as (Chapter 4)

```
if (X = Y) ...
```

```
if (X >= Y) ...
```

```
if (X <= Y) ...
```

**Syntax**

```
Equality_Threshold = v;
```

where  $0 \leq v \leq 1$ . The default value for `v` is 1.0E-5.

**Group\_Delay\_Freq\_Step:** *Step Size for Group Delay Calculation*

OSA90 calculates group delay by

$$[\phi(f + \Delta f) - \phi(f)] / \Delta f$$

where  $\phi$  is the phase of the appropriate `S` parameter and `f` represents the frequency. You can specify the frequency step size  $\Delta f$  in the Control block.

**Syntax**

```
Group_Delay_Freq_Step = v
```

where `v` must be a positive constant. The default value is 0.01.

A related control option is `One_Sided_Group_Delay`.

**Huber\_Threshold:** *a Parameter for Huber Optimization*

When you invoke optimization and select the Huber optimizer, you can interactively select the threshold value in the pop-up window (Chapter 11).

You can also designate the threshold value in the Control block.

**Syntax**

```
Huber_Threshold = v;
```

where  $v$  must be a positive value.

**Initial\_Temperature:** *a Parameter for Simulated Annealing Optimization*

This is a parameter for the simulated annealing optimizer in OSA90. It specifies the initial value of the annealing temperature.

The initial temperature has a significant impact on the outcome of the simulated annealing optimization. It influences the step size of exploratory moves. An initial temperature that is too small may retard the process by not allowing enough exploratory moves over a sufficiently large area of the parameter space.

Unfortunately, in general the "optimal" initial temperature is problem dependent and cannot be easily determined in advance.

You can specify an initial temperature either in the Control block or interactively in the optimization pop-up window (Chapter 11).

**Syntax**

```
Initial_Temperature = v;
```

where  $v$  must be a constant and  $v \geq 0.01$ . The default value is 5.0.

**Interpolate\_Real\_Imag:** *Option for Interpolating Imported Data*

When imported data is used to define linear subcircuits, OSA90 may perform interpolation if the data does not contain all the frequencies involved in simulation.

By default, the interpolation is done directly on the supplied data. For instance, if the data supplied contains  $S$  parameters in the polar form (MS11, PS11, ...), then interpolation, if necessary, is performed on MS11, PS11, etc.

You can instruct the program to convert the imported data to rectangular form before interpolation.

**Syntax:**

```
Interpolate_Real_Imag;
```

For well-behaved data, this should not make a significant difference. However, if the phase varies abruptly between frequencies, interpolation in rectangular or polar form may lead to different results. There are no general rules as to which is the better choice. The best solution is perhaps to supply data with more densely spaced frequencies, if this is feasible.

**Iterations\_of\_T:** *a Parameter for Simulated Annealing Optimization*

This is a parameter for the simulated annealing optimizer in OSA90. It controls the number of iterations before temperature reduction.

You can specify a value for this parameter either in the Control block or interactively in the optimization pop-up window (☞ Chapter 11).

**Syntax:**

```
Iterations_of_T = n;
```

where  $n$  is one of the choices: 5, 10, 20, 40, 70, 100.

Other values of  $n$  will be rounded to the closest choice which is greater than  $n$ , e.g., 30 will be rounded to 40. If  $n > 100$ , it will be set to 100.

**Jacobian\_Perturbation:** *Step Size for Estimating Nonlinear Device Jacobians*

For circuits containing nonlinear devices, the Jacobians of the nonlinear currents and charges with respect to the device intrinsic voltages are needed for the Newton iteration as well as for linearizing the devices for small-signal analysis.

For built-in device models, the Jacobians are evaluated analytically. But for user-defined models, the Jacobians have to be computed numerically by perturbations. For instance, the derivative of a nonlinear current  $i$  with respect to a voltage  $v$  can be estimated by

$$[i(v + \Delta v) - i(v)] / \Delta v$$

You can specify the relative step size for  $\Delta v$  in the Control block.

**Syntax**

**Jacobian\_Perturbation** =  $v$

where  $v$  must be a constant and  $1.0e-6 \leq v \leq 0.1$ . The default value is 1.e-4.

A related control option is **Two\_Sided\_Jacobian**.

**NAN:** *Representing Not A Number*

Sometimes you may define expressions that are meaningful only when the parameters fall within a given range. The predefined label NAN allows you to signal OSA90 that the result of an expression is out of range.

Example:

```
Power_dBm = if (Power > 1.0e-6) (10 * log10(Power) + 30) else (NAN);
```

OSA90 will not display values that are marked by NAN. Any arithmetic operations which involve an NAN operand will result in an NAN value.

In the Control block, you can designate a value to be used to represent NAN.

**Syntax**

**NAN** =  $x$ ;

where  $x$  is a numerical value. The default value for NAN is 98989.0.

You should choose a value that is least likely to coincide with the "normal" numerical values in the same input file.

**Newton\_EPS\_Relative:** *Solution Accuracy Required for Newton Iteration*

Newton iteration is employed for nonlinear circuit simulation. You can specify the accuracy required for the Newton iteration in the Control block.

**Syntax**

```
Newton_EPS_Relative = v;
```

where  $v$  must be a constant and it represents a multiplier to the internally determined accuracy threshold. For example,  $v = 0.1$  means the solution should be 10 times more accurate than the default. The valid range is  $1.0e-6 \leq v \leq 10$ . The default is 1.

A tighter accuracy threshold may require longer simulation time. Excessively tight accuracy requirement may even lead to non-convergence.

**Non\_Microwave\_Units**

This keyword controls the default physical unit system to be used, see Section 3.9.

**No\_Default\_Bounds:** *Disabling Implicit Bounds on Optimization Variables*

In OSA90, you can assign explicit bounds on optimization variables (Chapter 4). On the other hand, if an optimization variable is defined without explicit bounds, then it is implicitly constrained within a set of default bounds.

The default bounds depend on the starting value (initial value) of the optimization variable. If the starting value is positive or zero, the default bounds are  $(0, \infty)$ , otherwise the default bounds are  $(-\infty, 0)$ .

You can disable the assignment of default bounds in the Control block. This will allow variables defined without explicit bounds to vary freely from  $-\infty$  to  $\infty$ .

**Syntax**

```
No_Default_Bounds;
```

**No\_Q\_Model:** *Disabling Quadratic Modeling for Yield Optimization*

When you invoke yield optimization, by default a quadratic modeling feature is employed to reduce computational cost.

You can disable the use of Q-model in the Control block.

**Syntax:**

```
No_Q_Model;
```

**N\_Iterations:** *Maximum Number of Optimization Iterations*

When you invoke optimization, you can interactively select the maximum number of optimization iterations in the pop-up window (☞ Chapter 11).

You can also designate your choice in the Control block.

**Syntax:**

```
N_Iterations = n;
```

where  $n$  is one of the choices: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 30, 50, 100, 999, 9999.

Other values of  $n$  will be rounded to the closest choice which is greater than  $n$ , e.g., 45 will be rounded to 50. If  $n > 9999$ , it will be set to 9999.

**N\_Yield\_Outcomes:** *Number of Statistical Outcomes for Yield Optimization*

When you invoke yield optimization, you can interactively select the number of statistical outcomes in the pop-up window (☞ Chapter 13).

You can also designate your choice in the Control block.

**Syntax:**

```
N_Yield_Outcomes = n;
```

where  $n$  is one of the choices: 10, 20, 50, 100, 200, 500.

Other values of  $n$  will be rounded to the closest choice which is greater than  $n$ , e.g., 80 will be rounded to 100. If  $n > 500$ , it will be set to 500.



**Objective\_Function:** *Choosing the Objective Function for Optimization*

Some of the optimizers offered by OSA90 can optimize more than one objective function. For example, the Huber optimizer can optimize the one- or two-sided Huber objective function. You can indicate the preferred objective function in the Control block.

**Syntax:**

```
Objective_Function = name;
```

```
name can be: L1,      L2,      Minimax,   Generalized_L2,   Sum_of_Errors,
              Yield,   GL1,     Huber,      One_Sided_Huber,  One_Sided_L1
```

Not all the objective functions are appropriate for every optimizer. In the event that there is a conflict between the objective function and the optimizer chosen, the objective function specified in the Control block will be ignored.

**One\_Sided\_Group\_Delay:** *Calculating Group Delay by One-Sided Perturbations*

By default, two-sided perturbations are used for group delay computation. In the Control block, you can specify one-sided perturbations to be used instead.

**Syntax:**

```
One_Sided_Group_Delay;
```

**Optimizer:** *Designating the Default Optimizer*

OSA90 offers a comprehensive set of optimizers (☞ Chapter 11). You can designate the default optimizer in the Control block.

**Syntax:**

```
Optimizer = optimizer_name;
```

```
optimizer_name can be: L1, L2, Minimax, Conjugate_Gradient,
                       Huber, One_Sided_L1, Quasi_Newton,
                       Random, Simplex, Simulated_Annealing,
                       Yield
```

**Perturbation\_Scale:** *Perturbation Scale for Estimating Gradients*

OSA90 employs a number of advanced techniques to calculate in the most efficient way the derivatives required for gradient-based optimization. However, in certain situations some of the derivatives may be estimated by perturbations (finite differences).

For example, the derivative of a function  $f(x)$  with respect to  $x$  can be estimated by

$$[f(x + \Delta x) - f(x)] / \Delta x$$

You can specify the relative step size for  $\Delta x$  in the Control block.

**Syntax:**

**Perturbation\_Scale** =  $\nu$

where  $\nu$  must be a constant and  $1.0e-6 \leq \nu \leq 0.5$ . The default value is  $5.0e-6$ .

A related control option is **Two\_Sided\_Perturbation**.

**Print\_Best\_Iterations:** *an Option for Printing Optimization Results*

When you invoke optimization, you can interactively select in the pop-up window the option to print all the iterations or the best iterations only (see Chapter 11).

The default is to print all the iterations. You can instruct the program to print only the best iterations in the Control block.

**Syntax:**

**Print\_Best\_Iterations;**

**Two\_Sided\_Jacobian:** *Two-Sided Perturbation for Nonlinear Device Jacobians*

For circuits containing nonlinear devices, the Jacobians of the nonlinear currents and charges with respect to the device intrinsic voltages are needed for the Newton iteration as well as for linearizing the devices for small-signal analysis.

For most built-in device models, the Jacobians can be evaluated analytically. But for user-defined models, the Jacobians may have to be computed numerically by perturbations (finite differences).

By default, one-sided perturbations (forward difference) are used. For instance, the derivative of a nonlinear current  $i$  with respect to a voltage  $v$  can be estimated by

$$[i(v + \Delta v) - i(v)] / \Delta v$$

In the Control block, you can request two-sided perturbations (full differences):

**Syntax**

`Two_Sided_Jacobian;`

Using two-sided perturbations, the derivative of  $i$  with respect to  $v$  is estimated by

$$[i(v + \Delta v) - i(v - \Delta v)] / (2 \Delta v)$$

This usually improves the accuracy at the expense of increased computational effort.

A related control option is `Jacobian_Perturbation`.

### **Two\_Sided\_Perturbation:** *Two-Sided Perturbation for Estimating Gradients*

OSA90 employs a number of advanced techniques to calculate in the most efficient way the derivatives required for gradient-based optimization. However, in certain situations some of the derivatives may be estimated by perturbations (finite differences).

By default, one-sided perturbations (forward difference) are used, i.e., the derivative of a function  $f(x)$  w.r.t. a variable  $x$  is estimated by

$$[f(x + \Delta x) - f(x)] / \Delta x$$

In the Control block, you can request two-sided perturbations (full differences):

**Syntax**

`Two_Sided_Perturbation;`

Using two-sided perturbations, the derivative of  $f(x)$  w.r.t.  $x$  is estimated by

$$[f(x + \Delta x) - f(x - \Delta x)] / (2 \Delta x)$$

This usually improves the accuracy at the expense of increased computational effort.

## 3.9 Physical Units

When entering numerical constants in an input file, you can include physical units from those listed in Table 3.3. For example, you can use 3.5CM to represent 3.5 centimeters.

TABLE 3.3 PHYSICAL UNITS

Quantity	Available Keywords			
capacitance	F	(farad)	MF	(millifarad)
	UF	(microfarad)	NF	(nanofarad)
	PF	(picofarad)		
conductance	/OH	(siemens)	/KOH	(millisiemens)
	/MOH	(microsiemens)		
frequency	HZ	(hertz)	KHZ	(kilohertz)
	MHZ	(megahertz)	GHZ	(gigahertz)
	THZ	(terahertz)	PHZ	(petahertz)
inductance	H	(henry)	MH	(millihenry)
	UH	(microhenry)	NH	(nanohenry)
	PH	(picohenry)		
resistance	OH	(ohm)	KOH	(kilo-ohm)
	MOH	(mega-ohm)		
time	SEC	(second)	MS	(millisecond)
	US	(microsecond)	NS	(nanosecond)
	PS	(picosecond)		
power	DBM	(dBm)	W	(watt)
	MW	(milliwatt)		
current	A	(ampere)	MA	(milliampere)
	UA	(microampere)	NA	(nanoampere)
voltage	V	(volt)	MV	(millivolt)
	UV	(microvolt)		
phase	DEG	(degree)	RAD	(radian)
length	M	(meter)	CM	(centimeter)
	MM	(millimeter)	UM	(micron)
	IN	(inch)	MIL	(milli-inch)
	UIN	(microinch)		

Like all keywords, the physical units are case insensitive, e.g., MHZ, MHz and mhz all represent megahertz.



The unit must not be separated by space from the numerals. For instance, 3.5cm cannot be entered as 3.5 cm.

## Default Units

Numerical constants without explicit units in the input file are assigned default units. Two systems of default units are available, as listed in Table 3.4 under the headings Microwave and Non-Microwave.

**TABLE 3.4 DEFAULT UNITS**

Quantity	Microwave	Non-Microwave
capacitance	NF	F
conductance	/OH	/OH
frequency	GHZ	HZ
inductance	NH	H
resistance	OH	OH
time	NS	SEC
power	DBM	DBM
current	A	A
voltage	V	V
phase	DEG	DEG
length	M	M

The difference between these two systems is their default units for capacitance, frequency, inductance and time.

The Microwave unit system is the default system. You can change the default unit system to Non-Microwave in the Control block.

```
Syntax
Control
    Non_Microwave_Units;
    ...
End
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000

## 4

**Expressions**

<b>4.1 Overview</b> .....	4-1
<b>4.2 Label Names</b> .....	4-2
<b>4.3 Constant Labels</b> .....	4-3
<b>4.4 Predefined Labels</b> .....	4-4
<b>4.5 Optimization Variables</b> .....	4-6
<b>4.6 Expressions</b> .....	4-8
<b>4.7 Conditional Expressions</b> .....	4-10
<b>4.8 Arrays</b> .....	4-13
<b>4.9 Array Expressions</b> .....	4-17
<b>4.10 Array Functions</b> .....	4-19
<b>4.11 String Labels</b> .....	4-41
<b>4.12 Transformations</b> .....	4-43
<b>4.13 Cubic Spline Functions</b> .....	4-49
<b>4.14 Piece-Wise Linear Interpolation</b> .....	4-52





## 4

# Expressions

## 4.1 Overview

In the Expression block of the input file, you can define

- ▷ labels representing constants and optimization variables
- ▷ labels using expressions to describe formulas, equations and functions
- ▷ Datapipe connections to external programs
- ▷ statistical variables

**Syntax:****Expression**

```
label_name: constant;  
label_name: optimization variable;  
label_name: expression;
```

```
Datapipe: protocol descriptions;
```

```
...
```

```
End
```

This chapter focuses on labels and expressions. Datapipe is described in Chapter 5 and statistical variables are described in Chapter 12.



Labels and expressions can be defined in the Model block also. The functionality of the Model block is a superset of the functionality of the Expression block. All the descriptions in this chapter are applicable to the Model block. Technically, you do not need to have a separate Expression block. For clarity, though, it may be desirable to keep only the circuit descriptions in the Model block, and keep abstract variables and functions in the Expression block.

## 4.2 Label Names

Labels are symbolic names created by the user to represent constants, variables, formulas and functions.

A label can be defined by a statement which begins with the label name, followed by a colon ":" or an equal sign "=", and ends with a semicolon ";".

For example,

```
V1: 2.5;
Formula1 = V1 * V1 - 0.76 * V1 + 2;
```

where V1 and Formula1 are label names. The label V1 represents a constant, and the label Formula1 represents a formula (an expression).

**Reserved symbols.** Label names can be arbitrary character strings but they must not contain any of the reserved symbols listed in Table 4.1, and they must not begin with a digit (0, ..., 9). For instance, "3\_VAR" or "ABC!" cannot be used as a label name.

TABLE 4.1 RESERVED SYMBOLS

+	-	.	,	*	/	\	!	@	#	\$
%	^	&	(	)	=	{	}	[	]	:
;	'	<	>	?		~	'			

**Label names are case insensitive by default**, e.g., FORMULA1, formula1 and Formula1 are considered to be referring to the same label.

**Literal label names** can be created by enclosing the label names in quotation marks. Literal label names are case sensitive, e.g., "formula1" and "Formula1" will be treated as distinct and different label names. Also, literal label names are not restricted by reserved symbols, e.g.,

```
"/osa90/my_data.dat"
"1989-device-33-55-7"
```

are legitimate label names. This is especially relevant for file names, because they are case sensitive on UNIX systems and often contain the reserved symbols "." and "/".

**Length of label names** can be up to 4096 characters.

## 4.3 Constant Labels

In the simplest form, a label represents a numerical constant, such as

```
Boltzmann_Constant: 1.3806E-23;
```

Constant labels and preprocessor macro constants (Chapter 3) are nearly identical in their usage. For example, a macro constant defined as

```
#define Boltzmann_Constant 1.3806E-23
```

can be used almost identically to the constant label defined above.

The subtle difference lies in their internal structure: a macro is meaningful only to the file preprocessor as an alias of its substitution text, whereas a label is structured as a permanent identifier. A label can be used as an output for graphical and numerical display, but a macro cannot. On the other hand, a label requires slightly more memory and processing time than a macro.

## 4.4 Predefined Labels

Predefined labels are internally created by OSA90 rather than by the user, otherwise they are no different from other labels. The purpose of predefined labels is to facilitate access to certain internal variables by the user.

**TABLE 4.2 PREDEFINED LABELS**

Name	Description
FREQ	the frequency in circuit simulation
Iteration_Count	index of optimization iterations
NAN	representing "Not A Number"
Perturbed_Analysis	index of perturbation for gradients
PI	the constant 3.14159

### The FREQ Label

In the Sweep, Specification and MonteCarlo blocks, you can prescribe a set of frequencies for circuit simulation and optimization (see Chapters 9, 11 and 12). During simulation and optimization, OSA90 will automatically update FREQ with the then current value. Using FREQ, you can create frequency-dependent formulas and functions, e.g.,

$$\text{Reactance}_1 = \text{FREQ} * \text{Inductance}_1;$$


The unit of FREQ is GHZ (gigahertz) in the default microwave unit system, but you can change the default to the non-microwave unit system in which the unit for FREQ is HZ (see Chapter 3).

### Iteration Count During Optimization

The predefined label Iteration\_Count facilitates user access to the internal variable of optimization iteration index. At the start of optimization, the label is initialized to the value of 1. It is then incremented at each and every iteration during optimization.

This label can be useful when OSA90 is used to drive an external simulator and the iteration count is of some interest to the external simulator. For example, the external simulator may wish to store some intermediate output data in a file, indexed by the iteration count.

## The NAN Label

Sometimes you may define expressions that are meaningful only when the parameters fall within a given range. The predefined label NAN allows you to signal OSA90 that the result of an expression is out of range.

Example:

```
Power_dBm = if (Power > 1.0e-6) (10 * log10(Power) + 30) else (NAN);
```

OSA90 will not display values that are marked by NAN. Any arithmetic operations which involve an NAN operand will result in an NAN value.

You can choose a value for the NAN label in the Control block (Chapter 3). By default, the value of NAN is set to 98989.0.

## Perturbed Analysis Index

If you use OSA90 to drive an external simulator through Datapipe (Chapter 5) and invoke optimization, then perturbed analyses may be required to estimate the gradients.

In a perturbed analysis, one of the optimization variables is perturbed from its nominal value. The external simulator is called only if the variable being perturbed affects the Datapipe inputs (i.e., inputs to the external simulator).

The predefined label Perturbed\_Analysis can be used to distinguish between a nominal analysis and a perturbed analysis. You can pass Perturbed\_Analysis through Datapipe to an external simulator which can take advantage of such a distinction.

The value of Perturbed\_Analysis is set to 0 for a nominal analysis and set to  $i$  when the  $i$ th variable is perturbed. If the total number of optimization variables is  $n$ , then the value of Perturbed\_Analysis is between 0 and  $n$ .

## Circuit Response Labels

In addition to the predefined labels listed in Table 4.2, OSA90 may also create labels to represent circuit responses. Such labels will be created according to the circuit definition in the Model block (Chapter 6).

## 4.5 Optimization Variables

Optimization variables in the file are marked by question marks.

### Syntax

```
label_name: ?initial_value?;
```

Example:

```
X1: ?2.5?;
```

After optimization, the variables will be automatically updated with their optimized values. For instance, consider the variable labelled X1 in the above example. Suppose that its value is changed to 1.234 after optimization. Then the corresponding text in the input file will be automatically updated to be

```
X1: ?1.234?;
```



Optimization variables can also be directly defined for circuit model parameters in the Model block without being associated with a label (see Chapter 6).

### Bounds on Variables

Optional bounds can be assigned to an optimization variable:

### Syntax

```
label_name: ?lower_bound initial_value upper_bound?;
```

Example:

```
X1: ?0.1 2.5 10?;
```

This will force the value of the variable to remain within *lower\_bound* and *upper\_bound* during optimization. If you specify bounds, you must specify both bounds.

**Relative bounds** can also be specified.

### Syntax

```
label_name: ?lower% initial_value upper%?;
```

The effective bounds are calculated by the file parser as

$$\text{lower\_bound} = \text{initial\_value} - \text{ABS}(\text{lower} * \text{initial\_value}) / 100$$

$$\text{upper\_bound} = \text{initial\_value} + \text{ABS}(\text{upper} * \text{initial\_value}) / 100$$

Example:

```
X1: ?80% 10 70%?;
```

The effective lower bound will be 2 and the effective upper bound 17.



Note that the effective bounds are evaluated based on the *initial\_value* when the input file is parsed and they will not change during optimization.

## Default Bounds

An optimization variable defined without explicit bounds will be assigned the default bounds: (0 +∞) if the initial value is positive or zero, or (-∞ 0) otherwise.

For example,

```
X1: ??2.5?;
```

X1 will be given the default bounds (0 +∞), i.e., its value will remain positive.

The default bounds are based on the assumption of natural bounds which exist in many physical problems. For instance, when a physical measurement (length, weight, etc.) is optimized, its value should never become negative.

**The assignment of default bounds can be disabled** in the Control block:

**Syntax:**

```
Control
  No_Default_Bounds;
  ...
End
```

The `No_Default_Bounds` control keyword indicates that optimization variables defined without explicit bounds should be allowed to vary freely from -∞ to ∞. (The Control block is described in Chapter 3).

## 4.6 Expressions

Expressions generically refer to formulas, equations and functions defined through combinations of constants, algebraic operators, mathematical functions, and references to labels that are already defined.

For example,

```
Euclidean_Norm = SQRT(X * X + Y * Y);
```

where `Euclidean_Norm` is the label representing the expression, `SQRT` is the mathematical function of square root, "\*" and "+" are algebraic operators, and `X` and `Y` refer to labels that should be already defined.

**Optimization variables** cannot be created within an expression. For example,

```
F1 = FREQ * ?2.5?;
```

is not acceptable. The optimization variable must be defined separately, such as

```
X1: ?2.5?;
```

```
F1 = FREQ * X1;
```

### Algebraic Operators

Table 4.3 lists the set of algebraic operators available.

**TABLE 4.3 ALGEBRAIC OPERATORS**

Symbol	Operation
&	logical AND
	logical OR
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
=	equal to
+	addition
-	subtraction
*	multiplication
/	division
^	exponentiation
()	precedence



The operators are listed in ascending order of precedence in Table 4.3. Parentheses can be used to override the natural order of precedence, e.g.,

$$\text{Error} = \text{Weight} * (\text{Response} - \text{Goal});$$

## Mathematical Function Library

TABLE 4.4 LIBRARY OF MATHEMATICAL FUNCTIONS

Function	Description
ABS( <i>x</i> )	absolute value of <i>x</i>
ACOS( <i>x</i> )	arccosine of <i>x</i>
ASIN( <i>x</i> )	arcsine of <i>x</i>
ATAN( <i>x</i> )	arctangent of <i>x</i>
ATAN2( <i>x,y</i> )	arctangent of <i>x/y</i>
CEIL( <i>x</i> )	the smallest integer $\geq x$
COS( <i>x</i> )	cosine of <i>x</i>
COSH( <i>x</i> )	hyperbolic cosine of <i>x</i>
EVEN( <i>x</i> )	returns 1 if <i>x</i> is an even number, returns 0 otherwise
EXP( <i>x</i> )	exponential function of <i>x</i>
FLOOR( <i>x</i> )	the largest integer $\leq x$
INTEGER( <i>x</i> )	returns 1 if <i>x</i> is an integer, returns 0 otherwise
J0( <i>x</i> )	Bessel function of the first kind, order 0
J1( <i>x</i> )	Bessel function of the first kind, order 1
JN( <i>n,x</i> )	Bessel function of the first kind, order <i>n</i>
LOG( <i>x</i> )	natural logarithm of <i>x</i>
LOG10( <i>x</i> )	base-10 logarithm of <i>x</i>
MAX( <i>x1,...,xn</i> )	the maximum of <i>x1</i> , ..., <i>xn</i>
MIN( <i>x1,...,xn</i> )	the minimum of <i>x1</i> , ..., <i>xn</i>
NEG( <i>x</i> )	returns 1 if <i>x</i> < 0, returns 0 otherwise
NINT( <i>x</i> )	the nearest integer to <i>x</i>
ODD( <i>x</i> )	returns 1 if <i>x</i> is an odd number, returns 0 otherwise
POS( <i>x</i> )	returns 1 if <i>x</i> > 0, returns 0 otherwise
SIN( <i>x</i> )	sine of <i>x</i>
SINH( <i>x</i> )	hyperbolic sine of <i>x</i>
SQRT( <i>x</i> )	square root of <i>x</i>
TAN( <i>x</i> )	tangent of <i>x</i>
TANH( <i>x</i> )	hyperbolic tangent of <i>x</i>
Y0( <i>x</i> )	Bessel function of the second kind, order 0
Y1( <i>x</i> )	Bessel function of the second kind, order 1
YN( <i>n,x</i> )	Bessel function of the second kind, order <i>n</i>
ZERO( <i>x</i> )	returns 1 if <i>x</i> = 0, returns 0 otherwise

## 4.7 Conditional Expressions

Conditional expressions can be used to assign different values to a label according to a given set of conditions. OSA90 supports both explicit `if - else` structures and implicit conditional expressions.

### If - Else Structure

#### Syntax

```
label: if (condition) (expression1) [else (expression2)];
```

Examples:

```
Absolute_Difference: if (X > Y) (X - Y) else (Y - X);
```

```
X_Truncated: if (X > 0) (X);
```



*condition*, *expression1* and *expression2* must all be enclosed within parentheses.

If *condition* is true, then *label* is evaluated from *expression1*, otherwise it is evaluated from *expression2*.

If *expression2* is omitted and *condition* is false, then *label* is set to zero.

### Logical Condition

The *condition* in an `if - else` structure can be a logical expression constructed using the operators `&`, `|`, `>`, `<`, `>=`, `<=` and `=`.

In OSA90, a logical expression evaluates to 1 if it is true, or 0 if false.

Example:

```
Y: if (X > -2 & X <= 2) (X) else (-2);
```

Y is set to X if  $-2 < X \leq 2$ , otherwise Y is set to -2.

Another example:

```
Y: if (X < -2 | X > 2) (X);
```

Y is set to X if  $X < -2$  or  $X > 2$ , otherwise Y is set to 0.

## Threshold Value for Comparing Two Floating-Point Numbers

Conditional expressions such as

```
if (X = Y) ...
```

```
if (X >= Y) ...
```

```
if (X <= Y) ...
```

involves equality check between two numbers. If the numbers are floating-point values, the equality check is subject to truncation errors.

In the Control block (Chapter 3), you can define `Equality_Threshold`. If

$$|X - Y| \leq \text{Equality\_Threshold} \cdot (|X| + |Y|)$$

then X and Y are considered equal.

### Syntax:

```
Equality_Threshold = v;
```

where  $0 \leq v \leq 1$ . The default value for  $v$  is 1.0E-5.

## Nested If - Else

Nested if - else structures are supported:

```
Y: if (X > X_MAX) (exp(X_MAX))
    else (
        if (X < X_MIN) (exp(X_MIN)) else (exp(X))
    );
```

The inner if - else structure(s) must be entirely enclosed within parentheses.

Only the value of a label but not the existence of the label itself can be made conditional. Hence the following "statement" is not permissible:

```
if (X > X_MAX) (
    Y: exp(X_MAX);
)
```

## Algebraic Condition

The *condition* in an if-else structure can also be an algebraic expression. In this case, it is considered "true" if it evaluates to a nonzero value, or "false" if it evaluates to zero.

Example:

```
Y: if (X - 3.5) (4.44) else (7.77);
```

Y will be set to 4.44 if X equals to 3.5, otherwise Y is set to 7.77.

## Implicit Conditional Expression

In addition to explicit if-else structures, implicit conditional expressions can be defined using the &, |, >, <, >=, <= and = operators and the POS(), NEG() and ZERO() functions.

Example:

```
Y = POS(X) * Y1 + ZERO(X) * Y2 + NEG(X) * Y3;
```

is equivalent to

```
Y = if (X > 0) (Y1) else (if (X = 0) (Y2) else (Y3));
```

The same conditions can also be expressed as

```
Y = (X > 0) * Y1 + (X = 0) * Y2 + (X < 0) * Y3;
```

Implicit conditional expressions are generally less efficient. In the example, if an explicit if-else structure is used, only one of the three formulas, namely Y1, Y2 or Y3, needs to be evaluated under any given condition. Using implicit expressions all three formulas are evaluated first and then multiplied by the corresponding conditions (1 or 0). This is especially vulnerable if one of the formulas may result in a floating-point error under certain conditions.

For example, expression

```
Y = POS(X) * log(X);
```

will cause an error when X is zero or negative. The correct expression should be

```
Y = if (X > 0) (log(X));
```

On the other hand, implicit conditional expressions can be useful for complex expressions which depend on multi-layer and inter-related algebraic conditions.

## 4.8 Arrays

Labels can be defined to represent arrays, including one-dimensional arrays (vectors) and two-dimensional arrays (matrices).

### Vectors

#### Syntax:

```
label_name[n]: [x1 x2 ... xn];
```

where  $n$  is the number of elements in the vector,  $1 \leq n \leq 4096$ .

The number of the initializers within the square brackets, namely  $x_1, x_2, \dots, x_n$ , must be equal to the length of the vector. The initializers can be constants, optimization variables, other labels that are already defined and expressions.

#### Example:

```
Vector1[5] = [1.2 ?5? F1 (F2 * sin(2*PI*FREQ*T)) -12.34];
```

where  $F_1, F_2$  and  $T$  refer to labels that must be already defined, and  $PI$  and  $FREQ$  refer to predefined labels.



When an array element is initialized by an expression, the entire expression must be enclosed within a pair of parentheses to clearly delimit it from the other entries.

Optionally, the initializers can be separated by commas for clarity, e.g.,

```
Vector1[5] = [1.2, ?5?, F1, (F2 * sin(2*PI*FREQ*T)), -12.34];
```

The array definition statement can occupy as many lines as necessary. For instance, the initializers of a long vector may take up several lines:

```
Long_Vector[200] = [1.0 2.0 3.0 ...
...
... 199.0 200.0];
```

Individual elements of a vector are denoted by  $label\_name[k]$ , where  $1 \leq k \leq n$ . For instance,  $Vector1[3]$  refers to the 3rd element of the vector  $Vector1$ .

## Specify the First Index of a Vector

You can specify the first index of a vector (i.e., the index of the first element) to be different than 1.

### Syntax

```
label_name[n1:n2]: [xn1 ... xn2];
```

where  $n1$  and  $n2$  are the indices of the first and the last elements, respectively,  $-65535 \leq n1 \leq n2 \leq 65535$ , the number of elements =  $(n2 - n1 + 1) \leq 4096$ .

Example:

```
Vector2[-2:2] = [1.1 2.2 3.3 4.4 5.5];
```

The indices of the vector elements range from  $n1$  to  $n2$ . For instance, `Vector2[0]` refers to the 3rd element of the vector `Vector2`.

## Matrices

### Syntax

```
label_name[n,m]: [x11 x12 ... x1m
                  x21 x22 ... x2m
                  ...
                  xn1 xn2 ... xnm];
```

where  $n$  is the number of rows and  $m$  is the number of columns,  $1 \leq n \leq 4096$ ,  $1 \leq m \leq 4096$ ,  $(n \times m) \leq 4096$ .

Example:

```
Matrix1[3,4] =
[ 1.23  2.34  3.45  4.56
  5.67  7.89  9.01 -1.23
 -2.34 -3.45 -4.56 -5.67 ];
```

**Order of initialization is row-wise**, i.e., the first row of the matrix is completely filled first, then the second row, and so on.

The initializers may occupy as many lines as necessary and each line does not have to correspond to one row of the matrix. For example:

```
Matrix1[3,4] =
[ 1.23  2.34  3.45  4.56  5.67  7.89
  9.01 -1.23 -2.34 -3.45 -4.56 -5.67 ];
```

The initializers can be constants, optimization variables, other labels that are already defined and expressions.

## Specify the First Indices of a Matrix

You can specify the indices of the first row and/or the first column of a matrix to be different than 1.

### Syntax

```
label_name[n1:n2,m1:m2]: [xn1m1 ... xn1m2
                          ...
                          xn2m1 ... xn2m2];
```

where  $n1$ ,  $n2$ ,  $m1$  and  $m2$  are the indices of the first row, the last row, the first column and the last column, respectively,  
 $-65535 \leq n1 \leq n2 \leq 65535$ ,  $-65535 \leq m1 \leq m2 \leq 65535$ ,  
 the size of the matrix =  $(n2 - n1 + 1) \times (m2 - m1 + 1) \leq 4096$ .

## Initialization by Sub-Vectors and Sub-Matrices

Arrays can be initialized not only by scalars but also by other arrays. For example,

```
Vector1[3] = [ ... ];
Vector2[7] = [ ... ];

Vector3[10] = [ Vector1 Vector2 ];
```

which in effect concatenates *Vector1* and *Vector2* to produce *Vector3*. *Vector1* and *Vector2* may be considered as sub-vectors of *Vector3*.

The sizes of the arrays must be consistent, i.e., the combined size of all initializers (scalars and sub-vectors) must not exceed (must be equal to) the size of vector being initialized.

Matrices can be initialized block-wise, i.e., by sub-matrices. Initializers of a matrix may include vectors which are oriented row-wise. For example,

```
row1[5] = [ ... ];
row2[5] = [ ... ];
row3[3] = [ ... ];

Matrix1[3,5] = [ row1 row2 1.0 row3 2.0 ];
```

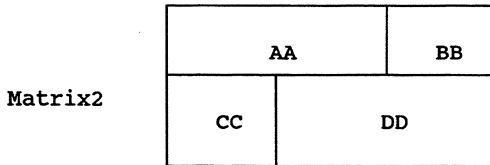
where *row1* and *row2* initialize the first and second rows of *Matrix1*, respectively, and *row3* initializes part of the third row.

Initializers of a matrix can also be other matrices, e.g.,

```
AA[2,4] = [ ... ];
BB[2,2] = [ ... ];
CC[3,2] = [ ... ];
DD[3,4] = [ ... ];

Matrix2[5,6] = [ AA  BB
                  CC  DD ];
```

The result can be illustrated as follows.



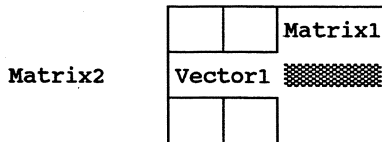
The matrix being initialized is filled row-wise. Each initializer starts at the next available location in the matrix (i.e., the next element not yet initialized). If the initializer is a vector or a matrix, it fills a sub-matrix of the same size and dimensions as the initializer itself. The sub-matrix must not extend beyond the dimensions of the overall matrix. Also, the sub-matrix must not overlap with any other initializers, i.e., the elements within the sub-matrix must not have been initialized already.

For example, the following statements

```
Matrix1[3,2] = [ ... ];
Vector1[4] = [ ... ];

Matrix2[3,4] = [ 1.0  2.0  Matrix1
                  Vector1
                  3.0  4.0          ];
```

would generate an error, because Matrix1 and Vector1 would overlap:



where the shaded area indicates the overlap.



## 4.9 Array Expressions

Array expressions are expressions which involve arrays (vectors or matrices) as operands.

All the operands of an array expression must have identical sizes and dimensions. The result of an array expression is an array of the same size and dimension as the array operands.

Example:

$$A[3] = [ \dots ];$$

$$B[3] = [ \dots ];$$

$$C[3] = A * \cos(B) + 3.0;$$

This array expression is interpreted as the equivalence of

$$C[k] = A[k] * \cos(B[k]) + 3.0, \quad k = 1, 2, 3$$

The algebraic operations and mathematical functions in an array expression are applied to individual elements of the operand arrays with the same offset. You may visualize array expressions as a short-hand notation for a set of parallel scalar expressions involving the corresponding elements of the operand arrays.



Array expressions do not lead to inner or outer products of vectors and matrices (such products can be obtained using array functions, [see](#) Section 4.10).

An example of array expressions with matrix operands:

$$A[3,5] = [ \dots ];$$

$$B[3,5] = [ \dots ];$$

$$C[3,5] = A * \cos(B);$$

which is interpreted as

$$C[i,j] = A[i,j] * \cos(B[i,j]), \quad i = 1, 2, 3, \quad j = 1, 2, \dots, 5$$

## Alignment by Offsets

Arrays of different first indices are aligned by using element offsets instead of element indices. For example,

```
D[1:5] = [ ... ];
F[101:105] = [ ... ];
G[0:4] = D + F;
```

where the vectors have identical length but different first indices. In such cases, the operation applies to array elements of the same offset, where offset is defined as

$$\text{offset} = \text{index} - \text{first\_index}$$

In the above example, the vector **G** is computed as

```
G[0] = D[1] + F[101]
G[1] = D[2] + F[102]
...
G[4] = D[5] + F[105]
```

## Scalar Operands

Array expressions may include references to scalar constants, labels and individual elements of other arrays.

Example,

```
X[10] = 5;
```

which in effect sets all the elements of **X** to the same value of 5.

Another example:

```
X[10] = [ .... ];
X_normalized[10] = X / X[1];
```

means

$$X\_normalized[k] = X[k] / X[1], \quad k = 1, \dots, 10$$

## 4.10 Array Functions

Array functions are functions which accept arrays as arguments. The set of array functions available in OSA90 is listed in Table 4.5. These functions provide powerful and efficient tools for manipulating vectors and matrices, solving linear systems, etc.

**TABLE 4.5 LIBRARY OF ARRAY FUNCTIONS**

Function	Description
AABS( $x$ )	absolute values of the elements of array $x$
AMAX( $x$ )	maximum of the elements of array $x$
AMIN( $x$ )	minimum of the elements of array $x$
SUM( $x$ )	sum of all the elements of array $x$
PRODUCT( $x,y$ )	product of arrays $x$ and $y$
SUBSET( $x,k$ )	sub-vector of vector $x$ starting from $x[k]$
SUBSET( $x,i,j$ )	sub-matrix of matrix $x$ starting from $x[i,j]$
ROW( $x,k$ )	the row of matrix $x$ with row index $k$
COL( $x,k$ )	the column of matrix $x$ with column index $k$
TRANPOSE( $x$ )	transposition of matrix $x$
INVERSE( $x$ )	inverse of matrix $x$
SOLVE( $x,y$ )	solution of a linear system of equations with the coefficient matrix $x$ and the right-hand-side vector $y$
LU( $x$ )	LU factorization of matrix $x$
SUBST( $x,y$ )	solution of a linear system of equations by forward and backward substitutions with the LU factors $x$ and the right-hand-side vector $y$
SUBSTT( $x,y$ )	solution of a transposed linear system by forward and backward substitutions with the LU factors $x$ and the right-hand-side vector $y$
LUF( $x$ )	LU factorization of matrix $x$ without pivoting
EXTRACT_L( $x$ )	extracts the lower-triangular matrix L from the LU factors $x$ which must be obtained from LUF()
EXTRACT_U( $x$ )	extracts the upper-triangular matrix U from the LU factors $x$ which must be obtained from LUF()
INDEX( $x,y$ )	finds the index of the scalar $y$ in the array $x$
QR( $A,X,V$ )	QR factorization of matrix $A$ for eigenvalues and eigenvectors
MEAN( $x$ )	mean value of the data in array $x$
StdDev( $x$ )	standard deviation of the data in array $x$
CORR( $x,y$ )	correlation coefficient between arrays $x$ and $y$
CORR( $x$ )	correlation matrix for the data matrix $x$
SKEW( $x,k$ )	skewness of the data in array $x$
KURT( $x$ )	kurtosis of the data in array $x$

## Absolute Values of Array Elements

**Syntax**

$$A[n] = \text{AABS}(B);$$

where  $A$  and  $B$  are vectors of identical size,  $A[\cdot] = |B[\cdot]|$

Example:

$$B[4] = [ 1 \ -2 \ 3 \ -4 ];$$

$$A[4] = \text{AABS}(B);$$

The result is  $A = [ 1 \ 2 \ 3 \ 4 ]$ .

AABS can also be applied to matrices:

**Syntax**

$$A[n,m] = \text{AABS}(B);$$

where  $A$  and  $B$  are matrices of identical dimensions,  $A[\cdot,\cdot] = |B[\cdot,\cdot]|$

Example:

$$B[4,2] = \begin{bmatrix} -1 & -2 \\ -3 & -4 \\ 2 & 1 \\ -5 & 3 \end{bmatrix};$$

$$A[4,2] = \text{AABS}(B);$$

The result is

$$A[4,2] = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 2 & 1 \\ 5 & 3 \end{bmatrix};$$

## Maximum of Array Elements

### Syntax

```
v = AMAX(A);
```

where  $v$  is a scalar label and  $A$  is a vector,  $v = \max \{A[\cdot]\}$

### Example:

```
A[10] = [ 1 2 3 4 5 9 8 7 6 0 ];
```

```
v = AMAX(A);
```

The result is  $v = 9$ .

AMAX can also be applied to the column vectors of a matrix:

### Syntax

```
v[m] = AMAX(A);
```

where  $A$  is an  $n \times m$  matrix,  $v[i] = \max \{A[\cdot, i]\}$

### Example:

```
B[4,2] = [ -1 -2
           -3 -4
            2  1
           -5  3 ];
```

```
v[2] = AMAX(B);
```

The result is  $v = [2 \ 3]$ .

You can obtain the maximum of all the elements of a matrix using AMAX of AMAX, e.g.,

```
v = AMAX(AMAX(B));
```

The result is  $v = 3$ .

## Minimum of Array Elements

**Syntax:**

$$v = \text{AMIN}(A);$$

where  $v$  is a scalar label and  $A$  is a vector,  $v = \min \{A[\cdot]\}$

**Example:**

$$A[10] = [ 1 -2 3 -4 5 -9 8 -7 6 0 ];$$

$$v = \text{AMIN}(A);$$

The result is  $v = -9$ .

AMIN can also be applied to the column vectors of a matrix:

**Syntax:**

$$v[m] = \text{AMIN}(A);$$

where  $A$  is an  $n \times m$  matrix,  $v[i] = \min \{A[\cdot, i]\}$

**Example:**

$$B[4,2] = \begin{bmatrix} -1 & -2 \\ -3 & -4 \\ 2 & 1 \\ -5 & 3 \end{bmatrix};$$

$$v[2] = \text{AMIN}(B);$$

The result is  $v = [-5 \ -4]$ .

You can obtain the minimum of all the elements of a matrix using AMIN of AMIN, e.g.,

$$v = \text{AMIN}(\text{AMIN}(B));$$

The result is  $v = -5$ .

## Sum of Array Elements

**Syntax:**

$$v = \text{SUM}(A);$$

where  $v$  is a scalar label and  $A$  is a vector,  $v = \sum A[\cdot]$

**Example:**

$$A[10] = [ 1 2 3 4 5 6 7 8 9 10 ];$$

$$v = \text{SUM}(A);$$

The result is  $v = 55$ .

SUM can also be applied to the column vectors of a matrix:

**Syntax:**

$$v[m] = \text{SUM}(A);$$

where  $A$  is an  $n \times m$  matrix,  $v[i] = \sum A[\cdot, i]$

**Example:**

$$B[4,2] = \begin{bmatrix} -1 & -2 \\ -3 & -4 \\ 2 & 1 \\ -5 & 3 \end{bmatrix};$$

$$v[2] = \text{SUM}(B);$$

The result is  $v = [-7 \ -2]$ .

You can obtain the sum over all the elements of a matrix using SUM of SUM, e.g.,

$$v = \text{SUM}(\text{SUM}(B));$$

The result is  $v = -9$ .

## Inner Product of Two Vectors

**Syntax:**

$$v = \text{PRODUCT}(A, B);$$

where  $v$  is a scalar label, and  $A$  and  $B$  are two vectors of identical length,  

$$v = \Sigma (A[\cdot] \times B[\cdot])$$

Example:

$$\begin{aligned} A[5] &= [ 1 \ 2 \ 3 \ 4 \ 5 ]; \\ B[5] &= [ 5 \ 4 \ 3 \ 2 \ 1 ]; \end{aligned}$$

$$v = \text{PRODUCT}(A, B);$$

The result is  $v = 35$ .

In comparison, the array expression

$$C[5] = A * B;$$

leads to  $C = [5 \ 8 \ 9 \ 8 \ 5]$ .

## Product of a Matrix and a Vector

**Syntax:**

$$A[n] = \text{PRODUCT}(B, C);$$

where  $A$  and  $C$  are two vectors and  $B$  is a matrix,  $A[k] = \Sigma (B[k, \cdot] \times C[\cdot])$ .  
 The length of  $A$  is  $n$ , and if the length of  $C$  is  $m$ , then  $B$  must be an  $n \times m$  matrix.

Example:

$$\begin{aligned} B[2,3] &= \begin{bmatrix} 1 & 2 & 3 \\ & 4 & 5 & 6 \end{bmatrix}; \\ C[3] &= [ 7 \ 8 \ 9 ]; \end{aligned}$$

$$A[2] = \text{PRODUCT}(B, C);$$

The result is  $A = [50 \ 122]$ .



## Product of a Transposed Matrix and a Vector

### Syntax

$$A[n] = \text{PRODUCT}(B, C);$$

where  $A$  and  $B$  are vectors and  $C$  is a matrix.

$$A[k] = \sum (B[\cdot] \times C[\cdot, k]), \text{ i.e., } A = C^T B.$$

The length of  $A$  is  $n$ , and if the length of  $B$  is  $m$ , then  $C$  must be an  $m \times n$  matrix.

### Example:

$$\begin{aligned} B[2] &= [ 1 \ 2 ]; \\ C[2,3] &= [ 3 \ 4 \ 5 \\ &\quad 6 \ 7 \ 8 ]; \end{aligned}$$

$$A[3] = \text{PRODUCT}(B, C);$$

The result is  $A = [15 \ 18 \ 21]$ , i.e., the product of matrix  $C$  transposed and vector  $B$ .

## Product of Two Matrices

### Syntax

$$A[n,m] = \text{PRODUCT}(B, C);$$

where  $A$ ,  $B$  and  $C$  are matrices,  $A[i,j] = \sum (B[i,\cdot] \times C[\cdot,j])$ .

If  $B$  is an  $n \times k$  matrix, then  $C$  must be a  $k \times m$  matrix.

### Example:

$$B[2,3] = [ 1 \ 2 \ 3 \\ 4 \ 5 \ 6 ];$$

$$C[3,4] = [ -1 \ -2 \ 1 \ 2 \\ -3 \ -4 \ 3 \ 4 \\ -5 \ -6 \ 5 \ 6 ];$$

$$A[2,4] = \text{PRODUCT}(B, C);$$

The result is

$$A[2,4] = [ -22 \ -28 \ 22 \ 28 \\ -49 \ -64 \ 49 \ 64 ].$$

## Sub-Vector

**Syntax**

$$A[n] = \text{SUBSET}(B, k);$$

where  $A$  and  $B$  are vectors and  $k$  is an integer constant or label.

$A[i] = B[k + i - 1]$ , the length of  $B$  must be at least  $n + k - 1$ .

Example:

$$B[10] = [ 1 2 3 4 5 6 7 8 9 10 ];$$

$$A[3] = \text{SUBSET}(B, 5);$$

The result is  $A = [5 6 7]$ .

Another example:

$$B[-10:-1] = [ 1 2 3 4 5 6 7 8 9 10 ];$$

$$A[3] = \text{SUBSET}(B, -9);$$

The result is  $A = [2 3 4]$ .

## Sub-Matrix

**Syntax**

$$A[n, m] = \text{SUBSET}(B, i, j);$$

where  $A$  and  $B$  are matrices and  $i$  and  $j$  are integer constants or labels.

$A$  will be a sub-matrix of  $B$  with  $A[1,1]$  being  $B[i,j]$ .

$B$  must be able to accommodate  $A$ , i.e.,  $A$  must not extend beyond the dimensions of  $B$ .

Example:

$$B[4,4] = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix};$$

$$A[2,2] = \text{SUBSET}(B, 3, 2);$$

The result is

$$A[2,2] = \begin{bmatrix} 10 & 11 \\ 14 & 15 \end{bmatrix}.$$

**Row of a Matrix****Syntax:**

$$A[n] = \text{ROW}(B, k);$$

where  $A$  is a vector,  $B$  is a matrix and  $k$  is an integer constant or label,  
 $A[\cdot] = B[k, \cdot]$ .

**Example:**

$$B[2, 3] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix};$$

$$A[3] = \text{ROW}(B, 2);$$

The result is  $A = [4 \ 5 \ 6]$ .

**Column of a Matrix****Syntax:**

$$A[n] = \text{COL}(B, k);$$

where  $A$  is a vector,  $B$  is a matrix and  $k$  is an integer constant or label,  
 $A[\cdot] = B[\cdot, k]$ .

**Example:**

$$B[2, 3] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix};$$

$$A[2] = \text{COL}(B, 2);$$

The result is  $A = [2 \ 5]$ .

## Transposition of a Matrix

**Syntax:**

$$A[n,m] = \text{TRANSPPOSE}(B);$$

where  $A$  and  $B$  are matrices,  $B$  must be an  $m \times n$  matrix,  $A[i,j] = B[j,i]$ .

**Example:**

$$B[2,3] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix};$$

$$A[3,2] = \text{TRANSPPOSE}(B);$$

The result is

$$A[3,2] = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}.$$

## Inverse of a Matrix

**Syntax:**

$$A[n,n] = \text{INVERSE}(B);$$

where  $A$  and  $B$  are square matrices of identical dimensions.

**Example:**

$$B[2,2] = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix};$$

$$A[2,2] = \text{INVERSE}(B);$$

The result is

$$A[2,2] = \begin{bmatrix} -2 & 1 \\ 1.5 & -0.5 \end{bmatrix}.$$

The matrix to be inverted must be non-singular, otherwise you will see an error message when the program attempts to compute the inverse during simulation.

## Solution of a Linear System of Equations

**Syntax:**

```
A[n] = SOLVE(B,C);
```

where  $A$  and  $C$  are vectors of length  $n$  and  $B$  must be an  $n \times n$  square matrix.  $B$  is the coefficient matrix of a linear system of equations,  $C$  is the right-hand-side vector and  $A$  will be the solution vector.

**Example:**

```
B[2,2] = [ 1  2  
          3  4 ];  
C[2] = [ 1  0 ];  
A[2] = SOLVE(B,C);
```

The result is  $A = [-2 \ 1.5]$ .

The linear system of equations is solved by LU factorization of the coefficient matrix, followed by forward and backward substitutions with the right-hand-side vector. The coefficient matrix must be non-singular, otherwise you will see an error message when the program attempts to solve the equations during simulation.



The coefficient matrix is not overwritten by the computed LU factors. The LU factors of the coefficient matrix are computed every time SOLVE() is called.

## Solution of Linear Equations by LU Factorization

**Syntax:**

$$A[m,n] = \text{LU}(B);$$

where  $B$  must be an  $n \times n$  coefficient matrix of a linear system of equations.  $A$  is an  $m \times n$  matrix,  $m = n + 1$ , which stores the LU factors and pivoting information to be used in conjunction with the following functions.

$$c[n] = \text{SUBST}(A,d);$$

$c$  is the solution to the linear system of equations obtained by forward and backward substitutions with the right-hand-side vector  $d$ , i.e.,  $Bc = d$ .

$$e[n] = \text{SUBSTT}(A,f);$$

$e$  is the solution to the transposed linear system of equations obtained by forward and backward substitutions with the right-hand-side vector  $f$ , i.e.,  $B^T e = f$ .

**Example:**

$$B[3,3] = \begin{bmatrix} 1 & 3 & 5 \\ 6 & 4 & 2 \\ 7 & 9 & 8 \end{bmatrix};$$

$$D[3] = [ 14 \quad 28 \quad 47 ];$$

$$F[3] = [ 34 \quad 38 \quad 33 ];$$

$$A[4,3] = \text{LU}(B);$$

$$C[3] = \text{SUBST}(A,D);$$

$$E[3] = \text{SUBSTT}(A,F);$$
**The results are**

$$C[3] = [ 3 \quad 2 \quad 1 ],$$

$$E[3] = [ 1 \quad 2 \quad 3 ].$$


The LU factors are stored in a compact format in the rectangular matrix produced by the LU() function. If you try to analyze the contents of the matrix, you may find the information difficult to interpret, especially when pivoting is required and has been performed during the LU factorization.

## Comparison of INVERSE, SOLVE and LU

You can solve a linear system of equations, such as

$$A x = y$$

using three different array functions:

- (1)  $B = \text{INVERSE}(A)$   
 $x = \text{PRODUCT}(B,y)$
- (2)  $x = \text{SOLVE}(A,y)$
- (3)  $B = \text{LU}(A)$   
 $x = \text{SUBST}(B,y)$

where the notation of array dimensions has been omitted for brevity.

Method (1), by matrix inversion, is the most computationally intensive. It should be chosen only when you are also interested in the inverse matrix in addition to the solution.

Method (2) is the most straightforward approach. It has the simplest syntax and should be the best choice if you need the solution for only one right-hand-side vector.

Method (3) is very efficient if you need a number of solutions with the same coefficient matrix but different right-hand-side vectors.

To solve the transposed system, i.e.,

$$A^T x = y$$

you can use

- (1)  $B = \text{INVERSE}(A)$   
 $x = \text{PRODUCT}(y,B)$
- (2)  $B = \text{TRANSPOSE}(A)$   
 $x = \text{SOLVE}(B,y)$
- (3)  $B = \text{LU}(A)$   
 $x = \text{SUBST}(B,y)$
- (4)  $B = \text{TRANSPOSE}(A)$   
 $C = \text{LU}(B)$   
 $x = \text{SUBST}(C,y)$

## LU Factorization without Pivoting

### Syntax

```
A[m,n] = LUF(B);
```

where  $B$  must be an  $n \times n$  matrix whose LU factors can be computed without pivoting.  $A$  is an  $m \times n$  matrix,  $m = n + 1$ , which stores the LU factors. The last row of  $A$  contains "pivots" which are always set to 0, 1, ...,  $n - 1$ . This puts  $A$  in a consistent format with the result from `LU()`.

### Example:

```
B[3,3] = [ ... ];
A[4,3] = LUF(B);
```

The formats of the LU factors produced by the `LU()` and `LUF()` functions are consistent. Therefore, you can use the LU factors computed by `LUF()` to find the solutions of a linear system of equations in the same way as the `LU()` function would be used:

```
c[n] = SUBST(A,d);
```

$c$  is the solution to the linear system of equations obtained by forward and backward substitutions with the right-hand-side vector  $d$ , i.e.,  $Bc = d$ .

```
e[n] = SUBSTT(A,f);
```

$e$  is the solution to the transposed linear system of equations obtained by forward and backward substitutions with the right-hand-side vector  $f$ , i.e.,  $B^T e = f$ .

## Extracting the L and U Factors

You can use the functions `EXTRACT_L()` and `EXTRACT_U()` to separate the L and U factors and store them in square matrices.

### Example:

```
B[3,3] = [ ... ];
A[4,3] = LUF(B);

L[3,3] = EXTRACT_L(A);
U[3,3] = EXTRACT_U(A);
```

The argument to the `EXTRACT_L()` and `EXTRACT_U()` functions must have been obtained from the `LUF()` function. If you use the result from `LU()` which may involve pivoting in the factorization, you will get an error message.



## Finding the Index of an Array Element

### Syntax

```
i = INDEX(A, x);
```

where  $A$  is a vector and  $i$  is the index such that  $A[i] = x$ .  
 $i$  is set to 0 if  $x$  does not match any element of  $A$ .

Example:

```
A[6] = [ 1.2 -3.5 0.1 16 -2 0 ];
X = 0.1;
I = INDEX(A, X);
```

The result is  $I = 3$ .

The function INDEX() can be applied to matrices as well:

### Syntax

```
i[2] = INDEX(A, x);
```

where  $A$  is a matrix (i.e., a two-dimensional array) and  $i$  contains two indices such that  $A[j, k] = x$ , where  $j = i[1]$  and  $k = i[2]$ .  
 $i$  is set to  $[0 \ 0]$  if  $x$  does not match any element of  $A$ .

Example:

```
A[3,4] = [ 1.2 -3.5 0.1 6.0
           -2.0 0.0 7.2 0.3
           1.9 0.6 -4.4 -8.6];
```

```
X = 7.2;
```

```
I[2] = INDEX(A, X);
```

The result is  $I = [2 \ 3]$ .

## Eigenvalues and Eigenvectors of Matrix

**Syntax:**

```
QR(A,X[n],V[n,n]);
```

where  $A$  is an  $n \times n$  real symmetrical matrix. Upon return,  $X$  contains the eigenvalues of  $A$  and the rows of  $V$  contain the corresponding eigenvectors.

**Example:**

```
A[3,3] = [ 7.3  1.9  2.8
          1.9  8.2  6.4
          2.8  6.4  9.1 ];
```

```
QR(A,X[3],V[3,3]);
```

**The result is**

```
X[3] = [16.3094  6.1197  2.1709]
V[3,3] = [ 0.348566  0.629626  0.694315
           0.929453 -0.327753 -0.169396
          -0.120907 -0.704379  0.699451 ]
```

The rows of  $V$  are the eigenvectors. For instance,

```
V2[3] = ROW(V,2);
```

is the eigenvector corresponding to the eigenvalue  $X[2]$ .

You can verify this by comparing

```
P1[3] = PRODUCT(A,V2);
```

with

```
P2[3] = X[2] * V2;
```

$P1$  and  $P2$  should be identical.

## Mean Values of Data

### Syntax

$$v = \text{MEAN}(X);$$

where  $X$  is a vector of size  $n$ ,  $v = (\sum X[\cdot]) / n$

### Example:

$$X[10] = [ 1 2 3 4 5 6 7 8 9 10 ];$$

$$V = \text{MEAN}(X);$$

The result is  $V = 5.5$ .

MEAN can also be applied to the column vectors of a matrix:

### Syntax

$$v[m] = \text{MEAN}(A);$$

where  $A$  is an  $n \times m$  matrix,  $v[i] = (\sum A[\cdot, i]) / n$

### Example:

$$A[5,3] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & 15 \end{bmatrix};$$

$$V[3] = \text{MEAN}(A);$$

The result is  $V = [7 \ 8 \ 9]$ .

## Standard Deviations of Data

**Syntax:**

```
v = StdDev(X);
```

where  $X$  is a vector of size  $n$ , the standard deviation of  $X$  is given by

$$v = \sqrt{(\sum(X[\cdot] - X0)^2) / (n - 1)}$$

where  $X0$  denotes the mean of  $X$ .

Example:

```
X[20] = [ 1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0  0.0
          1.1  2.2  3.3  4.4  5.5  6.6  7.7  8.8  9.9  10.0 ];
```

```
V = StdDev(X);
```

The result is  $V = 3.11057$ .

StdDev can also be applied to the column vectors of a matrix:

**Syntax:**

```
v[m] = StdDev(A);
```

where  $A$  is an  $n \times m$  matrix, the standard deviations of the columns are given by

$$v[i] = \sqrt{(\sum(A[\cdot, i] - Ai0)^2) / (n - 1)}$$

where  $Ai0$  denotes the mean value of the  $i$ th column of  $A$ .

Example:

```
A[50,3] = [ ... ];
```

```
V[3] = StdDev(A);
```

## Correlation Coefficient Between Two Sets of Data

### Syntax:

$c = \text{CORR}(X, Y);$

where  $X$  and  $Y$  are vectors of size  $n$ , the correlation coefficient is given by

$$c = (\Sigma(X[\cdot] - X0)(Y[\cdot] - Y0)) / ((n - 1) \sigma_x \sigma_y)$$

where  $X0$  and  $Y0$  denote the means of  $X$  and  $Y$ , respectively, and  $\sigma_x$  and  $\sigma_y$  denote the standard deviations of  $X$  and  $Y$ , respectively.

### Example:

$X[10] = [ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 ];$   
 $Y[10] = [ 1 \ 2 \ 4 \ 8 \ 16 \ 32 \ 64 \ 128 \ 256 \ 512 ];$

$C = \text{CORR}(X, Y);$

The result is  $C = 0.798837$ .

## Correlation Matrix

### Syntax:

$C[m, m] = \text{CORR}(A);$

where  $A$  is an  $n \times m$  matrix.  $C$  is the correlation matrix.  $C[i, j]$  represents the correlation coefficient between the  $i$ th and  $j$ th columns of  $A$ .

$C$  is a symmetrical matrix and the value of its diagonal elements is always 1.

### Example:

$A[50, 3] = [ \dots ];$

$C[3, 3] = \text{CORR}(A);$

## Skewness of Statistical Data

**Syntax:**

```
v = SKEW(X);
```

where  $X$  is a vector of size  $n$ , the skewness of  $X$  is given by

$$v = (\Sigma(X[\cdot] - X0) / \sigma_x)^3 / n$$

where  $X0$  and  $\sigma_x$  denote the mean and standard deviation of  $X$ , respectively.

**Example:**

```
X[20] = [ 1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0  0.0
          1.1  2.2  3.3  4.4  5.5  6.6  7.7  8.8  9.9  10.0 ];
```

```
V = SKEW(X);
```

The result is  $V = -0.023745$ .

SKEW can also be applied to the column vectors of a matrix:

**Syntax:**

```
v[m] = SKEW(A);
```

where  $A$  is an  $n \times m$  matrix and  $v[i]$  contains the skewness of the  $i$ th column of  $A$ .

**Example:**

```
A[50,3] = [ ... ];
```

```
V[3] = SKEW(A);
```

## Kurtosis of Statistical Data

### Syntax

$v = \text{KURT}(X);$

where  $X$  is a vector of size  $n$ , the kurtosis of  $X$  is given by

$$v = ((\Sigma((X[\cdot] - X0) / \sigma_x)^4) / n) - 3$$

where  $X0$  and  $\sigma_x$  denote the mean and standard deviation of  $X$ , respectively.

### Example:

```
X[20] = [ 1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0  0.0
          1.1  2.2  3.3  4.4  5.5  6.6  7.7  8.8  9.9  10.0 ];
```

```
V = KURT(X);
```

The result is  $V = -1.36516$ .

KURT can also be applied to the column vectors of a matrix:

### Syntax

$v[m] = \text{KURT}(A);$

where  $A$  is an  $n \times m$  matrix and  $v[i]$  contains the kurtosis of the  $i$ th column of  $A$ .

### Example:

```
A[50,3] = [ ... ];
```

```
V[3] = KURT(A);
```

## Embedded Array Functions

References to array functions can be embedded in expressions.

Examples:

```
A[20,3] = [ ... ];
B[20] = COL(A,1) + COL(A,3);

V[10] = [ ... ];
D[10] = V - SUM(V) / 10;
```

References to array functions can also be used directly as arguments to another array function.

Example:

```
A[5,4] = [ ... ];
B[4,3] = [ ... ];

ABK[5] = PRODUCT(A, COL(B,K));

inverse_A_transpose_A[4,4] = INVERSE(PRODUCT(TRANPOSE(A), A));
```

## SUBSET Cannot Be Embedded

The dimension of the result of an embedded reference to an array function is determined implicitly from the dimensions of the arguments.

Embedded reference to the array function SUBSET is not allowed, because the dimension of the result cannot be determined implicitly.

For example,

```
A[100] = [ ... ];

AA[5] = SUBSET(A,K);
```

AA is a 5-element sub-vector of A, starting from the Kth element of A. The dimension of the result is explicitly defined by the dimension of AA. It cannot be determined implicitly from the reference SUBSET(A,K) alone.



## 4.11 String Labels

String labels can be defined to represent character strings instead of numerical values.

### Syntax

```
char label_name[]: "character_string";
```

```
char label_name[n]: "character_string";
```

where  $n$  is the size of the string label,  $16384 \geq n \geq m$ ,  $m$  represents the length of *character\_string* including the NULL byte. If omitted,  $n$  is set to  $m$ .

### Examples:

```
char Greetings[] = "hello, world";
```

```
char Filename[12] = "circuit.dat";
```



The size of string labels must include the NULL byte which terminates a character string. In the above example, the size of `Filename` must be no less than 12, though "circuit.dat" contains only 11 characters.

String labels are useful for passing text information between OSA90 and external programs via Datapipe (Chapter 5), e.g., passing an input file to an external simulator in its own syntax format.

## Multi-Line Strings

Multi-line strings can be defined in the natural way, any and all blanks and new line characters between the quotation marks are preserved as part of the string.

### Example:

```
char Multi_Line_Text[] = "
    Called from OSA90 via Datapipe.
    Calculate the transmitter characteristics.
    Script file name: /user/charlie/trnsmtrr.scr
    Output data file name: /osa90/data/trnsmtrr.dat
";
```

which defines a string label to represent 6 lines of text, the first and last lines being blank lines.

## Control Characters

Control characters such as the escape sequences used by some programming languages, are not supported by OSA90. For instance, the new-line character in "C", namely "\n", will not be interpreted but simply treated as two separate characters "\" and "n".

## Literal Quotation Marks

You can use two consecutive double quotation marks (""") to represent one literal double quotation mark (").

Example:

```
char Title[] = "Data from the ""Child"" Program";
```

This defines a label `Title` to represent the character string

```
Data from the "Child" Program
```

## 4.12 Transformations

For the user's convenience, OSA90 has several common numerical transformations as built-in features: polar form to rectangular form, rectangular form to polar form, and the discrete Fourier transforms.

Polar and rectangular forms are two standard representations of complex numbers. They represent a complex number by two real numbers: magnitude and phase in the polar form; real and imaginary parts in the rectangular form. The built-in transformations allow you to convert one form to the other.

The discrete Fourier transform (DFT) is useful for converting a periodic signal between its frequency-domain complex spectrum and time-domain waveform.

### Polar to Rectangular

#### Syntax:

```
MP2RI(mag, ang, real, imag);
```

where *mag* and *ang* are inputs (numerical constants or labels) representing the magnitude and phase (in degrees) of a complex number.

The output labels *real* and *imag* must not duplicate any existing label names.

$$real = mag * \cos(ang), \quad imag = mag * \sin(ang).$$

#### Example:

```
Mag = 3;
Ang = 30Deg;

MP2RI(Mag, Ang, Real, Imag);
```

The results are Real = 2.598 and Imag = 1.5.

The transformation can also be applied to arrays, e.g.,

```
Mag[10] = [ ... ];
Ang[10] = [ ... ];

MP2RI(Mag, Ang, Real[10], Imag[10]);
```

All the arrays must have identical dimensions.



The outputs of MP2RI() are in effect definitions of new labels (scalars or arrays). They can be used just like any other user-defined labels.

## Rectangular to Polar

**Syntax:**

```
RI2MP(real, imag, mag, ang);
```

where *real* and *imag* are inputs (numerical constants or labels) representing the real and imaginary parts of a complex number.

The output labels *mag* and *ang* must not duplicate any existing label names.

```
mag = sqrt(real * real + imag * imag),      ang = atan2(imag,real).
```

**Example:**

```
Real = 3;
Imag = 4;
```

```
RI2MP(Real, Imag, Mag, Ang);
```

The results are *Mag* = 5 and *Ang* = 53.13 (degrees).

The transformation can also be applied to arrays, e.g.,

```
Real[10] = [ ... ];
Imag[10] = [ ... ];
```

```
RI2MP(Real, Imag, Mag[10], Ang[10]);
```

All the arrays must have identical dimensions.



The outputs of RI2MP() are in effect definitions of new labels (scalars or arrays). They can be used just like any other user-defined labels.

## DFT Time to Frequency

**Syntax:**

```
DFT_TF(x_time, x_real[0:n], x_imag[0:n]);
```

**Example:**

```
VT[9] = [1.282 1.151 1.900 4.384 6.611 6.189 6.373 4.729 2.390];
```

```
DFT_TF(VT, VR[0:4], VI[0:4]);
```

**The results are**

```
VR[0:4] = [3.89 -1.35 0.1648 -0.3969 0.08222],
```

```
VI[0:4] = [0.0 2.63 0.007732 0.08699 -0.1459].
```

**DFT\_TF** implements the Discrete Fourier Transform (DFT) to convert a periodic signal from its time-domain waveform to its frequency-domain complex spectrum.

The input *x\_time* is a vector containing time-domain waveform samples of the signal. The number of samples is implied by the length of the vector *x\_time*. Uniform sampling over one complete period is assumed, i.e., the time points are assumed to be

$$0, T_0/m, 2 \times T_0/m, \dots, (m-1) \times T_0/m$$

where  $T_0$  is the period and  $m$  is the number of time points (i.e., the length of the vector *x\_time*).

The output vectors *x\_real* and *x\_imag* will contain the real and imaginary parts of the frequency-domain spectrum of the signal. It is recommended that they are defined with 0 as the first index and  $n$  as the last index,  $n$  being the highest harmonic to be included in the DFT. Then, *x\_real[k]* and *x\_imag[k]* correspond to the  $k$ th harmonic component of the spectrum, e.g., *x\_real[1]* and *x\_imag[1]* are the real and imaginary parts of the fundamental component of the spectrum. *x\_real[0]* is the DC component of the spectrum, and *x\_imag[0]* is always set to zero.

OSA90 uses the one-sided cosine DFT. The time-domain waveform is related to the frequency-domain spectrum as

$$x(t) = \sum_{k=0}^n (x\_real[k] \times \cos(2\pi \frac{kt}{T_0}) + x\_imag[k] \times \sin(2\pi \frac{kt}{T_0}))$$



To guarantee the recoverability of the time-domain sample from the calculated spectrum,  $n$  must be at least  $(m-1)/2$ .

## DFT Frequency to Time

The transformation `DFT_FT()` implements the inverse Discrete Fourier Transform to convert the frequency-domain complex spectrum of a periodic signal to its time-domain waveform.

Four different formats are available for using `DFT_FT()`, allowing flexibility in terms of specifying the time period, the number of time points, the number of tones, etc.

### DFT\_FT Format 1: Standard One Period

**Syntax:**

```
x_time[m] = DFT_FT(x_real, x_imag);
DFT_FT(x_real, x_imag, x_time[m]);
```

The input vectors `x_real` and `x_imag` supply the real and imaginary parts of the complex frequency-domain spectrum for consecutive harmonic frequencies, starting from 0 (DC). The sizes of `x_real` and `x_imag` must be identical and their size implies the number of harmonics included in the spectrum. For instance, if they are defined as `x_real[0:n]` and `x_imag[0:n]`, then `n` is the highest harmonic index in the spectrum.

The output vector `x_time` will contain the calculated time-domain samples of the signal at `m` uniformly spaced time points over one period.

**Example:**

```
VR[0:4] = [3.89  -1.35  0.1648  -0.3969  0.08222];
VI[0:4] = [0.0  2.63  0.007732  0.08699  -0.1459];

VT[9] = DFT_FT(VR,VI);
```

The result is

```
VT[9] = [1.282  1.151  1.900  4.384  6.611  6.189  6.373  4.729  2.390].
```

### Function or Subroutine

The transformation `DFT_FT()` can be invoked as a function, such as

```
VT[9] = DFT_FT(VR,VI);
```

or as a "subroutine", such as

```
DFT_FT(VR, VI, VT[9]);
```

Both calls produce identical results.

## DFT\_FT Format 2: Specific Time Range

### Syntax

```
x_time[m] = DFT_FT(x_real, x_imag, t0, t1);
DFT_FT(x_real, x_imag, t0, t1, x_time[m]);
```

The inputs *x\_real* and *x\_imag* are vectors of the same description as in **Format 1**.

The main difference from **Format 1** is that instead of sampling the time points over one period, you can specify the range of time points by the input parameters *t0* and *t1*.

The parameters *t0* and *t1* must be scalar constants or labels, and they represent a time range normalized with respect to the period. The *m* time points are uniformly spaced in the interval  $[t0 \cdot T_0 \ t1 \cdot T_0]$ , where  $T_0$  is the period.

The output vector *x\_time* will contain the calculated time-domain samples of the signal at the *m* time points.

Example:

```
VR[0:4] = [ ... ];
VI[0:4] = [ ... ];
VT[51] = DFT_FT(VR, VI, 0, 2];
```

where 51 time points between 0 and 2 periods are specified.

## DFT\_FT Format 3: Specify Spectral Frequencies

### Syntax

```
x_time[m] = DFT_FT(x_real, x_imag, frequency, t0, t1);
DFT_FT(x_real, x_imag, frequency, t0, t1, x_time[m]);
```

The inputs *x\_real* and *x\_imag* are vectors of the same description as in **Formats 1 and 2**.

The main difference from **Format 2** is the additional input vector *frequency* which contains the spectral frequencies. The size of *frequency* must be identical to the size of *x\_real* and *x\_imag*, and *frequency*[0] must be zero (DC). Including *frequency* as an argument allows you to use **DFT\_FT()** in cases where the spectral frequencies are not simply the consecutive multiples of the fundamental frequency, such as multitone intermodulation frequencies.

The spectrum data in *x\_real* and *x\_imag* must correspond to the specified frequencies.

In this format, the input parameters  $t0$  and  $t1$  specify the actual time range (in Format 2  $t0$  and  $t1$  are normalized by the period).

The output vector  $x\_time$  will contain the calculated time-domain samples of the signal at  $m$  uniformly spaced time points in the interval  $[t0\ t1]$ .

Example:

```
VR[0:4] = [3.89 -1.35 0.1648 -0.3969 0.08222];
VI[0:4] = [0.0 2.63 0.007732 0.08699 -0.1459];
FO = 3GHz;
F[0:4] = [0 FO (2 * FO) (3 * FO) (4 * FO)];
Two_TO = 2.0 / FO;

VT[51] = DFT_FT(VR, VI, F, 0, Two_TO);
```

#### DFT\_FT Format 4: Single Time Point

##### Syntax

```
 $x\_time$  = DFT_FT( $x\_real$ ,  $x\_imag$ , frequency,  $t$ );
DFT_FT( $x\_real$ ,  $x\_imag$ , frequency,  $t$ ,  $x\_time$ );
```

The inputs  $x\_real$  and  $x\_imag$  are vectors of the same description as in Format 3.

This format is different from the others because it calculates the time-domain signal at a single time point which is specified by the input parameter  $t$ .

The output  $x\_time$  must be a scalar label. Using this format, DFT\_FT() can be called repeatedly with different values of  $t$  (i.e., sweep  $t$ , [Chapter 9](#)).

The advantage is that when you wish to perform DFT for a large number of time points, you can avoid defining a large output vector  $x\_time$  to store all the time points. Instead, you obtain the result for one time point at a time.

Example:

```
VR[0:4] = [ ... ];
VI[0:4] = [ ... ];
F[0:4] = [ ... ];
T = ...;

VT = DFT_FT(VR, VI, F, T);
```



## 4.13 Cubic Spline Functions

A set of cubic spline functions is built into OSA90 for data interpolation, as summarized in Table 4.6.

TABLE 4.6 CUBIC SPLINE FUNCTIONS

Function	Description
SPLINE( $X,F$ )	computes the cubic spline coefficients (second-order) evaluates an approximate function value using the cubic splines where the coefficients $C$ must be obtained from SPLINE()
SPLINT( $x,C,X,F$ )	
SPLIND( $x,C,X,F$ )	estimates the first-order derivative of w.r.t. $x$ using the cubic splines where the coefficients $C$ must be obtained from SPLINE()
SPLINE2D( $XA,XB,F$ )	computes the bicubic (two-dimensional) spline coefficients evaluates an approximate function value using the bicubic splines (two-dimensional) where the coefficients $C$ must be obtained from SPLINE2D()
SPLINT2D( $xa,xb,C,XA,XB,F$ )	

### Spline Interpolation of a Function of One Variable

The cubic spline functions SPLINE() and SPLINT() are useful for interpolating functions of one variable of the form  $f(x)$ .

#### Syntax:

```
 $C[n] = \text{SPLINE}(X,F);$ 
```

where  $X$  must be an array of length  $n$  which contains  $n$  distinct values of the variable in ascending order and  $F$  must be an array of length  $n$  which contains the corresponding function values. The result  $C$  is an array of length  $n$  which contains the (second-order) coefficients of the cubic splines.

```
 $f = \text{SPLINT}(x,C,X,F);$ 
```

where  $f$  is the approximate function value computed using cubic spline interpolation at the given variable value of  $x$ . The coefficients  $C$  must be obtained from SPLINE( $X,F$ ).

Example:

```
X_data[11] = [0.00  0.05  0.10  0.15  0.20  0.25
              0.30  0.35  0.40  0.45  0.50];

F_data[11] = [-2.143 -2.166 -1.867  0.09704
              1.9     2.087  2.161  1.687
              0.099 -1.855 -2.143];

Coef[11] = SPLINE(X_data, F_data);

x: 0.325;

f_spline = SPLINT(x, Coef, X_data, F_data);
```

The interpolated value of `f_spline` at `x = 0.325` is 2.045.



The variable values (such as those contained in the array `X_data` in the example) must be sorted into ascending order.



See `demo41` for an example of the application of cubic splines.

## Estimating First-Order Derivatives Using Cubic Splines

The function `SPLIND()` evaluates the first-order derivative of  $f(x)$  w.r.t.  $x$  using the cubic spline model for  $f(x)$ .

### Syntax

```
 $d = \text{SPLIND}(x, C, X, F);$ 
```

where  $d$  is the first-order derivative computed using cubic spline interpolation at the given variable value of  $x$ . The coefficients  $C$  must be obtained from `SPLINE(X,F)`.

Example:

```
x: 0.325;

d_spline = SPLIND(x, Coef, X_data, F_data);
```

where `Coef`, `X_data` and `F_data` are the same as in the preceding example.

The first-order derivative of `F_data` w.r.t. `X_data` as estimated by `SPLIND()` is -8.96291 at `x = 0.325`.

## Bicubic Spline Interpolation

The functions `SPLINE2D()` and `SPLINT2D()` are provided for two-dimensional cubic spline interpolation, i.e., for interpolating functions of two variables of the form  $f(xa,xb)$ .

### Syntax

```
C[m,n] = SPLINE2D(XA,XB,F);
```

where  $XA$  must be an array of length  $m$  which contains  $m$  distinct values of the first variable,  $XB$  must be an array of length  $n$  which contains  $n$  distinct values of the second variable, and  $F$  must be an  $m \times n$  matrix which contains the corresponding function values. The values in  $XA$  and  $XB$  must be in ascending order. The result  $C$  is an  $m \times n$  matrix which contains the bicubic spline coefficients.

```
f = SPLINT2D(xa,xb,C,XA,XB,F);
```

where  $f$  is the approximate function value computed using bicubic spline interpolation at the given variable values  $(xa,xb)$ . The coefficient matrix  $C$  must be obtained from `SPLINE2D(XA,XB,F)`.

### Example:

```
VG_data[5] = [-2 -1.5 -1 -0.5 0];
VD_data[9] = [ 0  0.5  1  1.5  2  3  4  5  6];
```

```
ID_data[5,9] = [
    0.0 0.759 0.6787 0.5717 0.5396 0.7016 1.135 1.809 2.697
    0.0 2.697 3.521 4.027 4.552 5.744 7.12 8.666 10.37
    0.0 12.72 15.96 16.76 17.36 18.62 20.01 21.53 23.17
    0.0 24.86 33.64 35.03 35.5 36.35 37.28 38.29 39.38
    0.0 35.42 53.6 56.97 57.33 57.5 57.67 57.87 58.09
];
```

```
Coef[5,9] = SPLINE2D(VG_data, VD_data, ID_data);
```

```
VG: -0.7;
VD: 2.33;
```

```
ID_spline = SPLINT2D(VG, VD, Coef, VG_data, VD_data, ID_data);
```

The interpolated value of `ID_spline` at `VG = -0.77` and `VD = 2.33` is 28.08.



The variable values (such as those contained in the array `VG_data` and `VD_data` in the example) must be sorted into ascending order.



See `demo42` and `demo43` for examples of using the 2D bicubic spline functions.

## 4.14 Piece-Wise Linear Interpolation

A piece-wise linear interpolation function is built into OSA90 for interpolating functions of one variable of the form  $f(x)$ .

### Syntax:

```
y = PWL(x, X, Y);
```

where  $X$  must be an array of length  $n$  which contains  $n$  distinct values of the variable in ascending order and  $Y$  must be an array of length  $n$  which contains the corresponding function values. Using piece-wise linear interpolation,  $y$  is computed as the function value at the given variable value of  $x$ .

The PWL function works as follows.

If  $x \leq X[1]$ , then  $y = Y[1]$ .

If  $x \geq X[n]$ , then  $y = Y[n]$ .

Otherwise, find the indices  $i$  and  $j$  such that  $X[i] < x \leq X[j]$ , then

$$y = Y[i] + (x - X[i])(Y[j] - Y[i]) / (X[j] - X[i])$$

Example:

```
X_data[5] = [ 1.0  2.0  4.0  6.0  7.0 ];
```

```
Y_data[5] = [-2.0  0.0  5.5  1.6  0.3 ];
```

```
x: 3.0;
```

```
y = PWL(x, X_data, Y_data);
```

The interpolated value of  $y$  at  $x = 3.0$  is 2.75.

# 5

## Datapipe

<b>5.1</b>	<b>Overview</b> . . . . .	<b>5-1</b>
<b>5.2</b>	<b>Datapipe Server</b> . . . . .	<b>5-3</b>
<b>5.3</b>	<b>Datapipe Protocols</b> . . . . .	<b>5-6</b>
<b>5.4</b>	<b>SIM Protocol</b> . . . . .	<b>5-9</b>
<b>5.5</b>	<b>COM and COMD Protocols</b> . . . . .	<b>5-15</b>
<b>5.6</b>	<b>FUN Protocol</b> . . . . .	<b>5-24</b>
<b>5.7</b>	<b>FDF Protocol</b> . . . . .	<b>5-25</b>
<b>5.8</b>	<b>LINEAR Protocol</b> . . . . .	<b>5-29</b>
<b>5.9</b>	<b>Application Notes</b> . . . . .	<b>5-32</b>



## 5

## Datapipe

## 5.1 Overview

Traditional CAD programs are closed systems in the sense that all the features are built-in for a preconceived application. They offer little flexibility for customization or user-designed extensions. Modern frameworks promise full-scale integration of independent programs, but they require complicated rules, dedicated training, extensive reprogramming of existing codes, and excessive hardware support.

OSA90's Datapipe utilizes UNIX's interprocess pipe communication facility to establish high speed data connections between OSA90 and one or more external programs. The input parameters to the external programs can be defined and preprocessed in OSA90. The outputs from the external programs can be postprocessed, displayed and optimized. You can functionally integrate multiple external programs by making the outputs of one the inputs of another.

In the input file, Datapipe can be defined in the Expression and Model blocks. The Datapipe schematic is illustrated in Fig. 5.1.

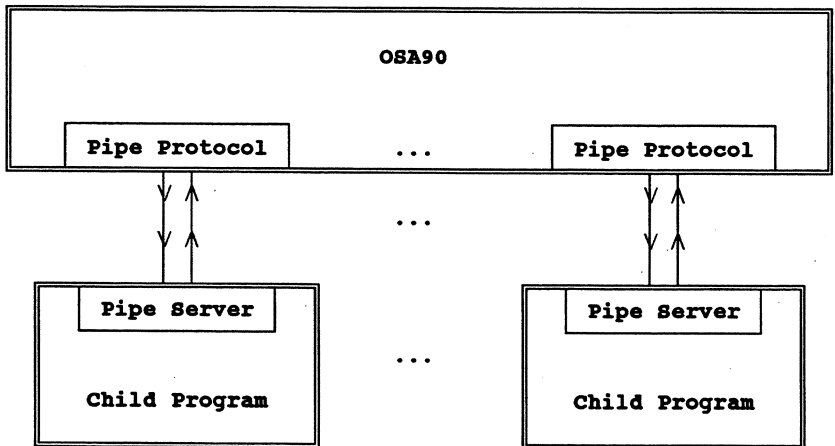


Fig. 5.1 Datapipe schematic

### Child Program

Child program refers to an external program which communicates with OSA90 via a Datapipe connection. A child program may also be referred to simply as a "child".

### Pipe Server

Pipe server is a small set of functions to be included in all child programs for reading data from and writing data to Datapipe. The pipe server is supplied by OSA.

### Child Process

When Datapipe is involved during simulation or optimization, OSA90 will fork (create) a new process and request the operating system to execute the child program in the new process. At the same time, a pair of interprocess communication pipes will be established between OSA90 and the child process.

### Dialogues

Once the Datapipe connection is established, OSA90 and the child program can begin to exchange data. OSA90 computes the inputs required by the child program according to their definitions (variables, labels, etc.) and writes the data to Datapipe. The child program reads the inputs from Datapipe, carries out the calculations, and sends back the outputs to OSA90.

Each exchange of data is called a dialogue. Many dialogues may be involved in one simulation or optimization session, in which many different sets of inputs and outputs are exchanged between OSA90 and the child program.

### Pipe Protocol

Datapipe protocols are a set of communication standards which defines the sequence and meaning of the data fields to be exchanged between OSA90 and the child program. The protocols are designed to accommodate a variety of data formats for different applications.



## 5.2 Datapipe Server

Datapipe server is a small set of functions to be included in all child programs for reading data from and writing data to Datapipe. These functions are written in "C" and based on standard UNIX facilities. The pipe server source code is supplied by OSA which contains about 150 lines of text (the source code is also listed in Appendix A).

**TABLE 5.1 DATAPIPE SERVER FUNCTIONS**

Name	Description
pipe_initialize2()	synchronizes dialogues
pipe_initialize3()	synchronizes dialogues with termination notification
pipe_read2()	reads data from Datapipe
pipe_write2()	writes data to Datapipe

### Pipe Server Structure in Child Program

```

main ()
{
    for (;;)
    {
        pipe_initialize2();

        pipe_read2( ... );

        ... /* calculating the outputs from the inputs */

        pipe_write2( ... );
    }
}

```

### Synchronization

Each iteration within the loop for (;;) { } represents one dialogue between the child program and OSA90. The loop is indefinite, because the total number of dialogues cannot be anticipated by the child program. The termination of the loop (and the child program) is implicitly controlled by the pipe server function pipe\_initialize2().

The pipe server function

```
int pipe_initialize2()
```

synchronizes the Datapipe communication by waiting for an encoded message from OSA90. The message is then interpreted as either (a) to open a dialogue, or (b) to terminate the Datapipe connection. In case (a), an acknowledgement is returned to OSA90 and a data exchange may then begin. In case (b), the child process is terminated.

### Reading Data

The child program obtains input data from OSA90 via Datapipe by calling the pipe server function `pipe_read2()`:

```
int pipe_read2 ( char *buffer, int size, int count )
```

The pointer *buffer* must point to a memory buffer in the child program where the data will be stored. The integer *size* is the number of bytes per data item, and the integer *count* is the number of items to be read.

`Pipe_read2()` returns 0 if the data is successfully read, or -1 if there is an error.

Functionally, the child program reads input data using `pipe_read2()` in the same way as it would read from the standard input (keyboard) or a data file.

### Writing Data

The child program sends output data back to OSA90 via Datapipe by calling the pipe server function `pipe_write2()`:

```
int pipe_write2 ( char *buffer, int size, int count )
```

The pointer *buffer* must point to a memory buffer which contains the data. The integer *size* is the number of bytes per data item, and the integer *count* is the number of items to be written.

`Pipe_write2()` returns 0 if the data is successfully written, or -1 if there is an error.

Functionally, the child program writes output data using `pipe_write2()` in the same ways as it would write to the standard output or a data file.

The calculations of the outputs from the inputs can be coded in any programming languages and organized in any fashion as desired.

To ensure that data is exchanged in the proper order and can be correctly interpreted, the child program must comply with a set of protocols.

## Pipe Server With Termination Notification

```

main ()
{
    for (;;)
    {
        if (pipe_initialize3()) {
            ... /* actions before the child program terminates */

            exit(0);
        }

        pipe_read2( ... );

        ... /* calculating the outputs from the inputs */

        pipe_write2( ... );
    }
}

```

### The pipe server function

```
int pipe_initialize3()
```

is identical to `pipe_initialize2()` except that it notifies the child program of the termination of Datapipe communication by OSA90.

Both `pipe_initialize2()` and `pipe_initialize3()` wait for an encoded message from OSA90. The message is interpreted as either (a) to open a dialogue, or (b) to terminate the Datapipe connection.

In case (a), `pipe_initialize2()` or `pipe_initialize3()` sends an acknowledgement to OSA90 and return to the main program of the child to begin a data exchange. The return value from `pipe_initialize2()` or `pipe_initialize3()` to the main program of the child is 0 in this case.

In case (b), i.e., upon receiving the termination code from OSA90, `pipe_initialize2()` invokes the C function `exit()` to terminate the child program. `pipe_initialize3()`, on the other hand, does not terminate within itself. Rather, `pipe_initialize3()` returns -1 to the main program of the child as a notification of termination. This allows the child program to perform any tasks necessary (updating and closing data files, for example) before the termination.



If you choose to use `pipe_initialize3()`, you must make sure that when `pipe_initialize3()` returns -1 the child program will invoke `exit()` after the necessary clean-up actions.

## 5.3 Datapipe Protocols

Datapipe can be defined in the Expression and Model blocks of the input file.

### Syntax:

```
Datapipe:  protocol      FILE="filename"
           N_INPUT=n     INPUT=(x1, ..., xn)
           N_OUTPUT=m    OUTPUT=(y1, ..., ym)
           TIMEOUT=k;
```

where *protocol* is a keyword identifying the protocol, *filename* is the name of the child program, *n* is the number of inputs to the child, *m* is the number of outputs from the child and *k* is the number of seconds before "time-out".

TABLE 5.2 DATAPIPE PROTOCOLS

Keyword	Description
SIM	simple protocol with numerical inputs and outputs
COM	comprehensive protocol allowing for string inputs/outputs
FUN	protocol for optimization error functions
FDf	protocol for optimization error functions and derivatives
LINEAR	protocol for importing linear subcircuit S/Y/Z data

### Example:

```
Datapipe:  SIM FILE="my_prog"
           N_INPUT=3 INPUT=(3.5, FREQ, ?10cm?)
           N_OUTPUT=2 OUTPUT=(Gain, Efficiency);
```

which defines a Datapipe to communicate with an external program named "my\_prog" using the SIM protocol. The child program expects three input parameters, which are specified in OSA90 by a constant 3.5, the label FREQ and an optimization variable ?10cm?. The child program returns two outputs which will be stored as labels Gain and Efficiency.



There are slight syntax variations between the different protocols. This section focuses on aspects that are common among the protocols. Protocol-dependent details are addressed in the subsequent sections.

## Child Program File Name

The child program as named in `FILE=filename` must be an executable file. If necessary, *filename* should include the full path. For instance, if the child program is named `my_prog` which does not reside in the working directory but, say, in the directory `/user/john`, then *filename* should be specified as `"/user/john/my_prog"`.

The user of OSA90 must have the (file protection) privilege to execute the child program.

## Inputs to the Child

The inputs to the child program  $x_1, \dots, x_n$  can be specified by constants, optimization variables and labels. Labels used as inputs must have been defined in the input file prior to the Datapipe statement.

## Outputs from the Child

The outputs from the child program are represented by labels which can then be displayed and optimized, and used in expressions and models.

The output label names  $y_1, \dots, y_m$  must not duplicate any label names which already exist prior to the Datapipe statement.

## Array Inputs and Outputs

Arrays can also be used as inputs to the child program and to represent outputs from the child program. For example:

```
X[3] = [3.5  FREQ  ?10cm?];

Datapipe:  SIM  FILE="my_prog"
           N_INPUT=3  INPUT=(X)
           N_OUTPUT=2  OUTPUT=(Y[2]);
```



Notice that each array counts as not just one input or output, but as many inputs or outputs as the size of the array. In the above example array X counts as three inputs and array Y represents two outputs.

Whether a numerical input is specified by a constant, a variable or a label is inconsequential to the child program, because its evaluation is handled by OSA90 and only its value is passed to the child program.

Another example:

```
XA[3] = [?-7? ?10cm? ?3PS?];
XB[2,2] = [1.0 2.0
           3.0 4.0];
```

```
Datapipe: SIM FILE="my_prog"
           N_INPUT=6 INPUT=(FREQ, XA[2], XB)
           N_OUTPUT=10 OUTPUT=(Y[2,5]);
```

where scalar label FREQ and array element XA[2] each counts as one input, matrix XB counts as 4 inputs, and matrix Y stores 10 output values from the child program.



Elements of matrices are passed row-wise. For instance, the matrix XB in the above example is passed as XB[1,1], XB[1,2], XB[2,1] and XB[2,2].

## Time Out

Mistakes in programming could cause the Datapipe communication to be out of synchronization. For example, a discrepancy in the number of outputs between the child program and OSA90 could cause OSA90 to wait indefinitely for data that the child program never sent (a *deadlock*).

The protocols are designed to prevent such problems. If the child program strictly complies with the protocol, the communication should be without problems.

However, to avoid a deadlock in the event of unintentional breach of protocol, a "time out" feature allows OSA90 to abort the Datapipe connection and force the termination of the child process after a specified amount of time has elapsed without receiving any data.

The amount of time OSA90 should wait for data is specified by TIMEOUT=*k*, where *k* is an integer representing the number of CPU seconds. The default is *k* = 0, which means the time-out feature is disabled. You can specify a different time-out value for each Datapipe definition.



When the time-out feature is activated, OSA90 has to continually poll (check) the pipe for data availability. This could consume a significant amount of CPU time. It is recommended that you activate the time-out feature only when testing Datapipe connections to new child programs. After the connection is tested and found to be reliable, you should disable the feature (by removing the TIMEOUT keyword or setting *k* = 0) to avoid wasting CPU time on a safeguard that is no longer needed.

## 5.4 SIM Protocol

The SIM protocol is simple and straightforward. It assumes that all the data to be exchanged between OSA90 and the child program are floating-point numbers.

### Syntax

```
Datapipe: SIM FILE="filename"
          N_INPUT=n   INPUT=(x1, ..., xn)
          N_OUTPUT=m  OUTPUT=(y1, ..., ym)
          TIMEOUT=k;
```

where *filename* is the name of the child program, *n* is the number of inputs to the child, *m* is the number of outputs from the child and *k* is the number of seconds for time-out.

Section 5.3 gives a general description of the keywords.

### Example:

```
Datapipe: SIM FILE="sim_prog"
          N_INPUT=3   INPUT=(3.5, FREQ, ?10cm?)
          N_OUTPUT=2  OUTPUT=(Gain, Efficiency);
```

which defines a Datapipe to communicate with an external program named "sim\_prog" using the SIM protocol. The child program *sim\_prog* expects three input parameters, which are specified in OSA90 by a constant 3.5, the label FREQ and an optimization variable ?10cm?. The child program returns two outputs which will be stored as labels Gain and Efficiency.

### Input Data Format

As described in the overview, the communication between OSA90 and the child takes place in the form of dialogues. OSA90 opens a dialogue by sending the input data to the child. The data sent by OSA90 to the child using the SIM protocol consists of three fields, as shown in Fig. 5.2.

<i>n</i>	integer, number of inputs
<i>m</i>	integer, number of outputs
<i>x</i>	<i>n</i> floats, input values

Fig. 5.2 SIM protocol input data format

The first two fields are the numbers of inputs and outputs, respectively, exactly as you have specified in the corresponding Datapipe statement in the input file.

The child program should first read these two fields. The child may be programmed for a fixed number of inputs (outputs), or perhaps it can accept a flexible number of inputs (outputs) within a certain range. In either case, the values of  $n$  and  $m$  should be validated. If they are found to be incorrect, the child should send an error signal back to OSA90 and then OSA90 will alert the user to the problem.

The third field contains the actual inputs:  $n$  float-point values.

Example of reading the inputs in the child program:

```
float x[16];
int n, m, error = 0;

for (;;) {
    pipe_initialize2();                /* synchronization */

    pipe_read2(&n, sizeof(int), 1);   /* number of inputs */
    pipe_read2(&m, sizeof(int), 1);   /* number of outputs */
    pipe_read2(x, sizeof(float), n);  /* input values */

    if (n != 16) error = -1;          /* invalid number of inputs */
    else { ... }
```

where the number of inputs is expected to be exactly 16.



All the input data must be read by the child even if some of the fields are found to be in error. Data left unread may block subsequent Datapipe communications.

Another example:

```
float *x;
int n, m, error = 0;

for (;;) {
    pipe_initialize2();                /* synchronization */

    pipe_read2(&n, sizeof(int), 1);   /* number of inputs */
    pipe_read2(&m, sizeof(int), 1);   /* number of outputs */

    x = (float *) malloc(n * sizeof(float));
    pipe_read2(x, sizeof(float), n);  /* input values */

    if (n < 2 || n > 16 || m < 1 || m > 3) error = -1;
    else { ... }
```

In this case, any number of inputs between 2 and 16 is acceptable, and the number of outputs can be between 1 and 3. The child also allocates memory for data storage according to the actual number of inputs.



## Output Data Format

After receiving the input data, the child will proceed to compute the outputs and then send the results back to OSA90 via Datapipe.

The information sent by the child depends on whether or not an error condition has occurred, either due to incorrect inputs or during the child's calculations. First we will consider the "normal" (i.e., error-free) situation.

The output data format for the SIM protocol is shown in Fig. 5.3.

0	integer, error flag, 0
y	m floats, output values

Fig. 5.3 SIM protocol output format when error free

The first field is an integer error flag, which should be set to zero when there is no error. The second field comprises  $m$  float output values computed by the child, where  $m$  must be identical to the number of outputs specified by OSA90 to the child.

Example:

```
float *x, sum;
int n, m, i, error = 0;

for (;;) {
    pipe_initialize2();                /* synchronization */

    pipe_read2(&n, sizeof(int), 1);    /* number of inputs */
    pipe_read2(&m, sizeof(int), 1);    /* number of outputs */

    x = (float *) malloc(n * sizeof(float));
    pipe_read2(x, sizeof(float), n);   /* input values */

    if (n < 1 || m != 1) error = -1;   /* invalid n or m */
    else {
        for (sum = 0.0, i = 0; i < n; i++) sum += x[i];

        pipe_write2(&error, sizeof(int), 1); /* error is 0 */
        pipe_write2(&sum, sizeof(float), m); /* m is 1 */

        free(x);
    }
    ...
}
```

## Error Flag

When an error is detected within the child program, the Datapipe protocols provide two methods of reporting the error to OSA90: with an error message, or without.

Without an error message, the child program simply sends an error flag with the value `-1` or `1`, using the format shown in Fig. 5.4.

<code>error</code> integer, error flag, -1 or 1
---

Fig. 5.4 Output format: error flag without message

Example:

```

pipe_read2(&n, sizeof(int), 1); /* number of inputs */
pipe_read2(&m, sizeof(int), 1); /* number of outputs */
pipe_read2(x, sizeof(float), n); /* input values */

if (n < 2 || n > 16 || m < 1 || m > 3) {
    error = -1; /* error: invalid n or m */
    pipe_write2(&error, sizeof(int), 1);
}
else { ... }
```

After sending the error flag, the child must stop sending outputs and wait for OSA90 to decide whether to terminate the connection or to open the next dialogue (if the error can be rectified).

## Error Message

The format for sending an error message is shown in Fig. 5.5.

<code>error</code> integer, error message length
<code>message</code> char string, NULL terminated

Fig. 5.5 Output format for error messages

where *message* is a character string terminated by a NULL byte, and *error* is the length of *message* including the NULL byte. The child program must include the NULL byte in the message string. For instance, to send the message "Hello" (5 bytes) plus the NULL byte, the child program must set *error* to 6 and send it first.

## Example:

```

float *x, sum;
char *message;
int n, m, i, error = 0;

for (;;) {
    pipe_initialize2();                /* synchronization */

    pipe_read2(&n, sizeof(int), 1);   /* number of inputs */
    pipe_read2(&m, sizeof(int), 1);   /* number of outputs */

    x = (float *) malloc(n * sizeof(float));
    pipe_read2(x, sizeof(float), n); /* input values */

    if (n < 1) {
        message = "N_INPUT must be a positive integer";
        error = strlen(message) + 1; /* include the NULL byte */
    }
    else if (m != 1) {
        message = "N_OUTPUT must be 1: sum of the inputs";
        error = strlen(message) + 1; /* include the NULL byte */
    }
    else for (sum = 0.0, i = 0; i < n; i++) sum += x[i];

    free(x);

    pipe_write2(&error, sizeof(int), 1); /* error flag */

    if (error) pipe_write2(message, 1, error);
    else pipe_write2(&sum, sizeof(float), m);
}

```

Upon receiving a nonzero error flag from the child, OSA90 will alert the user to the error condition. If *error* > 1, the message supplied by the child program will also be displayed.

In the example, the child program calculates the sum of the inputs and therefore the number of outputs must be 1. Suppose that by mistake you defined the number of outputs to be 2, as

```

Datapipe: SIM FILE="calc_sum"
           N_INPUT=4   INPUT=(X1, X2, X3, X4)
           N_OUTPUT=2  OUTPUT=(SumX, SumY);

```

The child program will return the error message

```

N_OUTPUT must be 1: sum of the inputs

```

which OSA90 will display on the screen in the same way as built-in error messages are displayed.

## Template for Child Programs Using the SIM Protocol

```

main ()
{
    float x[N_MAX], y[M_MAX];
    char *message;
    int n, m, error = 0;

    for (;;) {
        pipe_initialize2();           /* synchronization */

        pipe_read2(&n, sizeof(int), 1); /* number of inputs */
        pipe_read2(&m, sizeof(int), 1); /* number of outputs */
        pipe_read2(x, sizeof(float), n); /* input values */

        if (n < ...) {               /* validate n and m */

            if (!error) {
                /* compute the outputs from the inputs ... */
                in_house_simulator(n, x, m, y, &error);
                ...
            }

            pipe_write2(&error, sizeof(int), 1);

            if (!error) pipe_write2(y, sizeof(float), m);
            else if (error > 1) pipe_write2(message, 1, error);
        }
    }
}

```

`N_MAX` and `M_MAX` represent the maximum numbers of inputs and outputs expected by the child program. If they cannot be easily determined (e.g., the child program can handle an unlimited number of inputs), then the storage memory `x` and `y` will have to be dynamically allocated according to the actual numbers of inputs and outputs.

### Utilizing Termination Notification

If you wish to perform some clean-up tasks before the child terminates, then replace

```
pipe_initialize2();           /* synchronization */
```

by

```

if (pipe_initialize3()) {
    ... /* perform clean-up tasks */
    exit(0);
}

```

## 5.5 COM and COMD Protocols

The COM and COMD protocols are more comprehensive than the SIM protocol: they permit character string inputs and outputs, in addition to numerical data.

### Syntax

```
Datapipe:  COM  FILE="filename"
           N_INPUT=n   INPUT=(x1, ..., xn)
           N_OUTPUT=m  OUTPUT=(y1, ..., ym)
           TIMEOUT=k;
```

where *filename* is the name of the child program, *n* is the number of inputs to the child, *m* is the number of outputs from the child and *k* is the number of seconds for time-out.

COMD is a double precision version of COM. The syntax for COMD is identical to COM with the exception of the protocol keyword.

### Syntax

```
Datapipe:  COMD FILE="filename"
           N_INPUT=n   INPUT=(x1, ..., xn)
           N_OUTPUT=m  OUTPUT=(y1, ..., ym)
           TIMEOUT=k;
```

where *filename* is the name of the child program, *n* is the number of inputs to the child, *m* is the number of outputs from the child and *k* is the number of seconds for time-out.

Throughout this section any reference to "float" should be interpreted as "double" in the case of the COMD protocol.

☞ Section 5.3 gives a general description of the keywords.

The inputs to the child program may include numerical data (constants, optimization variables and labels) as well as character string labels (see Chapter 4). String inputs can be used to pass to the child messages, file names, or even an entire file.

Example:

```
X[3] = [0.1 50 -3.2];
char Data_file[] = "/user/osa90/data/dev_spar.dat";
char SPICE_file[] = "
VCC 5 0 12
VIN 1 0 AC 1 PULSE(0 1 2PS 2PS 2PS 25NS 50NS)
RB 2 5 1310K
RE 3 0 670
RC 4 5 5K
CB 1 2 100UF
Q1 4 2 3 MOD
.MODEL MOD NPN(BF=100 VAF=50 IS=1.E-9 RB=100 CJC 2PF)
.PRINT TRAN V(4) V(1)
.TRAN 1NS 50NS ONS
.END
";

Datapipe: COM FILE="my_prog"
          N_INPUT=5 INPUT=(X, Data_file, SPICE_file)
          N_OUTPUT=2 OUTPUT=(V4t, V1t);
```

where the inputs consist of three floating-point values given by the array X and two string labels. The string label Data\_file specifies a file name, and string label SPICE\_file appears to contain SPICE commands. The child program may utilize these inputs to drive a SPICE compatible simulator and then return the results to OSA90.

If the child program handled double precision input from and output to OSA90 then the Datapipe statement should read as

```
Datapipe: COMD FILE="my_prog"
          N_INPUT=5 INPUT=(X, Data_file, SPICE_file)
          N_OUTPUT=2 OUTPUT=(V4t, V1t);
```

## String Outputs

The outputs from the child program can also include character string labels.

Example:

```
Datapipe:  COM  FILE="gen_file"
           N_INPUT=n   INPUT=(...)
           N_OUTPUT=6  OUTPUT=(Y[4], char A[64], char B[10]);
```

where the outputs include four numerical values to be stored in the array Y, and two string outputs to be labelled as A and B.

The dimension of a Datapipe output string label represents the expected maximum length of the string including the NULL byte. The actual string returned by the child can be shorter than the dimension.

Since the outputs from one child can be used as inputs to another child, string outputs can be useful for passing text information between separate child programs via OSA90.

## Input Data Format

The COM protocol format for sending input data from OSA90 to a child program is shown in Fig. 5.6.

<i>n</i>	integer, number of inputs
<i>m</i>	integer, number of outputs
<i>k</i>	integer, number of packets
<i>p1</i>	the first data packet
...	
<i>pk</i>	the last data packet

Fig. 5.6 COM protocol input data format

The first two fields are the numbers of inputs and outputs, respectively, exactly as you have specified in the corresponding Datapipe statement in the input file.

The inputs are divided into **data packets** (i.e., subsets, groups). The third field specifies the number of data packets. There are two types of packets: string packets and float packets.

## String Packets

1	integer, string packet ID
$L$	integer, length of the string
$s$	char string, NULL terminated

*Fig. 5.7 COM protocol string data packet*

The first field in a packet is a packet type identifier, which is set to 1 for string packets. Instead of the explicit integer "1", you can also use the manifest constant `IPPC_DATA_CHAR`, which is defined in the pipe server header file "ipcv2.h" (listed in Appendix A).

The second field in the packet specifies the length of the string. Each packet contains one string which is terminated with a NULL byte. The string length  $L$  includes the NULL byte. For example, for the string "Hello",  $L$  is set to 6.

The third field in the packet contains the string of  $L$  bytes.

## Float Packets

2	integer, float packet ID
$L$	integer, number of floats in the packet
$x$	$L$ floats, data body

*Fig. 5.8 COM protocol float data packet*

The first field in a packet is a packet type identifier, which is set to 2 for float packets. Instead of the explicit integer "2", you can also use the manifest constant `IPPC_DATA_FLOAT` which is defined in the pipe server header file "ipcv2.h" (listed in Appendix A).

The second field is an integer  $L$  which specifies the number of floating-point values contained in the packet.

The third field in the packet contains  $L$  floating-point values.



## Example:

```
char String1[] = ...;
char String2[] = ...;
X[5] = ...;
```

```
Datapipe: COM FILE=...
          N_INPUT=10 INPUT=(10, 20, String1, String2, X, 30)
          ...;
```

The inputs will be grouped into 4 packets. The first one is a float packet containing two numerical values, namely 10 and 20. The second packet is a string packet passing the string label String1. The third packet is also a string packet containing String2. The fourth packet is a float packet containing 6 numerical values, namely the array X and the constant 30.

The following example illustrates how a child program would read the input data using the COM protocol.

```
float x[N_MAX], *xp;
char s[N_MAX][L_MAX];
int n, m, k, n_floats, n_strings, i, L, type, error = 0;

for (;;) {
    pipe_initialize2();                /* synchronization */

    pipe_read2(&n, sizeof(int), 1);    /* number of inputs */
    pipe_read2(&m, sizeof(int), 1);    /* number of outputs */
    pipe_read2(&k, sizeof(int), 1);    /* number of packets */

    n_floats = n_strings = 0;

    for (i = 0; i < k; i++) {         /* read one packet at a time */
        pipe_read2(&type, sizeof(int), 1);
        pipe_read2(&L, sizeof(int), 1);

        if (type == IPPC_DATA_FLOAT) { /* float packet */
            xp = x + n_floats;
            pipe_read2(xp, sizeof(float), L); /* read L floats */
            n_floats += L;
        }
        else {                          /* string packet */
            pipe_read2(s[n_strings], 1, L); /* read L chars */
            n_strings++;
        }
    }
    ...
}
```



All the input data must be read by the child even if some of the fields are found to be in error. Data left unread may block subsequent Datapipe communications.

## Output Data Format

If an error is encountered in the child program, it should be handled in the same way as described for the SIM protocol (see Section 5.4).

Otherwise, the child program sends the outputs to OSA90 in the format shown in Fig. 5.9.

0	integer, error flag, 0
$p1$	the first output packet
...	
$pk$	the last output packet

Fig. 5.9 COM protocol output format when error free

The first field is an error flag, which should be set to zero when there is no error.

The outputs are organized into data packets in the same way as the inputs (the formats for string and float packets are shown in Figs. 5.7 and 5.8, respectively).



Note that the total number of output data packets is not explicitly specified. OSA90 will read from the child via Datapipe as many packets as necessary to obtain the total number of outputs as specified by  $N\_OUTPUT=m$ .

Example:

```
Datapipe: COM FILE=...
          ...
          N_OUTPUT=4 OUTPUT=(Y[2], char String1[64], Z);
```

The corresponding child program may be as follows.

```
float ..., y[2], z;
char ..., *string;
int ..., type, L, error = 0;

for (;;) {
    pipe_initialize2(); /* synchronization */

    ... /* read inputs and calculate outputs */

    y[0] = ...
    y[1] = ...
    z = ...

    string = "...";
    ...

    if (!error) {
        pipe_write2(&error, sizeof(int), 1); /* error flag 0 */

        type = IPPC_DATA_FLOAT; /* first packet: 2 floats */
        L = 2;
        pipe_write2(&type, sizeof(int), 1);
        pipe_write2(&L, sizeof(int), 1);
        pipe_write2(y, sizeof(float), L);

        type = IPPC_DATA_CHAR; /* second packet: string */
        L = strlen(string) + 1;
        pipe_write2(&type, sizeof(int), 1);
        pipe_write2(&L, sizeof(int), 1);
        pipe_write2(string, 1, L);

        type = IPPC_DATA_FLOAT; /* third packet: 1 float */
        L = 1;
        pipe_write2(&type, sizeof(int), 1);
        pipe_write2(&L, sizeof(int), 1);
        pipe_write2(&z, sizeof(float), L);
    }
}
```

## Template for Child Programs Using the COM Protocol

```

main ()
{
    float x[N_MAX], y[M_MAX];
    char string_inputs[N_MAX][L_MAX], *message;
    int n, m, k, n_floats, n_strings, i, L, type, error = 0;

    for (;;) {
        pipe_initialize2();           /* synchronization */
        pipe_read2(&n, sizeof(int), 1); /* number of inputs */
        pipe_read2(&m, sizeof(int), 1); /* number of outputs */
        pipe_read2(&k, sizeof(int), 1); /* number of packets */

        n_floats = n_strings = 0;
        for (i = 0; i < k; i++) {     /* read one packet at a time */
            pipe_read2(&type, sizeof(int), 1);
            pipe_read2(&L, sizeof(int), 1);

            if (type == IPPC_DATA_FLOAT) { /* float packet */
                xp = x + n_floats;
                pipe_read2(xp, sizeof(float), L); /* read L floats */
                n_floats += L;
            }
            else { /* string packet of L bytes */
                pipe_read2(string_inputs[n_strings], 1, L);
                n_strings++;
            }
        }

        ... /* calculations */

        pipe_write2(&error, sizeof(int), 1);

        if (!error) {
            for ( ... ) { /* write outputs in packets */
                type = ...; /* IPPC_DATA_FLOAT or IPPC_DATA_CHAR */
                L = ...; /* number of floats or length of string */
                pipe_write2(&type, sizeof(int), 1);
                pipe_write2(&L, sizeof(int), 1);

                if (type == IPPC_DATA_FLOAT)
                    pipe_write2(..., sizeof(float), L);
                else pipe_write2(..., 1, L);
            }
        }
        else if (error > 1) pipe_write2(message, 1, error);
    }
}

```

## Utilizing Termination Notification

If you wish to perform some clean-up tasks before the child terminates, then replace

```
pipe_initialize2();          /* synchronization */
```

by

```
if (pipe_initialize3()) {  
    ...          /* perform clean-up tasks */  
    exit(0);  
}
```

## 5.6 FUN Protocol

The FUN protocol is specifically designed for optimization: the outputs from the child program are directly taken as error functions and the outputs are not labelled.

### Syntax

```
Datapipe:  FUN  FILE="filename"
           N_INPUT=n    INPUT=(x1, ..., xn)
           N_OUTPUT=m   NAME=errf_name
           TIMEOUT=k;
```

where *filename* is the name of the child program, *n* is the number of input to the child, *m* is the number of outputs from the child, *errf\_name* is a collective identifier for all the outputs and *k* is the number of seconds for time-out.

☞ Section 5.3 gives a general description of the keywords.

The outputs are not represented by individual labels, instead they are collectively identified by *errf\_name* which can then be referenced in the Specification block for optimization.

### Example:

#### Expression

```
Datapipe:  FUN  FILE="sim_errf"
           N_INPUT=5    INPUT=(X1, X2, ?1.0?, 2.0, FREQ)
           N_OUTPUT=6   NAME=error_set1;
```

```
...
end
```

#### Specification

```
Datapipe:  error_set1;
```

```
...
end
```

This instructs OSA90 to include all the outputs from "sim\_errf" as error functions for optimization (☞ more details in Chapter 11).

The only difference between the FUN protocol and the SIM protocol is the way outputs are handled by OSA90. As far as the child program is concerned, there is no difference between the FUN and SIM protocols. The input and output data formats of the FUN protocol are identical to those of the SIM protocol (Section 5.4).

By not labelling the individual outputs, the FUN protocol consumes less memory and CPU time in processing the Datapipe outputs than the SIM protocol. The disadvantage is that the outputs are not individually accessible for display or postprocessing.

## 5.7 FDF Protocol

The FDF protocol, like the FUN protocol, is specifically designed for optimization: the outputs from the child program are not labelled but directly taken as error functions. Furthermore, the FDF protocol is capable of taking advantage of derivatives supplied by the child program.

### Syntax

```
Datapipe: FDF FILE="filename"
          N_INPUT=n   INPUT=(x1, ..., xn)
          N_OUTPUT=m  NAME=errf_name
          TIMEOUT=k;
```

where *filename* is the name of the child program, *n* is the number of inputs to the child, *m* is the number of outputs from the child, *errf\_name* is a collective identifier for all the outputs and *k* is the number of seconds for time-out.

☞ Section 5.3 gives a general description of the keywords.

The outputs are not represented by individual labels, instead they are collectively identified by *errf\_name* which can then be referenced in the Specification block for optimization.

Example:

```
Expression
  Datapipe: FDF FILE="sim fdf"
            N_INPUT=5   INPUT=(X1, X2, ?1.0?, 2.0, FREQ)
            N_OUTPUT=6  NAME=error_set1;
            ...
end

Specification
  Datapipe: error_set1;
  ...
end
```

This instructs OSA90 to include all the outputs from "sim\_fdf" as error functions for optimization (☞ more details in Chapter 11).

Additionally, the FDF protocol allows the child program to supply first-order derivatives of the outputs with respect to the inputs. These derivatives are then incorporated into the gradients required by the optimizers. If the child program is capable of computing the exact derivatives efficiently (e.g., by adjoint analysis), then the FDF protocol can be used to improve the accuracy and speed of optimization.

## Input Data Format

The FDF protocol format for sending input data from OSA90 to a child program is shown in Fig. 5.10.

$n$	integer, number of inputs
$m$	integer, number of outputs
$x$	$n$ floats, input values
$d$	integer, flag for derivative requirement

Fig. 5.10 FDF protocol input data format

The first two fields are the numbers of inputs and outputs, respectively, exactly as you have specified in the corresponding Datapipe statement in the input file.

The third field contains the actual inputs:  $n$  floating-point values.

The fourth field is an integer flag which indicates whether it is necessary for the child to calculate and return the derivatives. Derivatives may not be needed for some iterations during optimization. Some optimizers (e.g., the random optimizer) may not need derivatives at all. The flag is set to 1 when the derivatives are needed, and set to 0 when they are not necessary.

## Output Data Format: without Derivatives

If an error is encountered in the child program, it should be handled in the same way as described for the SIM protocol (Section 5.4).

Otherwise, if derivatives are not required as indicated by the flag in the inputs, the output data format for the FDF protocol is shown in Fig. 5.11.

$0$	integer, error flag, 0
$f$	$m$ floats, output values

Fig. 5.11 FDF protocol output format without derivatives

The first field is an integer error flag, which should be set to zero when there is no error. The second field contains the  $m$  float-point output values.



## Output Data Format: with Derivatives

When derivatives are required as indicated by the flag in the inputs and the child program did not encounter any error, the output data format shown in Fig. 5.12 should be used for the FDF protocol.

0	integer, error flag, 0
<i>f</i>	<i>m</i> floats, output values
<i>df</i>	$n \times m$ floats, derivatives

Fig. 5.12 FDF protocol output format with derivatives

The total number of first-order derivatives is  $n \times m$ , where  $n$  is the number of inputs and  $m$  is the number of outputs. The derivatives must be sent row-wise, in the following order:

$$\begin{aligned} &\partial f_1/\partial x_1, \partial f_2/\partial x_1, \dots, \partial f_m/\partial x_1, \\ &\partial f_1/\partial x_2, \partial f_2/\partial x_2, \dots, \partial f_m/\partial x_2, \\ &\dots \\ &\partial f_1/\partial x_n, \partial f_2/\partial x_n, \dots, \partial f_m/\partial x_n \end{aligned}$$

where  $f_j$  represents the  $j$ th output and  $x_i$  represents the  $i$ th input.

## Template for Child Programs Using the FDF Protocol

```

main ()
{
    float x[N_MAX], y[M_MAX], df[N_MAX * M_MAX];
    char *message;
    int n, m, need_df, error = 0;

    for (;;) {
        pipe_initialize2();                /* synchronization */

        pipe_read2(&n, sizeof(int), 1);    /* number of inputs */
        pipe_read2(&m, sizeof(int), 1);    /* number of outputs */
        pipe_read2(x, sizeof(float), n);   /* input values */
        pipe_read2(&need_df, sizeof(int), 1); /* flag */

        ... /* calculations */

        if (need_df) { df = ... } /* compute the derivatives */

        pipe_write2(&error, sizeof(int), 1);

        if (!error) {
            pipe_write2(y, sizeof(float), m);

            if (need_df) pipe_write2(df, sizeof(float), (n * m));
        }
        else if (error > 1) pipe_write2(message, 1, error);
    }
}

```

## Utilizing Termination Notification

If you wish to perform some clean-up tasks before the child terminates, then replace

```
pipe_initialize2();                /* synchronization */
```

by

```

if (pipe_initialize3()) {
    ... /* perform clean-up tasks */
    exit(0);
}

```

## 5.8 LINEAR Protocol

The LINEAR protocol is designed for importing into OSA90 linear subcircuits described by  $S$  (scattering),  $Y$  (admittance) or  $Z$  (impedance) matrices.

### Syntax

```
Datapipe: LINEAR FILE="filename"
          N_INPUT= $n$  INPUT=( $x_1, \dots, x_n$ )
          N_PORT= $m$  NAME=subcircuit_name FORMAT=syz
          TIMEOUT= $k$ ;
```

where *filename* is the name of the child program,  $n$  is the number of inputs to the child,  $m$  is the number of ports of the imported linear subcircuit,  $1 \leq m \leq 31$ , *subcircuit\_name* is a string label to identify the imported subcircuit, *syz* can be specified as  $S$ ,  $Y$  or  $Z$  (the default is  $Y$ ), and  $k$  is the number of seconds for time-out.

➤ Section 5.3 gives a general description of the keywords.

The imported linear subcircuit must be an  $m$ -port with a common reference node (i.e., grounded to the reference node), and described by an  $m \times m$  definite  $S$ ,  $Y$  or  $Z$  matrix. Imported subcircuits are defined and referenced in the Model block of the input file.

Example:

```
Model
  Datapipe: LINEAR FILE="my_prog"
            N_INPUT=5 INPUT=(...)
            N_PORT=4 NAME=SubCircuit_1 FORMAT=Y;
  ...
  DATAPORT 5 2 7 4 DATA = SubCircuit_1;
  ...
end
```

A linear subcircuit is imported from the child program "my\_prog", represented by a 4-port  $Y$  matrix and identified as SubCircuit\_1. This subcircuit is later referenced through a DATAPORT element connected to the nodes 5, 2, 7 and 4.

➤ Chapters 6 and 8 provide further details on imported linear subcircuits.

## Frequency Dependence

Typically, linear subcircuits are frequency-dependent. You can include the predefined label FREQ as one of the inputs to the child so that the  $S$ ,  $Y$  or  $Z$  data will be computed at the desired frequency.

## Input Data Format

The input data format for the LINEAR protocol is shown in Fig. 5.13.

$n$	integer, number of inputs
$k$	integer, number of outputs: $2 \times m \times m$
$x$	$n$ floats, input values

Fig. 5.13 LINEAR protocol input data format

The first field is the number of inputs as specified in the corresponding Datapipe statement.

The second field is the number of outputs, which is  $2 \times m \times m$ , because the  $m \times m$  complex  $S$ ,  $Y$  or  $Z$  matrix is sent as  $2 \times m \times m$  floating-point values.

The third field contains the actual inputs:  $n$  floating-point values.

## Output Data Format

If an error is encountered in the child program, it should be handled in the same way as described for the SIM protocol (see Section 5.4).

Otherwise, the  $S/Y/Z$  data is sent back to OSA90 using the format shown in Fig. 5.14.

$0$	integer, error flag, 0
$v$	$2 \times m \times m$ floats, $S/Y/Z$ data

Fig. 5.14 LINEAR protocol output format when error free

The  $m \times m$  complex  $S$ ,  $Y$  or  $Z$  matrix is sent as  $2 \times m \times m$  floating-point values. For example, a  $Y$  matrix is sent row-wise, in the following order:

```

RY11, IY11,  RY12, IY12,  ...,  RY1m, IY1m,
RY21, IY21,  ...,
...
RYm1, IYm1,  RYm2, IYm2,  ...,  RYmm, IYmm

```

where  $RY_{ij}$  and  $IY_{ij}$  represent the real and imaginary parts of  $Y(i,j)$ , respectively.

## Template for Child Programs Using the LINEAR Protocol

```

main ()
{
    float x[N_MAX], y[2 * M_MAX * M_MAX];
    char *message;
    int n, m, k, error = 0;

    for (;;) {
        pipe_initialize2();                /* synchronization */

        pipe_read2(&n, sizeof(int), 1);    /* number of inputs */
        pipe_read2(&k, sizeof(int), 1);    /* number of outputs */
        pipe_read2(x, sizeof(float), n);   /* input values */

        m = (int) sqrt(k / 2);            /* number of ports */

        ... /* calculations */

        pipe_write2(&error, sizeof(int), 1);

        if (!error) pipe_write2(y, sizeof(float), k);
        else if (error > 1) pipe_write2(message, 1, error);
    }
}

```

## Utilizing Termination Notification

If you wish to perform some clean-up tasks before the child terminates, then replace

```

pipe_initialize2();                /* synchronization */

```

by

```

if (pipe_initialize3()) {
    ... /* perform clean-up tasks */

    exit(0);
}

```

## 5.9 Application Notes

### Exciting New Dimension of Optimization-Driven EM Simulation

Empipe is a Datapipe interface connecting OSA90 and *em*. *em* is an efficient full-wave electromagnetic field simulator from Sonnet Software, Inc.

Empipe3D is a Datapipe interface connecting OSA90 to Ansoft HFSS and HP HFSS, finite element based electromagnetic field simulators offered by Ansoft Corporation and Hewlett-Packard Co., respectively.

Empipe and Empipe3D allow you to incorporate EM simulation results into circuit-oriented OSA90, opening up an exciting new dimension of optimization-driven EM field simulation.

A comprehensive library of *em*-ready microstrip components has been built into Empipe. The library includes both simple components, such as microstrip line, step, cross and tee, and more sophisticated structures, such as folded double stub, interdigital capacitor and overlay double patch capacitor. These components are fully parameterized.

Furthermore both Empipe and Empipe3D feature OSA's exclusive Geometry Capture for parameterization of arbitrary structures.

An intelligent database management system is built into both Empipe and Empipe3D. By keeping track of the EM simulation history, it ensures that once a component is simulated by the field solver, the simulation will never have to be duplicated at the same point(s). This computation saving mechanism is especially valuable in optimization and statistical analysis.

### Using OSA90 as a Datapipe Child

OSA90 itself is Datapipe ready, i.e., OSA90 can invoke another copy of OSA90 as a Datapipe child. You can exploit this feature to create a virtual simulation hierarchy of unlimited depth and create capabilities that are not built-in.

For example, you can invoke optimization within a simulation loop (optimization performed by the child OSA90 within a simulation loop in the parent OSA90).

For details, see Chapter 14.

## Multiple References to the Same Child

A child program may be referenced in more than one Datapipe definition in the same input file, each time with a different set of inputs. For example,

```
Datapipe: SIM FILE="calc_sum"
          N_INPUT=4 INPUT=(40, 50, 60, 70)
          N_OUTPUT=1 OUTPUT=(Sum1);

Datapipe: SIM FILE="calc_sum"
          N_INPUT=5 INPUT=(X1, X2, X3, X4, X5)
          N_OUTPUT=1 OUTPUT=(SumX);
```

where the child program "calc\_sum" calculates the sum of its inputs.

A special keyword "SAME" can be used to simplify the syntax:

```
Datapipe: SIM FILE="calc_sum"
          N_INPUT=4 INPUT=(40, 50, 60, 70)
          N_OUTPUT=1 OUTPUT=(Sum1);

Datapipe: SIM FILE=SAME
          N_INPUT=5 INPUT=(X1, X2, X3, X4, X5)
          N_OUTPUT=1 OUTPUT=(SumX);
```

The keyword SAME in the second Datapipe statement means that it refers to the same child as in the preceding Datapipe statement.

## Cascading Datapipes

You can cascade (chain) a number of Datapipes by using the outputs from one child as the inputs to another:

```
Datapipe: SIM FILE="child_1"
          ...
          OUTPUT=(..., Y1[10], ...);

Y1_squared[10] = Y1 * Y1;

Datapipe: COM FILE="child_2"
          ...
          INPUT=(..., Y1_squared, ...) ...;
```

where the outputs from "child\_1" are processed and then used as inputs to "child\_2".

You can utilize this feature to functionally interconnect separate programs via Datapipe as modules of one integrated system. You can then use the expressions in OSA90 for pre-, post- and inter-processing. The interconnections of such a system can be modified easily on the input file level without any changes to the child programs.

## Initialization of Child Programs

A child program may need to be initialized the first time it is called. One method is for the child to keep a count of the number of times it has been called, i.e., a count of the dialogues:

```

/* in the child program */

int ..., count, ...;

for (count = 0; ; count++) { /* loop for dialogues */
    pipe_initialize2();      /* synchronization */
    ...
    if (!count) {
        ... /* first dialogue, perform initialization */
    }
}

```

Or, you can designate one of the inputs to the child as a flag which will explicitly indicate to the child whether initialization is required.

Another way is to have a separate initialization program:

```

Datapipe: SIM FILE="init_sim"
        ...
        INPUT=(X1, ...) ...
        OUTPUT=(status);

Datapipe: SIM FILE="sim_prog"
        ...
        INPUT=(status, ...) ...;

```

where "init\_sim" performs the initialization required for "sim\_prog".

For OSA90 to invoke `init_sim` and `sim_prog` in the appropriate sequence, there must be functional interdependency between `init_sim` and `sim_prog`. Such interdependency exists because `init_sim` and `sim_prog` are cascaded, i.e., the output from `init_sim`, namely `status`, is one of the inputs to `sim_prog`. Such a linkage through cascaded output-input must be established even if an output from the initialization program is not technically needed.



`init_sim` is invoked on an "as necessary" basis: `init_sim` will be invoked at least once and then whenever one of its inputs, such as `X1`, has changed. If all the inputs to `init_sim` remain constant, then it will be invoked only once (before the first call to `sim_prog`).



## Multi-Purpose Child Programs

A child program can be designed to handle several different tasks. Most likely, such tasks are related in some ways and best handled by one child program. Each task may require a different set of inputs from OSA90 and/or return a different set of outputs.

You can include in the inputs one or more encoded flags to be passed to the child program. The child program would then analyze these flags after receiving the inputs and determine which specific task is being requested.

For example, suppose that you have 10 subroutines each performing a different task, but you do not want to create 10 different child programs. You can design a multi-purpose child program by designating, for example, the first input parameter to be an index indicating which subroutine (task) is needed.

The file syntax may look like this:

```
#define TASK_A    3    ! task index between 1 and 10
#define TASK_B    8

Datapipe: SIM FILE="multisim"
        ...
        INPUT=(TASK_A, X1, ...) ...;

Datapipe: SIM FILE=SAME
        ...
        INPUT=(TASK_B, X2, ...) ...;
```

The child program:

```
int ..., task, error = 0;
....

pipe_read2(&n, sizeof(int), 1);
pipe_read2(&m, sizeof(int), 1);
pipe_read2(x, sizeof(float), n);

task = (int) x[0];

if (task == 1) {
    subroutine1(...);
}
...

else if (task == 10) {
    subroutine10(...);
}
```



## 6

## Circuit Models

<b>6.1 Overview</b> .....	6-1
<b>6.2 Nodes</b> .....	6-3
<b>6.3 Elements</b> .....	6-4
<b>6.4 Ideal Sources</b> .....	6-6
<b>6.5 Ports</b> .....	6-11
<b>6.6 Voltage and Current Labels</b> .....	6-16
<b>6.7 CIRCUIT Statement</b> .....	6-18
<b>6.8 DC Responses</b> .....	6-19
<b>6.9 Small-Signal Responses</b> .....	6-22
<b>6.10 Large-Signal Responses</b> .....	6-26
<b>6.11 Postprocessing Responses</b> .....	6-36
<b>6.12 Linear Subcircuits</b> .....	6-38
<b>6.13 User-Defined Linear Models</b> .....	6-41
<b>6.14 User-Defined Nonlinear Models</b> .....	6-44
<b>6.15 ImportData Block</b> .....	6-48
<b>6.16 LINEAR Datapipe</b> .....	6-56
<b>6.17 Macros and Symbolic Subcircuits</b> .....	6-58
<b>6.18 Oscillator Port</b> .....	6-61



## 6

## Circuit Models

## 6.1 Overview

OSA90 can simulate linear and nonlinear circuits of general n-port topology. The circuit models are described in the Model block of the input file.

**Syntax:****Model**

*labels and expressions;*

**SUBCIRCUIT** ...;

**Datapipe:** ...;

*elements;*

**VSOURCE** ...;

**ISOURCE** ...;

**PORT** ...;

**CIRCUIT** ...;

*response postprocessing*

**End**

**Labels and Expressions** are described in Chapter 4. Functionally, the Model block includes all the features of the Expression block.

**Datapipes** are described in Chapter 5.

**Elements** include linear and nonlinear circuit elements. A circuit model consists of a set of interconnected elements. The model represents a nonlinear circuit if it contains at least one nonlinear element. Otherwise the model represents a linear circuit.

**Subcircuits** are user-defined linear subcircuits. Subcircuits can be parameterized to create custom linear elements that are not available from the built-in element library.

Ideal **voltage sources** and **current sources** can be defined by **VSOURCE** and **ISOURCE** statements, respectively. Non-ideal sources can be included in **external ports** defined by **PORT** statements.

**Circuit Responses** are represented by labels which are automatically created as soon as the circuit definition is completed with a **CIRCUIT** statement. Circuit responses include DC, small-signal AC and large-signal AC responses.

**Response postprocessing** utilizes expressions to create customized (user-defined) responses for display and optimization.

## 6.2 Nodes

Nodes represent the interconnections between the circuit elements, sources and ports defined in the Model block.

Nodes can be designated by positive integers, such as

```
RES 1 2 ...
```

which means that a RES element is connected between the nodes designated as 1 and 2.

Node numbers do not need to be consecutive or in any particular order.

### Nodes Designated by Names

Nodes can also be designated by character strings with the prefix @.

Example:

```
RES @input @ground ...
FETM @gate @drain @source ...
```

Node names are case insensitive unless they are enclosed within quotation marks. For example,

```
@gate, @GATE, @Gate
```

designate the same node, but

```
@"gate", @"GATE"
```

are considered as two separate and distinct node names.

Mixed usage of numerical and string nodes is permitted:

```
RES 1 @output ...
```

### The Ground Node

The node number 0 and the node name @ground are automatically created and they always designate the ground node.

## 6.3 Elements

OSA90 provides a library of linear and nonlinear elements from which you can select components of a circuit model.

### Syntax

```
element_name  n1 n2 ... nk  par1=x1 par2=x2 ... parm=xm;
```

where *element\_name* is a keyword identifying a library element, *ni* is the *i*th connection node, *parj* is a keyword identifying one of the parameters of the element model, and *xj* is a value or a label assigned by the user to the parameter *parj*.

Example:

```
RES  3 5  R=50;
CAP  @input @ground  C=10pF;
```

Catalogues of the library elements are contained in Chapter 7 (nonlinear elements) and Chapter 8 (linear elements), which include the schematics, keywords, number of nodes, parameters and model equations where appropriate.

Each element definition is a statement and must end with a semicolon ";".

### Element Nodes

Each element has a predefined number of nodes. For example, the element RES (resistor) has 2 nodes. Nodes that are not explicitly specified will be grounded (i.e., default to the node 0 or @ground).

However, each element requires a minimum number of nonzero nodes for the element to be meaningfully connected, e.g., the element RES requires at least one nonzero node.

### Element Parameters

Each element has a number of parameters. An element parameter can be assigned a constant value, defined as an optimization variable or specified by a label.

Example:

```
SRLC  1 0  R=15  L=?0.2nH?  C=Label_C1;
```

where R, L and C are keywords representing the parameters for the element SRLC. R is assigned a constant value, L is defined as an optimization variable (see Chapter 4), and C is specified by the label Label\_C1 which must have been defined prior to this reference.

Numerical constants can have physical units, e.g., 10pF and 0.2nH (see Chapter 3).



## Parameters Defined by Expressions

An element parameter can also be defined by an expression (formula), such as

```
RES 1 0 R = (Resistivity * Length / Cross_Section);
```



Such an expression within an element definition must be enclosed in a pair of parentheses to clearly delimit the expression from the rest of the text. Otherwise, the expression may be confused with the other parameter keywords in the same statement.

For example,

```
SRL 1 0 R = Resistivity * Length / Cross_Section L = A + B;
```

will be considered as an error by the input file parser.

Alternatively, you can define the expression as a label in a separate statement:

```
Label_R: Resistivity * Length / Cross_Section;
RES 1 0 R=Label_R;
```

## Parameters Evaluated Externally

Element parameters can also be evaluated by an external program connected to OSA90 via Datapipe (Chapter 5).

Example:

```
Datapipe: SIM FILE="my_prog"
... INPUT=(...)
... OUTPUT=(X1, ...) ...;

RES 1 0 R=X1;
```

where "my\_prog" is the child program and one of its outputs is used to specify one of the element parameters.

## 6.4 Ideal Sources

Independent sources provide excitations for DC and large-signal harmonic balance analyses. DC sources may also be required to provide the appropriate bias for nonlinear active devices.

This section describes ideal sources. Non-ideal sources associated with ports are discussed in Section 6.5.

### Ideal Voltage Sources

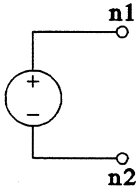


Fig. 6.1 Ideal voltage source.

#### Syntax

```
VSOURCE.name n1 n2 VDC=vdc V=v PHASE=a;
```

*name* is a character string to identify the source. *n1* and *n2* are nodes.

*vdc* specifies the DC voltage.

*v* and *a* specify the magnitude and phase (angle) of the AC voltage, respectively.

Both nodes *n1* and *n2* are required, other entries are optional.

#### Examples:

```
VSOURCE.DC_bias @bias_node @ground VDC=-1.74V;
```

```
VSOURCE 2 0 V=?0.2V? PHASE=15deg VDC=0.5V;
```

The identifier *name* is required only if you are interested in the current response of the source. If given, *name* will be incorporated into the response labels (Sections 6.8 and 6.10).

The parameters VDC, V and PHASE can be specified by constants, optimization variables, labels and expressions. If omitted, the default value is zero.

## Ideal Current Sources

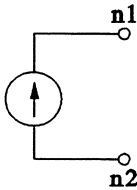


Fig. 6.2 Ideal current source.

### Syntax

```
ISOURCE.name n1 n2 IDC=idc I=i PHASE=a;
```

*name* is a character string to identify the source. *n1* and *n2* are nodes.

*idc* specifies the DC current.

*i* and *a* specify the magnitude and phase (angle) of the AC current, respectively.

Both nodes *n1* and *n2* are required, other entries are optional.

### Examples:

```
ISOURCE 1 0 IDC=1.5mA;
```

```
ISOURCE.input 2 0 I=0.2A PHASE=15deg IDC=0.5A;
```

The identifier *name* is required only if you are interested in the voltage response of the source. If given, *name* will be incorporated into the response labels (Sections 6.8 and 6.10).

The parameters IDC, I and PHASE can be specified by constants, optimization variables, labels and expressions. If omitted, the default value is zero.

## Alternative Syntax for Source Names

The names identifying voltage and current sources can also be defined using the keyword NAME. For example, the following definitions are equivalent

```
ISOURCE.input 1 0 I=0.2A ...;
```

```
ISOURCE 1 0 NAME=input I=0.2A ...;
```

## A Potential Problem with Ideal Current Bias Sources

When an ideal current source is used as a DC bias source for an active device, you may need to connect a large resistor in parallel with ISOURCE. For example, suppose that you use a current source to provide the base-emitter bias for a bipolar transistor. When the collector-emitter voltage is zero, the base current is zero. This can cause a problem for the simulator if the only path for the current of the ideal source is through the base of the transistor. This potential problem can be avoided if an alternate path for the source current is provided by a large resistor connected in parallel with ISOURCE.

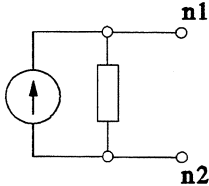


Fig. 6.3 Current source with a resistor in parallel.

A 1MOH - 10MOH resistor is usually suitable for this purpose.

## Source Frequency

AC sources in OSA90 can be of single-tone or two-tone.

In the single-tone case, all the AC sources in the circuit share the same (fundamental) frequency which is represented by the predefined label FREQ.

For AC analyses (small- and large-signal), you assign a value to FREQ or sweep FREQ over a set of values in the Sweep, Specification and MonteCarlo blocks.

Example:

```
Sweep
  FREQ: from 2GHZ to 10GHZ step=1GHZ ... ;
  ...
End
```

whereby the frequency of all the AC sources are swept over the specified range.

In the two-tone case, the tone frequencies are represented by FREQ and FREQ2.

☞ See Chapters 9, 11 and 12 for further details.

## Harmonic Excitations

An AC source may contain higher harmonics of the fundamental frequency.

### Syntax

```
VSOURCE.name n1 n2 VDC=vdc V[1]=v1 PHASE[1]=a1
... V[m]=vm PHASE[m]=am;
```

*name* is a character string identifier. *n1* and *n2* are nodes. *vdc* is the DC voltage. *m* is the highest harmonic contained in the source,  $1 \leq m \leq 8$ . *vk* and *ak* specify the magnitude and phase (angle) of the *k*th harmonic component,  $1 \leq k \leq m$ , and the *k*th harmonic frequency is *k* times the fundamental frequency. Both nodes *n1* and *n2* are required, other entries are optional.

### Example:

```
VSOURCE.Triangular_Wave 1 0 V[1]=1 V[3]=-(1/9)
V[5]=-(1/25) V[7]=-(1/49);
```

which represents a triangular waveform excitation truncated to the first 7 harmonics.

### Syntax

```
ISOURCE.name n1 n2 IDC=idc I[1]=i1 PHASE[1]=a1
... I[m]=im PHASE[m]=am;
```

*name* is a character string identifier. *n1* and *n2* are nodes. *idc* is the DC current. *m* is the highest harmonic contained in the source,  $1 \leq m \leq 8$ . *ik* and *ak* specify the magnitude and phase (angle) of the *k*th harmonic component,  $1 \leq k \leq m$ , and the *k*th harmonic frequency is *k* times the fundamental frequency. Both nodes *n1* and *n2* are required, other entries are optional.

### Example:

```
ISOURCE.Triangular_Wave 1 0 I[1]=1 I[3]=-(1/9)
I[5]=-(1/25) I[7]=-(1/49);
```

which represents a triangular waveform excitation truncated to the first 7 harmonics.

## Two-Tone Sources

By "two-tone sources", we mean that the AC sources in a circuit contain two different and harmonically unrelated stimulus frequencies.

### Syntax

```
VSOURCE.name n1 n2 VDC=vdc V=v1 PHASE=a1
                ...      V2=v2 PHASE2=a2;
```

*name* is a character string identifier. *n1* and *n2* are nodes. *vdc* is the DC voltage. *v1* and *a1* specify the magnitude and phase (angle) of the first tone, respectively. *v2* and *a2* specify the magnitude and phase (angle) of the second tone, respectively. Both nodes *n1* and *n2* are required, other entries are optional.

### Example:

```
VSOURCE.two_tone_input 1 0 V=0.2V V2=0.15V;
```

The two tones may also appear in separate sources:

```
VSOURCE.first_tone @input1 0 V=0.2V;
VSOURCE.second_tone @input2 0 V2=0.15V;
```

The tone frequencies are to be specified by **FREQ** and **FREQ2** in the Sweep, Specification and MonteCarlo blocks.

Two-tone current sources can be similarly defined.

### Syntax

```
ISOURCE.name n1 n2 IDC=idc I=i1 PHASE=a1
                ...      I2=i2 PHASE2=a2;
```

*name* is a character string identifier. *n1* and *n2* are nodes. *idc* is the DC current. *i1* and *a1* specify the magnitude and phase (angle) of the first tone, respectively. *i2* and *a2* specify the magnitude and phase (angle) of the second tone, respectively. Both nodes *n1* and *n2* are required, other entries are optional.

### Example:

```
ISOURCE.two_tone_input 1 0 I=0.2A I2=0.15A;
```

## 6.5 Ports

Generally, the overall circuit is defined as an  $n$ -port. Ports are defined individually by separate PORT statements. Up to 64 external ports can be defined.

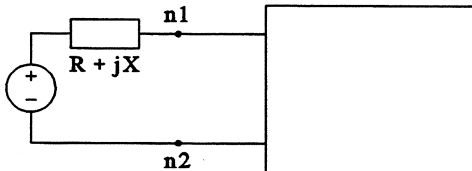


Fig. 6.4 Port.

### Syntax:

```
PORT.name n1 n2 R=r X=x;
```

*name* is a character string identifier. *n1* and *n2* are nodes. *r* and *x* specify the real and imaginary parts of the terminating impedance, respectively. Both nodes *n1* and *n2* are required, other entries are optional.

### Example:

```
PORT 2 0;
```

The identifier *name* is required only if you are interested in calculating the voltage and current responses through DC and/or large-signal simulation.

### Example:

```
PORT.output 2 0;
```

The name "output" will be incorporated into the response labels (Sections 6.8 and 6.10).

Alternatively, you can specify the port name using the keyword NAME:

```
PORT 2 0 NAME=output;
```

## Port Termination

Port terminations defined in the PORT statements are, by default, considered as the reference impedances in the calculation of small-signal  $S$  parameters. In DC and harmonic balance simulations, they are included in the overall circuit definition, as shown in Fig. 6.4.

The parameters R and X can be used to define the port termination.

Example:

```
PORT.output 2 0 R=10;
```

R and X can be specified by constant values, optimization variables, labels and expressions.

If both R and X are omitted, the default termination is  $R = 50\Omega$ . If only R or X is specified, the default value for the omitted parameter is zero.



A port must not be terminated by a short circuit, i.e., R and X cannot both be set to zero at the same time.

## Frequency-Dependent Complex Termination

The port termination can be frequency dependent if R and/or X is defined by a frequency-dependent label or expression.

You can also terminate a port with a one-port subcircuit by replacing the keywords R and X with the keyword Z:

### Syntax:

```
PORT.name n1 n2 Z=subcircuit_name;
```

*name* is a character string identifier. *n1* and *n2* are nodes.  
*subcircuit\_name* identifies a one-port linear subcircuit.

Example:

```
Port.output 1 0 Z=Load;
```

where Load identifies a one-port linear subcircuit defined by any one of the following:

a linear subcircuit defined within OSA90 (☞ Section 6.12),

an imported linear subcircuit defined by  $S/Y/Z$  data supplied in the ImportData block (☞ Section 6.15),

a linear subcircuit imported through LINEAR Datapipe (☞ Section 6.16).



## Voltage Source Associated with a Port

### Syntax:

```
PORT.name n1 n2 VDC=vdc V=v PHASE=a;
```

*name* is a character string identifier. *n1* and *n2* are nodes.

*vdc* specifies the DC voltage. *v* and *a* specify the magnitude and phase (angle) of the AC voltage, respectively.

### Example:

```
PORT.input 2 0 V=0.5V;
```

The source associated with a port is very similar to the ideal sources described in Section 6.4, except that the source associated with a port is non-ideal.

## Current Source Associated with a Port

### Syntax:

```
PORT.name n1 n2 IDC=idc I=i PHASE=a;
```

*name* is a character string identifier. *n1* and *n2* are nodes.

*idc* specifies the DC current. *i* and *a* specify the magnitude and phase (angle) of the AC current, respectively.

### Example:

```
PORT.input 2 0 I=10mA;
```

The current source associated with a port is very similar to the ideal sources described in Section 6.4.

## Source Specified by Available Power

An AC source associated with a port can be specified in terms of the available input power.

### Syntax

```
PORT.name n1 n2 R=r X=x VDC=vdc P=p PHASE=a;
```

*name* is a character string identifier. *n1* and *n2* are nodes.  
*r* and *x* specify the real and imaginary parts of the terminating impedance, respectively.  
*vdc* specifies the DC source voltage.  
*p* specifies the available input power in dBm.  
*a* specifies the phase (angle) of the AC source voltage.

### Example:

```
PORT.input 2 0 P=-5 R=50;
```

The available input power in dBm is related to the magnitude of the voltage by

$$p = 10 * \log_{10}(v^2 / (8 * r)) + 30$$

where *v* represents the magnitude of the AC source voltage.



If PHASE is specified together with the available power, it is considered to be the phase (angle) of the voltage.

## Harmonic Excitations

An AC source associated with a port may contain higher harmonic excitations.

### Syntax

```
PORT.name n1 n2 R=r X=x VDC=vdc V[1]=v1 PHASE[1]=a1
... V[m]=vm PHASE[m]=am;
```

*name* is a character string identifier. *n1* and *n2* are nodes.  
*r* and *x* specify the real and imaginary parts of the terminating impedance, respectively.  
*vdc* specifies the DC source voltage.  
*vk* and *ak* specify the magnitude and phase (angle) of the *k*th harmonic component,  $1 \leq k \leq m$ , *m* is the highest harmonic contained in the source,  $1 \leq m \leq 8$ .

The magnitudes can be specified in terms the available input powers by replacing the keywords *V[k]* with *P[k]*.

Current source containing higher harmonics can be defined by replacing the keywords VDC and *V[k]* with IDC and *I[k]*, respectively.

## Two-Tone Excitations

Ports may also contain two-tone sources, i.e., the sources may contain two different and harmonically unrelated stimulus frequencies, similar to the case of ideal sources.

### Syntax

```
PORT.name n1 n2 R=r X=x VDC=vdc V=v1 PHASE=a1
      ...           V2=v2 PHASE2=a2;
```

*name* is a character string identifier. *n1* and *n2* are nodes.  
*r* and *x* specify the real and imaginary parts of the terminating impedance, respectively.  
*vdc* is the DC source voltage.  
*v1* and *a1* specify the magnitude and phase (angle) of the first tone, respectively.  
*v2* and *a2* specify the magnitude and phase (angle) of the second tone, respectively.

The magnitudes can be specified in terms of the available input powers by replacing the keywords V and V2 with P and P2, respectively.

Two-tone current source can be defined by replacing the keywords VDC, V and V2 with IDC, I and I2, respectively.

## Multiple Ports Defined Using One Statement

You can use the keyword PORTS to define multiple ports with a single statement.

### Syntax

```
PORTS n1 n2 ... nk;
```

where *n1*, *n2*, ..., *nk* are nodes.  
*k* must be an even number and the number of ports defined is  $k / 2$ .  
 Nodes *n1* and *n2* forms the first port, *n3* and *n4* forms the second port, and so on.

Example:

```
PORTS 1 0 2 0;
```

This defines two ports. The first port is between the nodes 1 and 0, and the second port is between the nodes 2 and 0.

PORTS provides a convenient way of defining ports for small-signal simulation. However, you will not be able to assign names to the ports, define port terminations, or define sources associated with the ports. Therefore, PORTS may not be useful for DC or large-signal HB analysis where the voltage and current responses of the ports need to be identified through the names of the ports. Also, when PORTS is used, the default reference impedance of 50 ohm is implied for *S*-parameter calculation.

## 6.6 Voltage and Current Labels

A voltage label represents the voltage between two nodes in the circuit. A current label represents the current through a branch in the circuit.

### Voltage Labels



Fig. 6.5 Voltage label.

#### Syntax

```
VLABEL.name n1 n2 TAU= $\tau$ ;
```

where *name* is a string to identify the voltage label, *n1* and *n2* are nodes, TAU= $\tau$  specifies an optional time delay,  $\tau$  can be a constant, variable or label.

#### Example:

```
VLABEL.V52 5 2;
```

Then,

$$V52 = v5 - v2$$

where *v5* and *v2* represent the voltages at the nodes 5 and 2, respectively.

The label identifier *name* represents the time-domain voltage. It can be used as a DC response label (see Section 6.8). The voltage label name will also be incorporated into the name of an array which represents the large-signal voltage response (see Section 6.10).

### Time Delay

You can specify an optional time delay for a voltage label, for example,

```
VLABEL.V52_delayed 5 2 TAU=3pS;
```

Then,

$$V52\_delayed(t) = v5(t - 3pS) - v2(t - 3pS)$$

where *v5* and *v2* represent the voltage at the nodes 5 and 2, respectively.

You can define a number of voltage labels for the same pair of nodes but with different delays. For example,

```
VLABEL.V52 5 2;
VLABEL.V52_delayed_1 5 2 TAU=3pS;
VLABEL.V52_delayed_2 5 2 TAU=5pS;
```

## Current Labels



Fig. 6.6 Current label.

### Syntax

```
ILABEL.name n1 n2;
```

where *name* is a string to identify the current label. *n1* and *n2* are nodes.



Note that `ILABEL` introduces a short-circuit between nodes *n1* and *n2*. In order to measure the current through a branch, you need to break that branch, create a new node and insert an `ILABEL`.

Example:

```
ILABEL.I_52 5 2;
```

The label `I_52` represents the current through the short-circuit between nodes 5 and 2.

## Voltage and Current Measurements of a Circuit Component

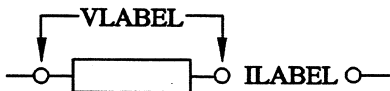


Fig. 6.7 Measuring the voltage and current of a component.

## 6.7 CIRCUIT Statement

To indicate that the circuit definition in the Model block is finished, you must use the CIRCUIT statement.

### Syntax

```
CIRCUIT .name HARM=m MIXER INTMOD;
```

where all the parameters are optional. If given, *name* must be a character string. HARM=*m* specifies the highest harmonic to be included in large-signal analysis. MIXER or INTMOD controls the formulation of spectral frequencies for two-tone analysis.

The keywords HARM, MIXER and INTMOD are relevant only for large-signal simulation, hence they are described in the context of large-signal responses in Section 6.10.

## Response Labels

As soon as the circuit definition is finished, OSA90 automatically generates a set of labels to represent the circuit responses. These are called response labels.

Response labels provide the means by which you can refer to the circuit simulation results for display and optimization. The response labels can also be incorporated into expressions to create user-defined responses.

Response labels are created and become available only after the CIRCUIT statement, because the responses depend on the overall circuit configuration (e.g., the number of *S* parameters depends on the total number of circuit ports).

### The proper position of the CIRCUIT statement

Model

*circuit definition: elements, sources, ports, etc.*

CIRCUIT ...;      *response labels created here*

*user-defined responses: postprocessing expressions*

End

## 6.8 DC Responses

For DC responses to be available, the circuit must contain at least one DC source, i.e., at least one VSOURCE, ISOURCE or PORT which includes a nonzero VDC or IDC.

TABLE 6.1 DC RESPONSE LABELS

Label	Response
<i>vname_DC</i>	DC voltage of a port defined as <i>PORT.name</i> or, of a source defined as <i>ISOURCE.name</i>
<i>iname_DC</i>	DC current of a port defined as <i>PORT.name</i> , or, of a source defined as <i>VSOURCE.name</i>
<i>vname</i>	DC voltage of a voltage label defined as <i>VLABEL.vname</i>
<i>iname</i>	DC current of a current label defined as <i>ILABEL.iname</i>

### DC Voltages and Currents of Ports

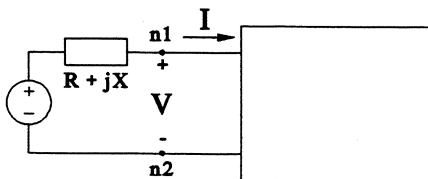


Fig. 6.8 Voltage and current of a port.

Examples:

```
PORT.input_port 3 4 ...;
```

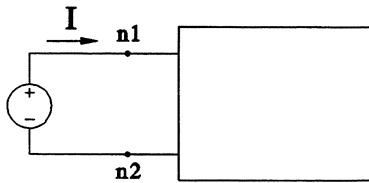
```
PORT.out 5 6 ...;
```

The corresponding DC response labels are

```
Vinput_port_DC, Iinput_port_DC
```

```
Vout_DC, Iout_DC
```

## DC Currents of Voltage Sources



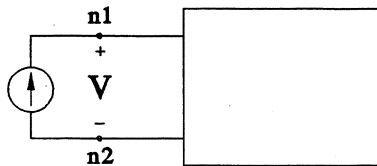
*Fig. 6.9 Current of an ideal voltage source.*

Example:

```
VSOURCE.bias 1 2 ...;
```

The corresponding DC response label is `Ibias_DC`.

## DC Voltages of Current Sources



*Fig. 6.10 Voltage of an ideal current source.*

Example:

```
ISOURCE.bias 1 2 ...;
```

The corresponding DC response label is `Vbias_DC`.



## Voltage and Current Labels

Voltage and current labels can be directly used to represent the DC voltage and current responses, respectively.

Example:

```
VLABEL.V52 5 2;
ILABEL.Current_Probe 11 15;
```

The DC voltage between node 5 and node 2 is represented by the label V52, and the DC current through the short-circuit between node 11 and node 15 is represented by the label Current\_Probe.



### Response Label Names Must Be Unique

The response label names created according to the templates, such as those listed in Table 6.1, must be unique. They cannot duplicate any existing label names defined by the user.

For example, the DC response labels corresponding to the port definition

```
PORT.output 5 6 ...;
```

would be Voutput\_DC and Ioutput\_DC. If either of these names is already taken by an existing label, the file parser will signal an error.

Therefore, when creating labels, avoid using names which may conflict with the templates for response labels, as listed in Tables 6.1 - 6.6.

## 6.9 Small-Signal Responses

### S, Y and Z Matrices

The small-signal circuit responses include the  $n$ -port  $S$ ,  $Y$  and  $Z$  parameters.

The complex  $S$  parameters are available in both the polar form (magnitude and phase) and the rectangular form (real and imaginary parts). The  $Y$  and  $Z$  parameters are available in the rectangular form only.

The  $S/Y/Z$  parameters are represented by the  $n \times n$  matrices listed in Table 6.2.

**TABLE 6.2 S, Y AND Z MATRICES**

Matrix	Description
MS[ $n, n$ ]	magnitudes of the $S$ parameters
PS[ $n, n$ ]	phases of the $S$ parameters in degrees
RS[ $n, n$ ]	real parts of the $S$ parameters
IS[ $n, n$ ]	imaginary parts of the $S$ parameters
RY[ $n, n$ ]	real parts of the $Y$ parameters
IY[ $n, n$ ]	imaginary parts of the $Y$ parameters
RZ[ $n, n$ ]	real parts of the $Z$ parameters
IZ[ $n, n$ ]	imaginary parts of the $Z$ parameters

You can refer to an  $S$ -,  $Y$ - or  $Z$ -parameter matrix as a whole or refer to its individual elements. For example, you can refer to MS as a matrix or use MS[2,1] to refer to the magnitude of  $S_{21}$ .

 See Chapter 4 for further details on matrices and array operations.



The indices of the  $S/Y/Z$  parameters refer to port numbers. Ports are numbered according to the order of their appearances in the Model block.

## Individual S/Y/Z Parameters

For your convenience, the *S/Y/Z* parameters are also made available as the scalar labels listed in Table 6.3.

**TABLE 6.3 S, Y AND Z LABELS**

Label	Description
$MS_{ij}$	equivalent to $MS[i, j]$
$PS_{ij}$	equivalent to $PS[i, j]$
$RS_{ij}$	equivalent to $RS[i, j]$
$IS_{ij}$	equivalent to $IS[i, j]$
$RY_{ij}$	equivalent to $RY[i, j]$
$IY_{ij}$	equivalent to $IY[i, j]$
$RZ_{ij}$	equivalent to $RZ[i, j]$
$IZ_{ij}$	equivalent to $IZ[i, j]$



The scalar labels are limited to  $1 \leq i \leq 9$  and  $1 \leq j \leq 9$ . If the number of ports exceeds 9, then only those *S*-, *Y*- and *Z*-parameters corresponding to the first 9 ports can be referred to in this way. To refer to the other *S/Y/Z* parameters, you will have to use the matrix notation, such as  $MS[11,1]$ .

## Unit of the Phase

Following convention, the phase is given in degrees. This means that they cannot be directly used in trigonometric functions. You can convert the phases to radians using an expression, such as:

$$PS21\_in\_radians = PS21 * PI / 180;$$

where *PI* is a predefined label. You can even convert the whole matrix:

$$PS\_in\_radians[n,n] = PS * PI / 180;$$

where *n* should be set to the number of ports.

## Reference Impedances for S Parameters

By default, the reference impedances for calculating the  $S$  parameters are those of the port terminations defined in each of the PORT statements (see Section 6.5); if the termination is not defined for a port then it is assumed to be 50 ohm by default. You can override these default settings by using the RREF keyword in the Sweep, Specification and MonteCarlo blocks (see Chapters 9, 11 and 12).

## Group Delay Responses

The group delay responses for an  $n$ -port circuit are represented by the matrix and labels listed in Table 6.4.

TABLE 6.4 GROUP DELAY RESPONSES

Name	Description
GD[n,n]	group delay matrix
GD <i>i,j</i>	equivalent to GD[i,j]

The group delay is defined as

$$GD_{ij} = \partial(\angle S_{ij}) / \partial \omega$$

where  $\angle S_{ij}$  is the phase of  $S_{ij}$  in radians and  $\omega$  is the angular frequency.

By default, the unit of frequency in OSA90 is GHZ. Consequently, the default unit of  $\omega$  is radians per nanosecond and the default unit of  $GD_{ij}$  is nanosecond.

You can change the unit of frequency to HZ by specifying Non\_Microwave\_Units in the Control block. Then, the unit of  $GD_{ij}$  will be second.

## One-Sided Group Delay

By default, two-sided perturbations are used for group delay computation. In the Control block, you can specify one-sided perturbations to be used instead:

```
Control
  One_Sided_Group_Delay;
  ...
End
```

See Chapter 3 for further details on the Control block.

## Two-Ports Responses

For two-ports, a number of additional small-signal responses are available.

**TABLE 6.5 SMALL-SIGNAL RESPONSES FOR TWO-PORTS**

<b>Label</b>	<b>Response</b>
<b>SK</b>	stability factor
<b>GMAX</b>	maximum available gain in decibels
<b>INSL</b>	insertion loss in decibels

These two-port responses are defined by assuming that the port which appears first in the Model block is the input port and the other port is the output port.

## 6.10 Large-Signal Responses

The large-signal responses computed by OSA90 are frequency-domain complex spectra of voltages and currents. Large-signal analysis requires that the circuit contain at least one AC source.

The number of spectral components of the large-signal responses corresponds to the number of spectral frequencies included in the harmonic balance simulation.

### Harmonic and Spectral Frequencies

Spectral frequencies are the result of harmonic generation and intermodulation of nonlinear circuits. In the case of single-tone excitations, we also use the term "harmonic frequencies" since they are the harmonics (multiples) of the fundamental frequency. In the two-tone case, spectral frequencies are linear combinations of the two tone frequencies.

Mathematically, there is possibly an infinite number of harmonic or spectral frequencies. In practice, only a limited number of them are actually included in the analysis.

### The Highest Harmonic

You can specify the highest harmonic to be included in the large-signal analysis in the CIRCUI statement.

#### Syntax:

```
CIRCUIT HARM=m ...;
```

*m* is the highest harmonic to be included in large-signal analysis,  $m_0 \leq m \leq 16$ , where *m*<sub>0</sub> is the highest harmonic among all the sources.

Default for linear circuits:  $m = m_0$

Default for nonlinear circuits:  $m = m_0 + 2$ , subject to  $4 \leq m \leq 16$

In the case of single-tone excitations, the harmonic frequencies are given by

$$k * f, \quad k = 0, 1, 2, \dots, m$$

where *f* is the fundamental frequency.

## MIXER and INTMOD

In the two-tone case, you can choose between two different methods of generating the spectral frequencies in the CIRCUIt statement.

### Syntax

```
CIRCUIT HARM=m MIXER INTMOD;
```

You can specify *either* MIXER or INTMOD. The default is MIXER.

If MIXER is specified, the spectral frequencies are generated as

$$k * f1 - \Delta f, \quad k * f1, \quad k * f1 + \Delta f, \quad k = 0, 1, \dots, m$$

where  $\Delta f = |f1 - f2|$ ,  $f1$  and  $f2$  are the two tone frequencies, and only the non-negative frequencies resulting from the above formula are used.

This method assumes that the circuit is a mixer,  $f1$  is the local oscillator frequency,  $f2$  is the RF signal frequency, and  $\Delta f$  is the intermediate frequency. This method is suitable if the magnitude (power level) of the local oscillator is much greater than that of the RF signal.

If INTMOD is specified, the spectral frequencies are given by

$$f = k1 * f1 + k2 * f2, \quad \text{subject to } |k1| + |k2| \leq m \\ \text{and } f \geq 0$$

where  $k1$  and  $k2$  are integers, and  $f1$  and  $f2$  are the two tone frequencies. This method is applicable to intermodulation analysis in general.

## Label N\_Spectra and Array Spectral\_Freq

You can conveniently refer to the total number of positive harmonic/spectral frequencies generated through an automatically generated label N\_Spectra, which is defined as

$$N\_Spectra = \begin{cases} m & \text{for single-tone} \\ 3 * m + 1 & \text{for two-tone MIXER} \\ m * (m + 1) & \text{for two-tone INTMOD} \end{cases}$$

The values of the harmonic/spectral frequencies are stored in an automatically generated array Spectral\_Freq[0:N\_Spectra]. The length of the array Spectral\_Freq is N\_Spectra + 1, because the DC frequency is included: Spectral\_Freq[0] is always zero.

In the single-tone case, `Spectral_Freq[k]` contains the  $k$ th harmonic frequency. For example, if the fundamental frequency is  $f = 2$  and `HARM = 4`, then

```
N_Spectra          = 4
Spectral_Freq[0]    = 0
Spectral_Freq[1]    = 2
Spectral_Freq[2]    = 4
Spectral_Freq[3]    = 6
Spectral_Freq[4]    = 8
```

In the two-tone case, the spectral frequencies are sorted and stored in ascending order. For example, consider the two tone frequencies  $f1 = 1$ ,  $f2 = 1.007$  and `HARM = 3`.

If `MIXER` is specified, then

```
N_Spectra          = 10
Spectral_Freq[0]    = 0
Spectral_Freq[1]    = 0.007
Spectral_Freq[2]    = 0.993
Spectral_Freq[3]    = 1
Spectral_Freq[4]    = 1.007
Spectral_Freq[5]    = 1.993
Spectral_Freq[6]    = 2
Spectral_Freq[7]    = 2.007
Spectral_Freq[8]    = 2.993
Spectral_Freq[9]    = 3
Spectral_Freq[10]   = 3.007
```

If `INTMOD` is specified, then

```
N_Spectra          = 12
Spectral_Freq[0]    = 0
Spectral_Freq[1]    = 0.007
Spectral_Freq[2]    = 0.993
Spectral_Freq[3]    = 1
Spectral_Freq[4]    = 1.007
Spectral_Freq[5]    = 1.014
Spectral_Freq[6]    = 2
Spectral_Freq[7]    = 2.007
Spectral_Freq[8]    = 2.014
Spectral_Freq[9]    = 3
Spectral_Freq[10]   = 3.007
Spectral_Freq[11]   = 3.014
Spectral_Freq[12]   = 3.021
```



## Spectral Index Mapping

In the two-tone case, the spectral frequencies are linear combinations of the two tone frequencies  $f1$  and  $f2$ :

$$k1 * f1 + k2 * f2$$

Because the spectral frequencies are sorted into ascending order before they are stored, the relationship between the index of the array `Spectral_Freq` and the harmonic indices  $k1$  and  $k2$  are not immediately obvious.

A built-in function `Spectral_Index` is provided to map  $k1$  and  $k2$  to the index of the array `Spectral_Freq`:

$$\text{Spectral\_Index}(\text{FREQ}, k1, k2)$$

where the predefined label `FREQ` is used as an argument to make clear the dependency of the function upon the frequency variable.

This function returns an index such that

$$\text{Spectral\_Freq}[\text{Spectral\_Index}(\text{FREQ}, k1, k2)] = k1 * f1 + k2 * f2$$

If the  $(k1, k2)$  combination is out of range, i.e., if the corresponding frequency is not an element of `Spectral_Freq`, then `Spectral_Index` returns 0.

To illustrate, consider again the example

$$f1 = 1, \quad f2 = 1.007, \quad \text{HARM} = 3$$

For the MIXER case,

$k1$	$k2$	<code>Spectral_Index(FREQ, <math>k1</math>, <math>k2</math>)</code>	$k1 * f1 + k2 * f2$
0	0	0	<code>Spectral_Freq[0] = 0</code>
-1	1	1	<code>Spectral_Freq[1] = 0.007</code>
2	-1	2	<code>Spectral_Freq[2] = 0.993</code>
1	0	3	<code>Spectral_Freq[3] = 1</code>
0	1	4	<code>Spectral_Freq[4] = 1.007</code>
3	-1	5	<code>Spectral_Freq[5] = 1.993</code>
2	0	6	<code>Spectral_Freq[6] = 2</code>
1	1	7	<code>Spectral_Freq[7] = 2.007</code>
4	-1	8	<code>Spectral_Freq[8] = 2.993</code>
3	0	9	<code>Spectral_Freq[9] = 3</code>
2	1	10	<code>Spectral_Freq[10] = 3.007</code>

For the INTMOD case,

$k1$	$k2$	Spectral_Index(FREQ, $k1,k2$ )	$k1 * f1 + k2 * f2$
0	0	0	Spectral_Freq[0] = 0
-1	1	1	Spectral_Freq[1] = 0.007
2	-1	2	Spectral_Freq[2] = 0.993
1	0	3	Spectral_Freq[3] = 1
0	1	4	Spectral_Freq[4] = 1.007
-1	2	5	Spectral_Freq[5] = 1.014
2	0	6	Spectral_Freq[6] = 2
1	1	7	Spectral_Freq[7] = 2.007
0	2	8	Spectral_Freq[8] = 2.014
3	0	9	Spectral_Freq[9] = 3
2	1	10	Spectral_Freq[10] = 3.007
1	2	11	Spectral_Freq[11] = 3.014
0	3	12	Spectral_Freq[12] = 3.021

You can use the function `Spectral_Index` to select a specific mixing or intermodulation product. For instance, in a two-tone intermodulation analysis, you may be particularly interested in the third-order intermodulation products

$$2 * f1 - f2 \quad \text{and} \quad 2 * f2 - f1$$

which can be mapped to the array `Spectral_Freq` using the indices given by

$$K21 = \text{Spectral\_Index}(\text{FREQ}, 2, -1);$$

and

$$K12 = \text{Spectral\_Index}(\text{FREQ}, -1, 2);$$

In other words,

$$\text{Spectral\_Freq}[K21] = 2 * f1 - f2$$

$$\text{Spectral\_Freq}[K12] = 2 * f2 - f1$$

## Large-Signal Response Labels

The built-in large-signal response labels are listed in Table 6.6.

**TABLE 6.6 LARGE-SIGNAL RESPONSES**

Array	Response
<i>MVname</i> [0:N_Spectra]	magnitudes of voltage spectrum of a port or an ideal current source
<i>PVname</i> [0:N_Spectra]	phases (in degrees) of voltage spectrum of a port or an ideal current source
<i>MIname</i> [0:N_Spectra]	magnitudes of current spectrum of a port or an ideal voltage source
<i>PIname</i> [0:N_Spectra]	phases (in degrees) of current spectrum of a port or an ideal voltage source
<i>Mvname</i> [0:N_Spectra]	magnitudes of voltage spectrum of a voltage label
<i>Pvname</i> [0:N_Spectra]	phases (in degrees) of voltage spectrum of a voltage label
<i>Miname</i> [0:N_Spectra]	magnitudes of current spectrum of a current label
<i>Piname</i> [0:N_Spectra]	phases (in degrees) of current spectrum of a current label
<i>PWname</i> [0:N_Spectra]	magnitudes of power spectrum (in watts) of a port

- ▷ *N\_Spectra* is the number of positive spectral frequencies
- ▷ *name* is a user-defined string, as in *PORT.name* ...  
*VSOURCE.name* ...  
*ISOURCE.name* ...
- ▷ *vname* is a user-defined string, as in *VLABELvname* ...
- ▷ *iname* is a user-defined string, as in *ILABELiname* ...

☞ See Chapter 4 for further details on arrays and array operations.

## Voltage, Current and Power Spectra of Ports

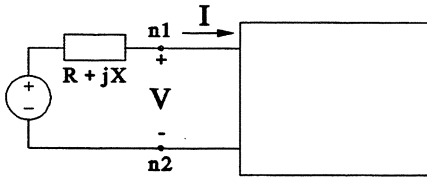


Fig. 6.11 Voltage and current of a port.

Example:

```
PORT.out 5 6 ...;
```

The corresponding large-signal response labels are

```
MVout[0:N_Spectra], PVout[0:N_Spectra],
MIout[0:N_Spectra], PIout[0:N_Spectra],
PWout[0:N_Spectra]
```

The magnitude of the power spectrum is related to the voltage and current spectra by

$$PWout[k] = -0.5 * MVout[k] * MIout[k] * \cos(\theta)$$

for  $k = 0, 1, \dots, N\_Spectra$ , where

$$\theta = (PVout[k] - PIout[k]) * PI / 180$$

Note the minus sign in the formula for  $PWout$ . It is because that the power spectrum response represents the power delivered to the port. The direction of power delivery is opposite to the direction of  $I$  shown in Fig. 6.11.

The unit of  $PWout$  is watt. You can create power spectrum in dBm as postprocessed responses (see Section 6.11).

## Current Spectra of Ideal Voltage Sources

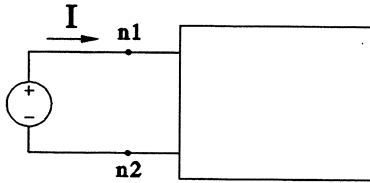


Fig. 6.12 Current of an ideal voltage source.

Example:

```
VSOURCE.bias 1 2 ...;
```

The corresponding large-signal response labels are

```
MIbias[0:N_Spectra], PIbias[0:N_Spectra]
```

## Voltage Spectra of Ideal Current Sources

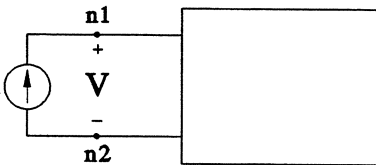


Fig. 6.13 Voltage of an ideal current source.

Example:

```
ISOURCE.bias 1 2 ...;
```

The corresponding large-signal response labels are

```
MVbias[0:N_Spectra], PVbias[0:N_Spectra]
```

## Voltage Labels

Example:

```
VLABEL.V52 5 2;
```

The corresponding large-signal response labels are

```
MV52[0:N_Spectra], PV52[0:N_Spectra]
```

## Current Labels

Example:

```
ILABEL.Current_Probe 5 2;
```

The corresponding large-signal response labels are

```
MCurrent_Probe[0:N_Spectra], PCurrent_Probe[0:N_Spectra]
```



### Response Label Names Must Be Unique

The response label names created according to the templates listed in Table 6.6 must be unique. They cannot duplicate any existing label names defined by the user.

For example, the voltage spectrum of a port defined by

```
PORT.output 5 6 ...;
```

are represented by the arrays `MVoutput` and `PVoutput`. If either of these names is already taken by an existing label, the file parser will signal an error.

Therefore, when creating labels, avoid using names which may conflict with the templates for response labels, as listed in Tables 6.1 – 6.6.

## Response Spectral Indices

The large-signal voltage and current spectra are stored in the arrays in exactly the same order as the spectral frequencies are stored in the array `Spectral_Freq`.

For single-tone, the  $k$ th array element is the  $k$ th harmonic component of the voltage or current spectrum.

In the two-tone case, you can use the built-in function `Spectral_Index` to map the array indices as described earlier in this section.

For example, assume that `MVout` is a large-signal response array. From the results of a two-tone intermodulation analysis, you can select the third-order intermodulation products by

```
K21 = Spectral_Index(FREQ, 2, -1);
MVout_21 = MVout[K21];
```

```
K12 = Spectral_Index(FREQ, -1, 2);
MVout_12 = MVout[K12];
```

Then, the user-defined labels `MVout_21` and `MVout12` correspond to the spectral frequencies

$$2 * f1 - f2 \quad \text{and} \quad 2 * f2 - f1$$

respectively, where  $f1$  and  $f2$  are the tone frequencies.

## 6.11 Postprocessing Responses

You can create customized responses by postprocessing, i.e., by using expressions which involve the built-in response labels.

User-created responses can be displayed and optimized just like the built-in responses.

### Power in dBm

The built-in response array *PWname* represents the output power spectrum of the port identified by *name*.

The unit of *PWname* is watt. You can define the output power in dBm as

$$Pout\_dBm[0:N\_Spectra] = 10 * \log_{10}(Pout) + 30;$$

where "out" is assumed to be the name of a port.

To safeguard against numerical overflow, you use a conditional expression:

$$Pout\_dBm[0:N\_Spectra] = \text{if}(PWout > Pmin) (10 * \log_{10}(PWout) + 30) \\ \text{else} (10 * \log_{10}(Pmin) + 30);$$

where *Pmin* is a suitable threshold value (e.g., 1.0E-10).

You can also utilize the predefined label NAN ("Not A Number", see Chapter 4):

$$Pout\_dBm[0:N\_Spectra] = \text{if}(PWout > Pmin) (10 * \log_{10}(PWout) + 30) \\ \text{else} (NAN);$$

Label values marked as NAN will be suppressed from the graphical displays in OSA90.

### Spectra in Rectangular Form

The built-in spectrum responses are given in polar form. You can convert them into rectangular form using expressions, such as

$$RVout[0:N\_Spectra] = MVout * \cos(PVout * PI / 180);$$

$$IVout[0:N\_Spectra] = MVout * \sin(PVout * PI / 180);$$

This can be done easily by the built-in transformation **MP2RI** (Chapter 4):

$$MP2RI(MVout, PVout, RVout[0:N\_Spectra], IVout[0:N\_Spectra]);$$



## Time-Domain Waveforms

The built-in large-signal responses are in the frequency domain.

You can use the built-in DFT transformation described in Chapter 4 to obtain the time-domain waveforms, such as

$$Vout\_T[nt] = DFT\_FT(RVout, IVout);$$

where *RVout* and *IVout* are the spectrum in rectangular form, following the preceding example, and *nt* is the desired number of time points.

In the two-tone case, the spectral frequencies are not uniformly spaced and must also be supplied to the DFT function:

$$Vout\_T[nt] = DFT\_FT(RVout, IVout, Spectral\_Freq, t0, t1);$$

where *Spectral\_Freq* is the spectral frequency array (described in this section), and *t0* and *t1* specify the time interval in which *nt* evenly spaced time points are to be sampled.

☛ OSA90 can also display waveforms as graphical views (see Chapter 10).

## 6.12 Linear Subcircuits

### Syntax

```
SUBCIRCUIT name n1 n2 ... nk {
    elements;
};
```

*name* is a required character string to identify the subcircuit.  
*n1*, *n2*, ..., *nk* represent nodes. The linear subcircuit is defined as a  $(k - 1)$  port.  
 The node *nk* is the reference node.

### Example:

```
SUBCIRCUIT T_Section 1 3 0 {
    CAP 1 2 C=10pF;
    RES 2 0 R=5;
    IND 2 3 L=2nH;
};
```

This defines a two-port linear subcircuit consisting of three elements.

Once defined, a linear subcircuit can be used like an element:

```
T_Section 1 2 0;
T_Section 2 3;
```

The number of nodes must be consistent with the subcircuit definition. The last node (which is the reference node of the subcircuit) is optional. If omitted, the last node is grounded by default.

### Subcircuits Must Be Defined Before the Top Level Circuit

All SUBCIRCUIT definitions must appear before the top level circuit.

The following example violates this rule:

```
Model
    RES 1 2 R=50;

    SUBCIRCUIT T_SECTION 1 3 0 {
        ...
    };
```

The subcircuit cannot be defined after the RES element which is part of the main circuit.

### Subcircuit Internal Nodes

The node numbers and names within a linear subcircuit are internal to that subcircuit. They can be reused for other definitions after the SUBCIRCUIT statement.

Example:

```

SUBCIRCUIT T_SECTION 1 3 0 (
    ...
);
CAP 1 2 C=10pF;

```

In this example, the node number 1 appears twice: first as part of the subcircuit definition and then as part of the main circuit. It represents two separate and distinct nodes.

### Subcircuits Cannot Exceed 64 External Nodes

A linear subcircuit must be configured as a 1- to 63-port. In other words, in a subcircuit the nodes for external connection cannot exceed 64.

### Subcircuits Cannot Contain Nonlinear Elements or Sources

Linear subcircuits cannot contain any nonlinear elements (nonlinear elements are listed in Chapter 7).

Linear subcircuits must not contain any PORT, VSOURCE, ISOURCE, VLABEL or ILABEL statements.

### Comparison with Symbolic Subcircuits

In comparison, symbolic subcircuits (Section 6.17) are subjected to fewer restrictions.

### Nested Linear Subcircuits

Linear subcircuits can be nested, i.e., the definition of a linear subcircuit can contain references to other linear subcircuits that are already defined.

## Using Subcircuits as Port Terminations


**Syntax:**

```
PORT n1 n2 Z=subcircuit ... ;
```

*subcircuit* must be the name of a 1-port linear subcircuit which is already defined.

**Example:**

```
SUBCIRCUIT ComplexLoad 1 0 {  
  RES 1 2 R=50;  
  CAP 2 0 C=5pF;  
  IND 2 0 L=2nH;  
};  
  
...  
  
PORT ... Z=ComplexLoad;
```

 See demo49 for an example of using a linear subcircuit as a complex load for phase plane plotting.

## 6.13 User-Defined Linear Models

### Syntax

```

ELEMENT name n1 n2 ... nk par_1=x1 ... par_m=xm {
    local labels and expressions;

    Datapipe: ...;

    elements;

};

```

*name* is a required character string to identify the user-defined linear element. *n1*, *n2*, ..., *nk* represent nodes. The element is defined as a (*k* - 1) port subcircuit. The node *nk* is the reference node. *par\_1*, ..., *par\_m* are the parameter names of the element, and *x1*, ..., *xm* are the corresponding default values.

### Example:

```

ELEMENT CAP_Q 1 0 C=0 Q=10000 F=1 {
    G = 2 * PI * F * C / Q;

    CAP 1 0 C=C;
    RES 1 0 R=(1/G);
};

```

This creates a user-defined element named CAP\_Q which is a model for lossy capacitors.

### User-Defined Element Parameters

Each user-defined linear element can have up to 64 parameters.

You define the names of the parameters and their default values following the nodes for external connection and before the opening curly brace "{". The generic form is

$$\text{parameter\_name} = \text{default\_value}$$

For example, the user-defined element CAP\_Q has three parameters, namely C, Q and F.

When a user-defined element is used, each parameter can be assigned a value, specified by a label or defined as an optimization variable.

### Example:

```

CAP_Q 7 23 C=?5pF? Q=1200 F=2GHZ;

```

If a parameter is not explicitly specified, it assumes the default value.

Example:

```
CAP_Q 7 23 C=?5pF? F=2GHZ;
```

The omitted parameter Q assumes its default value of 10000.

## Local Labels and Expressions

All the parameters of a user-defined element are local labels. In other words, you are free to choose any names for the parameters without fearing that they might be in conflict with other identifiers.

Furthermore, you can define additional local labels within the element definition. For example, the definition of CAP\_Q includes a local label G, which is expressed as a function of the element parameters.

Local labels are not visible beyond the end of the element definition.

## Visibility of Global Labels

Expressions within an ELEMENT definition can refer to not only local labels but also global labels (i.e., labels defined outside the ELEMENT definition).

However, if the name of a local label duplicates the name of a global label, then that global label will not be visible within the ELEMENT definition. In other words, local labels override global labels when they overlap.

Example:

```
Width: ...;
Height: ...;
Length: ...;

ELEMENT MyElem 1 2 3 0 Width=1mil ... {
  Length: ...;
  Volume = Width * Length * Height;
  ...
  ...
};
```

In the expression that defines Volume, Width and Length refer to local labels (which override the global labels of the same name), and Height refers a global label.

 demo60.ckt implements a YIG resonator model as a user-defined element.

## Element Definitions Must Appear Before Circuit Definition

The definitions of user-defined elements must appear before the circuit definition (i.e., before any connections of elements by nodes).

Example:

```
RES 1 0 R=10;  
ELEMENT ... ( ... );
```

This will cause an error message because an element RES is connected before the definition of ELEMENT.

## Using Datapipes within Element Definitions

User-defined elements can include Datapipe connections to external simulators. This opens the opportunity for you to incorporate in-house models into OSA90 as fully parameterized and optimizable elements in a truly seamless fashion.

## Element Definitions Cannot Exceed 64 External Nodes

A user-defined linear model must be configured as a 1- to 63-port. In other words, in a subcircuit the nodes for external connection cannot exceed 64.

## Creating Sharable Custom Element Library

You can place the definitions of custom elements in include files so that they can be shared by different applications and different users.

The Empipe element library is an excellent example. It is a library of microstrip structure elements to be calculated by an external EM simulator through Datapipe. For details, see the Empipe User's Manual.

## 6.14 User-Defined Nonlinear Models

One of the most powerful features of OSA90 is user-definable nonlinear models. There are two different types of user-definable models:

- ▷ using a predefined (fixed) topology for a specific type of device, and user-defined model equations
- ▷ using nonlinear controlled sources to build models of arbitrary topology and model equations

User-definable models provide you with the flexibility of customizing the built-in models and creating your own unique models, in an easy and friendly way.

### Nonlinear Currents and Charges

You create a nonlinear model by defining currents and charges as nonlinear functions of voltages. The model equations must be defined in the time-domain. In other words, the currents, charges and voltages in the model equations represent time-domain instantaneous variables.

### Controlling Voltages

The voltages that control the nonlinear currents and charges are state variables determined by solving the nonlinear circuit equations iteratively.

In order to define the model equations, you will need access to the instantaneous voltages. This can be done using voltage labels as described in Section 6.6.

#### Syntax

```
VLABEL.name n1 n2 TAU= $\tau$ ;
```

where *name* is a string to identify the voltage label, *n1* and *n2* are nodes, TAU= $\tau$  specifies an optional time delay,  $\tau$  can be a constant, variable or label.

Example:

```
VLABEL.Vds @drain @source;
```

The voltage label *Vds* represents the instantaneous voltage between the nodes *@drain* and *@source*.

You can also specify a time delay for the voltage label, for example,

```
VLABEL.Vgs_tau @gate @source TAU=3ps;
```

*Vgs\_tau* represents the voltage between the nodes *@gate* and *@source*, delayed by 3ps.



## Using Current Labels as Controlling Variables

You may attempt to use current labels (see Section 6.6) in the model equations to define nonlinear current-controlled-currents or current-controlled-charges. Be warned, however, that this can lead to a set of nonlinear equations which explicitly depend on its own results. As a consequence, the nonlinear solver may experience convergence difficulties.

## User-Definable Device Models

User-definable device models are designed to represent specific types of devices. Each model has a fixed topology, i.e., the number and structure of the currents and charges are predefined. The model equations are user-definable, i.e., you can express the nonlinear currents and charges in the model as functions of arbitrary model parameters and voltage labels.

Both built-in and user-definable nonlinear models are catalogued in Chapter 7.

For example, consider the Curtice cubic FET model. It is available as a built-in model (FETC), but you can also recreate it using the user-definable model FETU1:

```
VLABEL.Vgs_tau @gate @source TAU=Tau1;
VLABEL.Vds @drain @source;

V1 = Vgs_tau * (1 + Beta * (Vds0 - Vds));

FETU1 @gate @drain @source
  Igs = ...
  Ids = ((A0 + A1*V1 + A2*V1^2 + A3*V1^3) * tanh(Gamma*Vds))
  Qgs = ...;
```

where Tau1, Beta, Vds0, A0, A1, A2, A3 and Gamma represent user-defined model parameters (they can be given arbitrary names), and Vgs\_tau and Vds are voltage labels.

In this example, using FETU1 instead of the built-in FETC gives you the opportunity to modify the model equations if there is a need to do so.

## Nonlinear Controlled Sources

If you wish to modify not only the model equations but also the model topology, you can use nonlinear controlled current and charge sources to create your own unique models with arbitrary parameters and topology.

### Syntax

```
NLCCS n1 n2 I=expression;
```

```
NLCQ n1 n2 Q=expression;
```

where *n1* and *n2* are nodes, and *expression* is a user-defined expression.

You create a set of model parameters and a set of controlling voltage labels. Then, you express the current or charge as a function of the model parameters and voltage labels.

For example, you can apply the Curtice cubic formula to a generic nonlinear branch using the nonlinear controlled current source NLCCS:

```
VLABEL.Vds @drain @source;
VLABEL.Vgs_tau @gate @source TAU=Tau1;

V1 = Vgs_tau * (1 + Beta * (Vds0 - Vds));

NLCCS @drain @source I = ((A0 + A1*V1 + A2*V1^2 + A3*V1^3) *
                           tanh(Gamma*Vds));
```

where Tau1, Beta, Vds0, A0, A1, A2, A3 and Gamma are user-defined parameters, and Vds and Vgs\_tau are voltage labels.

Both the controlled source (NLCCS or NLCQ) and its controlling voltages can be arbitrarily placed within the circuit.

☞ See also the nonlinear element library in Chapter 7.

## Preprogrammed Models

A number of nonlinear FET models have been preprogrammed using the user-definable models. They are contained in the demo examples supplied with the OSA90 program, as listed in Table 6.7. You can use these examples as templates for creating your own models, or you can incorporate them directly into your own input files.

**TABLE 6.7 PREPROGRAMMED MODELS**

<b>File</b>	<b>Model</b>
demo14.ckt	Curtice and Ettenberg cubic symmetrical FET model
demo15.ckt	Curtice and Ettenberg cubic asymmetrical FET model
demo16.ckt	Materka and Kacprzak FET model
demo17.ckt	Plessey FET model
demo18.ckt	TriQuint's Own Model (TOM) for FETs
demo47.ckt	Raytheon-Statz FET model

## 6.15 ImportData Block

You can import linear subcircuits into OSA90 as n-port data ("black-box").

A linear subcircuit to be imported must be an n-port with a common reference node (i.e., all the ports are defined w.r.t. to the same reference node). The subcircuit must be represented by its scattering, admittance or impedance matrix ( $S/Y/Z$  matrix) supplied in the ImportData block.

### Syntax

```

ImportData
  PARAMETER ...;
  FORMAT ...;
  S/Y/Z data

  ...

  PARAMETER ...;
  FORMAT ...;
  S/Y/Z data
End

```

The block may contain a number of subcircuits, each represented by a data set which consists of a PARAMETER statement, a FORMAT statement and the  $S/Y/Z$  data.

### PARAMETER Statement

Each import data set begins with a PARAMETER statement:

### Syntax

```
PARAMETER NAME=label RREF=x;
```

where *label* is a character string to identify the set of imported data.

If the data is  $S$  parameters, RREF can be used to specify the reference impedance. The default reference impedance is 50OH.

The *label* following the keyword NAME is required, so that when the imported subcircuit is used in the Model block, it can be identified by *label*.

## The QUIET Keyword

An optional keyword QUIET can be used to instruct OSA90 to ignore unrecognizable entries. This feature makes it more friendly for you to share data files between OSA90 and other programs, because you will be able to keep keywords which are unknown to OSA90 but meaningful to other applications.

### Syntax

```
PARAMETER NAME=label RREF=x QUIET foreign_keyword-...;
```

Example:

```
PARAMETER ... QUIET TEMPERATURE=273 CODE=12;
```

where TEMPERATURE and CODE are keywords unknown to OSA90 but may be useful for another program. With the QUIET keyword, the entries TEMPERATURE=273 and CODE=12 will be ignored. Otherwise they would be treated as errors.

## FORMAT Statement

For each data set, following the PARAMETER statement, a FORMAT statement is required.

### Syntax

```
FORMAT keyword(unit) keyword(unit) ...;
```

where *keyword* represents a data keyword and *unit* is optional.

Example:

```
PARAMETER NAME=Import 2 Port;
FORMAT FREQ(GHZ) RY11 IY11 RY12 IY12 RY21 IY21 RY22 IY22;

1.0 0.0412 -0.6259 -0.0392 0.6266 -0.0392 0.6266 0.0392 -0.6266
2.0 0.0118 -0.3120 -0.0098 0.3133 -0.0098 0.3133 0.0098 -0.3133
5.0 0.0035 -0.1222 -0.0015 0.1253 -0.0015 0.1253 0.0015 -0.1253
6.0 0.0031 -0.1007 -0.0011 0.1044 -0.0011 0.1044 0.0011 -0.1044
7.0 0.0028 -0.0851 -0.0008 0.0895 -0.0008 0.0895 0.0008 -0.0895
```

Each keyword in the FORMAT statement represents a column of data. Using the FORMAT statement, you can indicate the data type, order and units, without being restricted to a rigid order of data entry.

The keywords for imported data are listed in Table 6.8.

TABLE 6.8 KEYWORDS FOR IMPORTED DATA

Keyword	Default Unit	Description
FREQ	GHZ	frequency
MS <sub>ij</sub>		magnitude of $S_{ij}$
PS <sub>ij</sub>	DEG	phase of $S_{ij}$
RS <sub>ij</sub>		real part of $S_{ij}$
IS <sub>ij</sub>		imaginary part of $S_{ij}$
MY <sub>ij</sub>	/OH	magnitude of $Y_{ij}$
PY <sub>ij</sub>	DEG	phase of $Y_{ij}$
RY <sub>ij</sub>	/OH	real part of $Y_{ij}$
IY <sub>ij</sub>	/OH	imaginary part of $Y_{ij}$
MZ <sub>ij</sub>	OH	magnitude of $Z_{ij}$
PZ <sub>ij</sub>	DEG	phase of $Z_{ij}$
RZ <sub>ij</sub>	OH	real part of $Z_{ij}$
IZ <sub>ij</sub>	OH	imaginary part of $Z_{ij}$

☞ See Chapter 3 for the complete list of physical units.

All the keywords used in one **FORMAT** statement (one data set) must be consistent. In other words, you cannot mix  $S$  parameters with  $Y$  parameters, or mix rectangular form with polar form, in the same data set.

When the keywords listed in Table 6.8 are used, the actual number of ports is determined from the highest index specified. However, the maximum number of ports cannot exceed 9. To import  $n$ -port data for  $n \geq 10$  you need to use the keywords listed in Table 6.9.



You must supply complete  $S/Y/Z$  matrices. For instance, a set of  $Y$  parameters for a two-port must include  $Y_{11}$ ,  $Y_{12}$ ,  $Y_{21}$ , and  $Y_{22}$ .

The  $S/Y/Z$  matrix for each frequency may occupy as many text lines as necessary, for example,

```
PARAMETER NAME=Import_2_Port;
FORMAT FREQ(GHZ) RY11 IY11 RY12 IY12 RY21 IY21 RY22 IY22;

      1.0    0.0412  -0.6259  -0.0392   0.6266
           -0.0392   0.6266   0.0392  -0.6266

      2.0    0.0118  -0.3120  -0.0098   0.3133
           -0.0098   0.3133   0.0098  -0.3133

      ...
```

However, each matrix (frequency) must begin on a new line.

## Data Interpolation

The frequencies supplied for the imported data need not be exactly the same frequencies involved in the simulation by OSA90. If necessary, OSA90 will interpolate or extrapolate the imported data using cubic splines. This requires that the imported data include at least three different frequencies. It is desirable and recommended that the imported data cover the frequency range of the simulation. For harmonic and spectral analyses, the frequency range is from DC to the highest harmonic/spectral frequency.

## Super Keywords

To accommodate  $n$ -ports with  $n \geq 10$ , as well as for your convenience, several super keywords are defined to represent typical sets of keywords for imported subcircuit data. These are listed in Table 6.9.

**TABLE 6.9 SUPER KEYWORDS FOR IMPORTED DATA**

Super Keyword	Sequence of Keywords Represented
S2MP	MS11, PS11, MS21, PS21, MS12, PS12, MS22, PS22
S2RI	RS11, IS11, RS21, IS21, RS12, IS12, RS22, IS22
S <sub>n</sub> MP $n \neq 2$	MS11, PS11, MS12, PS12, ..., MS1 <sub>n</sub> , PS1 <sub>n</sub> , ..., MS <sub>n</sub> <sub>n</sub> , PS <sub>n</sub> <sub>n</sub>
S <sub>n</sub> RI $n \neq 2$	RS11, IS11, RS12, IS12, ..., RS1 <sub>n</sub> , IS1 <sub>n</sub> , ..., RS <sub>n</sub> <sub>n</sub> , IS <sub>n</sub> <sub>n</sub>
Y <sub>n</sub> MP	MY11, PY11, MY12, PY12, ..., MY1 <sub>n</sub> , PY1 <sub>n</sub> , ..., MY <sub>n</sub> <sub>n</sub> , PY <sub>n</sub> <sub>n</sub>
Y <sub>n</sub> RI	RY11, IY11, RY12, IY12, ..., RY1 <sub>n</sub> , IY1 <sub>n</sub> , ..., RY <sub>n</sub> <sub>n</sub> , IY <sub>n</sub> <sub>n</sub>
Z <sub>n</sub> MP	MZ11, PZ11, MZ12, PZ12, ..., MZ1 <sub>n</sub> , PZ1 <sub>n</sub> , ..., MZ <sub>n</sub> <sub>n</sub> , PZ <sub>n</sub> <sub>n</sub>
Z <sub>n</sub> RI	RZ11, IZ11, RZ12, IZ12, ..., RZ1 <sub>n</sub> , IZ1 <sub>n</sub> , ..., RZ <sub>n</sub> <sub>n</sub> , IZ <sub>n</sub> <sub>n</sub>

All the keyword sequences represent matrices entered row-wise, except S2MP and S2RI which are arranged column-wise as conventionally done.

**Example:**

```
FORMAT  FREQ  Y2RI;
```

which is equivalent to

```
FORMAT  FREQ  RY11  IY11  RY12  IY12  RY21  IY21  RY22  IY22;
```

You can also specify a data unit, such as

```
FORMAT  FREQ  Z2RI(KOH);
```

which is equivalent to

```
FORMAT  FREQ  RZ11(KOH)  IZ11(KOH)  RZ12(KOH)  IZ12(KOH)
           RZ21(KOH)  IZ21(KOH)  RZ22(KOH)  IZ22(KOH);
```

## Imported Data Can Represent up to 31 Ports

OSA90 supports linear multiport subcircuits represented by *S/Y/Z* imported data for up to 31 ports. For *n*-ports with  $n < 10$  ports the keywords listed in either Table 6.8 or Table 6.9 can be used. For *n*-ports with  $n \geq 10$  only the keywords listed in Table 6.9 can be used. For example, Y16RI represents the *Y* parameters of a 16-port supplied in the rectangular (real-imaginary) format.

Example:

```

ImportData

PARAMETER NAME=Y_16_PORT;
FORMAT    FREQ  Y16RI;

      2GHZ  ry11 iy11 ry12 iy12 ... ry1,16 iy1,16
                                     ... ry16,16 iy16,16

      3GHZ  ry11 iy11 ry12 iy12 ... ry1,16 iy1,16
                                     ... ry16,16 iy16,16

...

end

```

The matrix for each frequency may occupy as many text lines as necessary. However, each matrix (frequency) must begin on a new line.

## The QUIET Keyword in FORMAT

An optional keyword QUIET can be used to instruct OSA90 to ignore unrecognizable entries. This feature makes it more friendly for you to share data files between OSA90 and other programs, because you will be able to keep keywords which are unknown to OSA90 but meaningful to other applications.

### Syntax

```

FORMAT keyword(unit) ... QUIET foreign_keyword(unit) ...;

```

Example:

```

FORMAT QUIET  FREQ  MS11  PS11  RETURN_LOSS(DB);
           XXXX  XXXX  XXXX  XXXX
...

```

where RETURN\_LOSS(DB) is unknown to OSA90 as an imported data keyword. With QUIET, you instruct OSA90 to ignore such foreign keywords and to skip over the corresponding columns of data.



## Reference to Imported Data

Imported linear subcircuits can be used as DATAPORT elements in the Model block.

### Syntax

```
DATAPORT n1 n2 ... nk DATA=label;
```

where  $n_1, n_2, \dots, n_k$  are nodes,  $k \leq (\text{number of ports} + 1)$ , and *label* identifies an imported data set in the ImportData block.

Example:

```
ImportData
  PARAMETER NAME=Imported_2_Port;
  FORMAT FREQ(GHZ) Y2RI(/KOH);
          0.000   XXX XXX XXX XXX XXX XXX XXX XXX
          0.500   XXX XXX XXX XXX XXX XXX XXX XXX
          ...
          10.000  XXX XXX XXX XXX XXX XXX XXX XXX
End

Model
  DATAPORT 1 2 3 DATA=Imported_2_Port;
  ...
End
```

The imported data must describe an  $n$ -port with a common reference node (i.e., all the ports are defined w.r.t. to the same reference node). The DATAPORT statement may specify up to  $n + 1$  nodes (i.e., including the reference node), where  $n$  is the number of ports. If the number of nodes specified is less than  $n + 1$ , then the unspecified nodes are assumed to be zero.

☞ Also see Chapter 8 for DATAPORT.

## Imported One-Ports as Terminations

Imported one-ports can be used to define frequency-dependent terminations for the external ports of the circuit (Section 6.5).

Example:

```

ImportData
  PARAMETER NAME=Imported_Impedance;
  FORMAT   FREQ(GHZ)  MZ11  PZ11;
           0.000      xxx  xxx
           0.500      xxx  xxx
           ...
           10.000     xxx  xxx
End

Model
  ...

  PORT.output_port @output @ground Z=Imported_Impedance;
  ...
End

```



The use of the keyword *Z* in the port definition does not imply that the data supplied in the `ImportData` block can only be *Z* (impedance) data. The data supplied can also be *S* or *Y* parameters, OSA90 will automatically make the necessary conversion. However, the data must be convertible to a finite and nonzero impedance value at all the frequencies involved in the circuit simulation.

## Importing Data in EEsof Format

*S*-parameter data files in EEsof (Touchstone®) format can be imported into OSA90 directly.

### Syntax

```
ImportData
  #include "file_name.snp"
  ...
End
```

where  $n$  is the number of ports,  $1 \leq n \leq 9$ .

### Example:

```
ImportData
  #include "nec700.s2p"
End
```

where "nec700.s2p" is a two-port *S*-parameter data file in EEsof format.

The imported data can be used to define a linear *n*-port in the Model block.

### Syntax

```
DATAPORT n1 n2 ... nk DATA=file_name;
```

where  $n_1, n_2, \dots, n_k$  are nodes,  $k \leq (\text{number of ports} + 1)$ , and *file\_name* identifies the imported data file in EEsof format.

### Example:

```
Model
  ...
  DATAPORT 1 2 Data=nec700;
  ...
End
```

## Other Flexible Formats for Import S/Y/Z Data

*S/Y/Z* data representing linear *n*-ports can be imported into OSA90 as data arrays using SPORT, YPORT and ZPORT also. A large variety of data formats can be accommodated.

☛ See Chapter 8 for details on SPORT, YPORT and ZPORT.

## 6.16 LINEAR Datapipe

In addition to the ImportData block, Datapipe can also be used to supply  $S/Y/Z$  matrices as imported linear subcircuits. The LINEAR Datapipe protocol is designed for this purpose.

### Syntax

```
Datapipe: LINEAR FILE="filename"
          N_INPUT=m INPUT=(x1, ..., xn)
          N_PORT=n NAME=label FORMAT=syz
          TIMEOUT=k;
```

where *filename* is the name of the child program, *m* is the number of inputs to the child,

*n* is the number of ports of the imported linear subcircuit,  $1 \leq n \leq 31$ ,

*label* is a character string to identify the imported subcircuit, *syz* can be specified as S, Y or Z (the default is S), and *k* is the number of seconds for time-out.

See Chapter 5 for general descriptions of Datapipe.

The imported subcircuits can be used as DATAPORT elements in the Model block.

### Syntax

```
DATAPORT n1 n2 ... nk DATA=label;
```

where *n1*, *n2*, ... *nk* are nodes,  $k \leq (\text{number of ports} + 1)$ ,

and *label* identifies a subcircuit imported through LINEAR Datapipe.

Example:

Model

```
Datapipe: LINEAR FILE="my_prog"
          N_INPUT=5 INPUT=(...)
          N_PORT=2 NAME=Imported_2_Port FORMAT=Y;

DATAPORT 1 2 3 DATA = Imported_2_Port;
...
```

End

Through the LINEAR Datapipe, the child program calculates and returns one  $S/Y/Z$  matrix at a time. Typically, the predefined label **FREQ** is included in the inputs to the child program so that the data is calculated at the proper frequency.

The imported linear subcircuit must be an  $n$ -port with a common reference node (i.e., grounded to the reference node), and described by an  $n \times n$  definite  $S$ ,  $Y$  or  $Z$  matrix. The DATAPORT statement may specify up to  $n + 1$  nodes (i.e., including the reference node). If the number of nodes specified is less than  $n + 1$ , then the unspecified nodes are assumed to be zero.

### Reference Impedance for $S$ Parameters

The reference impedance for  $S$  parameters imported using LINEAR Datapipe is assumed to be 50 ohm. Please make sure that your imported  $S$  parameters are indeed calculated with respect to 50 ohm in the child program.

### Imported One-Ports as Terminations

Imported one-ports can be used to define frequency-dependent terminations for the external ports of the circuit (Section 6.5).

Example:

```

Model
    Datapipe: LINEAR FILE="my_prog"
                N_INPUT=5 INPUT=(...)
                N_PORT=1 NAME=Imported_Impedance FORMAT=Z;
    PORT.output_port @output @ground Z=Imported_Impedance;
    ...
End

```

## 6.17 Macros and Symbolic Subcircuits

The text macros supported by OSA90's file parser are very useful for simplifying input files and customizing the file syntax (see Chapter 3).

Text macros are identifiers (names) created by the `#define` preprocessor directive to replace complex and repetitive portions of text.

### Customized Keywords

Text macros can be used to customize the syntax by replacing the built-in keywords with identifiers which are more readable, more familiar to you, or compatible with some other programs.

Example:

```
#define User_Definable_FET_Model    FETU1
#define Gate_Source_Current         IGS
#define Gate_Drain_Current          IGD
#define Drain_Source_Current        IDS
#define Gate_Source_Charge          QGS
#define Gate_Drain_Charge           QGD

User_Definable_FET_Model 1 2 3
  Gate_Source_Current = expression1
  Gate_Drain_Current  = expression2
  Drain_Source_Current = expression3
  Gate_Source_Charge  = expression4
  Gate_Drain_Charge   = expression5;
```

These macros will be expanded by the file parser into

```
FETU1 1 2 3
  IGS = expression1
  IGD = expression2
  IDS = expression3
  QGS = expression4
  QGD = expression5;
```

In this example, the built-in keywords are replaced by lengthier but less cryptic text.

Another example:

```
#define Resistor    RES
#define Resistance(length, area)  R=(Resistivity*length/area)

Resistor @n1 @n2 Resistance(L1,A1);
Resistor @n3 @n4 Resistance(L2,A2);
```

The expanded text becomes

```
RES @n1 @n2 R=(Resistivity*L1/A1);
RES @n3 @n4 R=(Resistivity*L2/A2);
```

You can even redefine input file block names, such as

```
#define CKT Model
```

which allows you to use a CKT block instead of a Model block for circuit definition.

## Symbolic Subcircuits

A more sophisticated usage of macros is to define symbolic subcircuits.

A symbolic subcircuit can be quite simple, such as

```
#define Gate_Line(n1, n2) TEM n1 n2 Z=40 E=34 F=10GHz
```

which defines a subcircuit containing a single transmission line with constant parameters to be used as gate lines in an FET travelling wave amplifier. The two arguments are the nodes. By using `Gate_Line` with different arguments, you can place different gate lines at the appropriate nodes. For example, `Gate_Line(10,14)` places a line between the nodes 10 and 14. By modifying the macro definition, you can change all the lines simultaneously.

The `$` argument (Chapter 3) is very useful for defining symbolic subcircuits. For example,

```
#define T_SECTION(n1, n2, $) {
    IND n1 @center_node$ L=10nH;
    IND n2 @center_node$ L=10nH;
    CAP @center_node$ @ground C=0.5pF
}
```

This defines a grounded L-C T-section between the nodes `n1` and `n2`. The T-section contains an internal node which must be a different node each time the subcircuit is placed. Instead of passing a new node, a new index is passed as the `$` argument, and a new internal node is constructed using a template and the `$` argument.

For example,

```
T_SECTION(17, 20, 1);
T_SECTION(33, 14, 2);
```

The corresponding internal nodes will be `@center_node1` and `@center_node2`.

A more complex example defines one stage in a multi-stage FET amplifier:

```
#define Amplifier_Stage(input, output, E1, E2, $) {
    TEM input 0 Z=70 E=E1 F=18GHZ;
    TEM output 0 Z=50 E=E2 F=18GHZ;

    TEM input @open1$ 0 @open2$ Z=100 E=65 F=18GHZ;
    OPEN @open1$ @open2$;
    TEM input @ext_gate$ Z=100 E=25 F=18GHZ;

    Extrinsic2 @gate$ @drain$ @source$ @ext_gate$ @ext_drain$
        LG=0.02NH LD=0.03NH LS=0.03NH RG=3.5 RD=2 RS=4
        GDS=0.005 CDS=0.1PF CX=10pF;

    FETR @gate$ @drain$ @source$
        CGS0=CGS_$ CGD0=CGD_$
        ALPHA=1.9 BETA=0.019 THETA=0.0035 TAU=3PS
        LAMBDA=0.002 IS=5E-15 GMIN=1.0E-07 VBR=20;

    TEM @ext_drain$ output Z=100 E=15 F=18GHZ;
}
```

The arguments include the input and output nodes and a parameter TEM\_E. A large number of internal nodes are constructed from the \$ argument. Even the model parameters CGS\_\$ and CGD\_\$ are specified through the \$ argument. For instance, if the \$ argument is set to 1 for one particular reference to this symbolic subcircuit, then the actual parameters will be CGS\_1 and CGD\_1 (which must have been already defined).

## Comparison with Linear Subcircuits

Different from the linear subcircuits defined by SUBCIRCUIT statements (Section 6.12), symbolic subcircuits are not treated as separate structures in circuit simulation. Rather, symbolic subcircuits are expanded by the preprocessor into the overall circuit. The overall circuit is simulated on a single level.

Nonlinear elements can be included in symbolic subcircuits but not in linear subcircuits.

The use of linear subcircuits generally leads to computational savings. The use of symbolic subcircuits merely provides convenience of circuit description.

Symbolic subcircuits can be nested, i.e., a symbolic subcircuit can contain other symbolic subcircuits, up to 6 levels (layers). Linear subcircuits can also be nested and with virtually unlimited depth.

Symbolic subcircuits cannot contain explicit definition of optimization variables, otherwise the explicit definition will be duplicated each time the subcircuit is used. This gives rise to ambiguity and difficulties in updating the variables after optimization. Optimization variables should be *defined* as labels outside the symbolic subcircuit and then *referenced* in the symbolic subcircuit.



## 6.18 Oscillator Port

OSCPORT is a dedicated element for oscillator analysis.

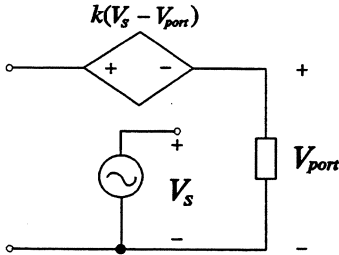


Fig. 6.14 Oscillator port.

OSCPORT contains a voltage source  $V_s$ , which represents an estimate of the voltage across the port termination, namely  $V_{port}$ , at the fundamental frequency.

The controlled voltage source provides a stimulus to "jump start" the oscillator analysis. If the harmonic balance simulation results in  $V_{port} = V_s$ , which means that the controlled source is zero (short circuit), then the circuit is in the state of free oscillation.

The voltage transfer ratio of the controlled source is given by

$$k = \begin{cases} 1 & \text{at the fundamental frequency} \\ 0 & \text{otherwise} \end{cases}$$

In other words, the stimulus is applied to the circuit at the fundamental frequency only.

### Defining OSCPORT in the Model Block of the Input File

#### Syntax:

```
OSCPORT.name n1 n2 V=Vs R=r X=x;
```

*name* is a character string to identify the port. *n1* and *n2* are nodes.  $V_s$  is an estimate of the voltage across the termination at the fundamental frequency. *r* and *x* specify the real and imaginary parts of the terminating impedance, respectively. By default,  $r = 50$  and  $x = 0$ .



For DC and small-signal analysis, the controlled source in OSCPORT is replaced by a short circuit. In this case OSCPORT is equivalent to PORT.

## Free Oscillation Conditions

If OSCPORT is defined, two large-signal response labels will be automatically created:

$$\text{OSC\_GAIN} = 20 \log_{10}(|V_{\text{port}}/V_s|)$$

$$\text{OSC\_PHASE} = \angle V_{\text{port}} - \angle V_s$$

The conditions for the circuit to achieve free oscillation are

$$\text{OSC\_GAIN} = 0$$

$$\text{OSC\_PHASE} = 0$$

The task of oscillator analysis and/or optimization is to find a combination or combinations of circuit parameter values, frequency and voltage level ( $V_s$ ) such that the free oscillation conditions are met.

## Oscillator Design by Optimization

In oscillator design, the desired oscillation frequency is given. We can try to satisfy the oscillation conditions at the given frequency by optimizing the voltage level and at least one more circuit parameter (such as a tuning capacitor in the resonant subcircuit or the bias voltage on a varactor).

Example:

```

Model
  CAP 1 2 C=?7.5pF?; ! tuning capacitor
  ...
  VS = ?1?;
  OSCPORT.out 10 0 V=VS;
  CIRCUIT;
end

Specification
  HB: FREQ=4GHZ OSC_GAIN=0 OSC_PHASE=0;
end

```



See Chapter 11 for details on optimization and the Specification block.  
See demo62.ckt for an example of oscillator design by optimization.

## Determining the Oscillation Frequency

At the present, OSA90 does not allow you to optimize the frequency. One way to overcome this restriction in order to determine the free oscillation frequency is to invoke another copy of OSA90 as a Datapipe child.

An example of oscillator analysis using this approach is given in demo61.ckt.



## 7

## Nonlinear Elements

<b>BJTU</b>	user-definable bipolar junction transistor model	7-2
<b>DIODE</b>	SPICE diode model	7-3
<b>DIODE0</b>	semiconductor diode model	7-7
<b>DIODEU</b>	user-definable diode model	7-9
<b>FETC</b>	Curtice and Eettenberg FET model	7-10
<b>FETCA</b>	Curtice asymmetrical FET model	7-12
<b>FETM</b>	Materka and Kacprzak FET model	7-14
<b>FETR</b>	Raytheon FET model	7-17
<b>FETT</b>	physics-based Khatibzadeh and Trew FET model	7-19
<b>FETT1</b>	modified physics-based FET model	7-23
<b>FETU1</b>	user-definable FET model 1	7-26
<b>FETU2</b>	user-definable FET model 2	7-27
<b>HBT</b>	heterojunction bipolar transistor model	7-28
<b>HEMTAC</b>	advanced Curtice HEMT model	7-31
<b>HEMTC</b>	Curtice HEMT model	7-34
<b>HEMTG1</b>	high order beta degradation HEMT model	7-37
<b>HEMTG2</b>	double parabolic HEMT model	7-40
<b>KTL</b>	physics-based bias-dependent small-signal FET model	7-43
<b>NLCCS</b>	nonlinear controlled current source	7-48
<b>NLCQ</b>	nonlinear controlled charge source	7-49
<b>NPN</b>	NPN Gummel and Poon bipolar transistor model	7-50
<b>PNP</b>	PNP Gummel and Poon bipolar transistor model	7-52



## 7

# Nonlinear Elements

This chapter describes OSA90's library of nonlinear element models.

The description for each model includes a schematic diagram, the number of nodes, the set of parameters and their default values, an example of the syntax, the model equations and some application notes if appropriate.

The nodes for each element can be divided into two subsets. The first subset represents nodes which must be nonzero for the element to be meaningfully connected (e.g., the element DIODE requires at least one nonzero node). The other set represents nodes which are optional. In the description, the optional nodes are enclosed within square brackets. The default for optional nodes that are not explicitly specified is the ground node.

All the device model parameters are optional except where noted. Model parameters can be specified by constant values, optimization variables and labels. Parameters that are omitted will be assigned the default values.

The nonlinear element library includes

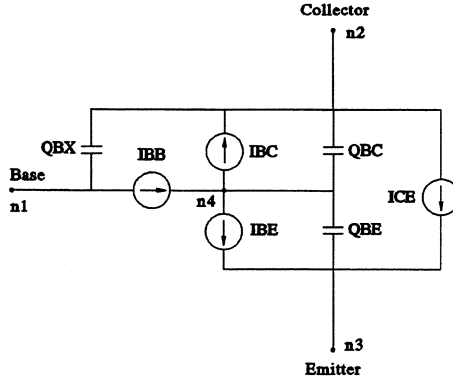
- ▷ built-in device models for FETs, HEMTs, bipolar transistors and diodes, with fixed topology and built-in model equations
- ▷ user-definable device models for FETs, bipolar transistors and diodes, with fixed topology and user-defined model equations
- ▷ nonlinear controlled current and charge sources for creating models with arbitrary topology

☞ See Chapter 6 for further details on creating user-defined models in the Model block.

BJTU

user-definable bipolar junction transistor model

BJTU



Keywords

Defaults

IBE	base-emitter current	0
IBC	base-collector current	0
ICE	collector-emitter current	0
IBB	external base-internal base current	0
QBE	base-emitter capacitor charge	0
QBC	base-collector capacitor charge	0
QBX	external base-collector capacitor charge	0

Form

```
BJTU n1 n2 n3 [n4] IBE = expression1 IBC = expression2 ICE = expression3
                    IBB = expression4 QBE = expression5 QBC = expression6
                    QBX = expression7;
```

To utilize BJTU, you can create a set of model parameters of arbitrary meaning and names, and a set of voltage labels to represent the controlling voltages. The currents and charges are then defined as functions (expression1, ..., expression7) of these model parameters and voltage labels.

The charge sources are represented in the schematic diagram by capacitors to emphasize their capacitive nature. The charge source QBE is directed from n4 toward n3. QBC is directed from n4 toward n2. QBX is directed from n1 toward n2.

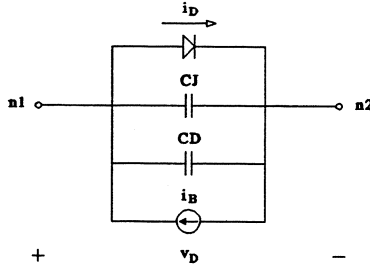
Examples are provided in the include files npn.inc and pnp.inc which implement the Gummel-Poon NPN and PNP models using BJTU.



DIODE

SPICE diode model

DIODE



Keywords

Defaults

IS	saturation current	1.0E-14
RS	ohmic resistance	0
N	emission coefficient	1.0
TT	transit time	0
CJO	zero-bias junction capacitance	0
VJ	junction potential	1.0
M	grading coefficient	0.5
EG	energy gap	1.11
XTI	saturation current temperature exponent	3.0
FC	coefficient for forward-bias depletion capacitance formula	0.5
BV	reverse breakdown voltage (positive number)	$\infty$
IBV	reverse breakdown current (positive number)	1.0E-3
AREA	area factor for device scaling	1.0
GMIN	junction conductance	1.0E-12
TEMP	operating temperature	300°K
T0	nominal (reference) temperature	TEMP

Form

```
DIODE n1 [n2] IS = x1  RS = x2  N = x3  TT = x4  CJO = x5
          VJ = x6  M = x7  EG = x8  XTI = x9  FC = x10
          BV = x11 IBV = x12 AREA = x13 GMIN = x14 TEMP = x15
          T0 = x16;
```

Example

```
DIODE 1 2 IS=6.4e-15 N=1.6 CJO=0.04PF VJ=0.9 FC=0.75;
```

## DIODE

## SPICE diode model

## DIODE

## Model Equations

DIODE is a nonlinear diode model implementing the SPICE model equations published in [1].

The diode model in Version 2.5 and earlier versions of OSA90 is preserved under the name DIODE0. It differs from DIODE in the modeling of breakdown, temperature dependence and area scaling.

## Diode Current

$$i_D = \begin{cases} IS \cdot [\exp(v_D/v_T) - 1] + v_D \cdot GMIN & \text{for } v_D > -v_B \\ -IS \cdot \exp(-(v_D+v_B)/v_T) + v_D \cdot GMIN & \text{for } v_D \leq -v_B \end{cases}$$

where  $v_T = N \cdot k \cdot \text{TEMP} / q$ ,  $k$  is the Boltzmann constant,  $q$  is the electron charge and  $IS$ ,  $N$ ,  $\text{TEMP}$  and  $GMIN$  are model parameters.

SPICE defines  $GMIN$  as a small junction conductance to aid convergence. OSA90 allows  $GMIN$  to be set to zero for an ideal diode (SPICE does not permit a zero value for  $GMIN$ ).

The breakdown voltage  $v_B$  is determined by the model parameters  $BV$  and  $IBV$ :

$$v_B = \begin{cases} BV - v_T \cdot \log((IBV - BV \cdot GMIN) / IS) & \text{if } IBV > IS + BV \cdot GMIN \\ BV & \text{otherwise} \end{cases}$$

## Stored Charge

$$Q_s = TT \cdot i_D$$

where  $TT$  is the model parameter representing the transit time.

The equivalent nonlinear capacitance is given by

$$C_s = \begin{cases} TT \cdot [IS \cdot \exp(v_D/v_T) / v_T + GMIN] & \text{for } v_D > -v_B \\ TT \cdot [IS \cdot \exp(-(v_D+v_B)/v_T) / v_T + GMIN] & \text{for } v_D \leq -v_B \end{cases}$$

Stored charge is applicable to  $pn$ -junction diodes. For Schottky barrier diodes  $TT = 0$ .

## DIODE

## SPICE diode model

## DIODE

## Space Charge

$$Q_D = \begin{cases} C_{J0} \cdot V_J / (1 - M) \cdot [1 - (1 - v_D / V_J)^{1-M}] & \text{for } v_D < FC \cdot V_J \\ C_{J0} \cdot [F1 + (0.5 \cdot M \cdot v_D^2 / V_J + F3 \cdot v_D + F4) / F2] & \text{for } v_D \geq FC \cdot V_J \end{cases}$$

where  $C_{J0}$ ,  $V_J$ ,  $FC$  and  $M$  are model parameters and

$$F1 = V_J / (1 - M) \cdot [1 - (1 - FC)^{1-M}]$$

$$F2 = (1 - FC)^{1+M}$$

$$F3 = 1 - FC \cdot (1 + M)$$

$$F4 = FC \cdot V_J \cdot [FC \cdot (1 + M/2) - 1]$$

The equivalent nonlinear capacitance is given by

$$C_D = \begin{cases} C_{J0} \cdot (1 - v_D / V_J)^{-M} & \text{for } v_D < FC \cdot V_J \\ C_{J0} / F2 \cdot (M \cdot v_D / V_J + F3) & \text{for } v_D \geq FC \cdot V_J \end{cases}$$

## Area Scaling

$$IS = IS \cdot AREA$$

$$C_{J0} = C_{J0} \cdot AREA$$

$$RS = RS / AREA$$

## DIODE

## SPICE diode model

## DIODE

## Temperature Dependence

Temperature dependence as defined by SPICE is taken into account if T0 and TEMP are given different values.

T0 represents the nominal (reference) temperature at which all the other model parameters are assumed to have been measured (as specified in SPICE by TNOM in the .OPTIONS card). TEMP represents the operating temperature (as specified in SPICE using the .TEMP card).

The temperature dependence of IS is given by

$$IS(TEMP) = IS(T0) \cdot t^{XTI/N} \cdot \exp[-q \cdot EG / (k \cdot TEMP) \cdot (1 - t)]$$

where XTI, N and EG are model parameters and  $t = TEMP / T0$ .

According to SPICE, you should specify XTI = 3 for *pn*-junction diodes and XTI = 2 for Schottky barrier diodes (SBD). Also, EG should be 1.11 for Si, 0.69 for SBD and 0.67 for Ge.

The temperature dependence of VJ is defined as

$$VJ(TEMP) = VJ(T0) \cdot t - 2 \cdot k \cdot TEMP / q \cdot \log(t^{1.5}) - [t \cdot E_g(T0) - E_g(TEMP)]$$

where  $E_g(T) = 1.16 - 7.02 \times 10^{-4} \cdot T^2 / (1108 + T)$ .

The temperature dependence of CJO is given by

$$CJO(TEMP) = CJO(T0) \cdot \{1 + M \cdot [4 \times 10^{-4} \cdot (TEMP - T0) - VJ(TEMP) / VJ(T0) + 1]\}$$

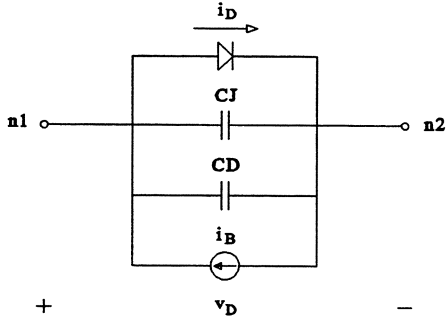
## Reference

- [1] P. Antognetti and G. Massobrio, *Semiconductor Device Modeling with SPICE*. New York, NY: McGraw-Hill, 1988.

DIODE0

semiconductor diode model

DIODE0



Keywords

Defaults

IS	saturation current	1.0E-14
N	emission coefficient	1.0
TEMP	temperature	298°K
VB	breakdown voltage	-∞
IB0	breakdown current at $v_D = VB - 1$	0
EB	breakdown power-law exponent	10
CJ0	zero-bias junction capacitance	0
VJ	junction potential	0.8
FC	coefficient for forward bias in capacitance equation	0.5
EJ	junction capacitance exponent	0.5
CD0	zero-bias diffusion capacitance	0

Form

DIODE0 n1 [n2] IS = x1 N = x2 TEMP = x3 VB = x4 IB0 = x5 EB = x6  
 CJ0 = x7 VJ = x8 FC = x9 EJ = x10 CD0 = x11;

Example

DIODE0 1 2 IS=6.4e-15 N=1.6 CJ0=0.04PF VJ=0.9 FC=0.75;

## DIODE0

## semiconductor diode model

## DIODE0

## Model Equations

DIODE0 is a generic semiconductor diode model suitable for both PN junction diodes and Schottky barrier diodes [1]. From Version 3.0 of OSA90, a fully SPICE compatible model is provided under the name DIODE. The model in earlier versions is renamed as DIODE0. Use of the new DIODE model is highly recommended.

All components of the equivalent circuit are assumed to be functions of the diode voltage  $v_D$  indicated in the diagram. The nonlinear model equations are as follows.

$$i_D = IS \cdot [ \exp((v_D \cdot q)/(N \cdot k \cdot TEMP)) - 1 ]$$

where  $q$  is the electron charge and  $k$  is the Boltzmann constant.  $IS$ ,  $N$  and  $TEMP$  are model parameters.

$$i_B = \begin{cases} IBO \cdot (VB - v_D)^{EB} & \text{for } v_D < VB \\ 0 & \text{for } v_D \geq VB \end{cases}$$

where  $IBO$ ,  $VB$  and  $EB$  are model parameters.

$$C_J = \begin{cases} CJO \cdot (1 - v_D/VJ)^{-EJ} & \text{for } v_D < FC \cdot VJ \\ CJO \cdot (1 - FC)^{-EJ} & \text{for } v_D \geq FC \cdot VJ \end{cases}$$

where  $CJO$ ,  $VJ$ ,  $EJ$  and  $FC$  are model parameters.

$$C_D = CDO \cdot \exp((v_D \cdot q)/(N \cdot k \cdot TEMP))$$

where  $q$  is the electron charge and  $k$  is the Boltzmann constant.  $CDO$ ,  $N$  and  $TEMP$  are model parameters.

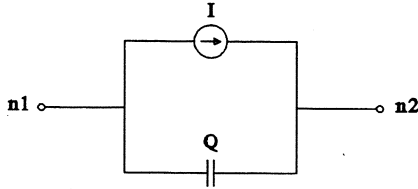
## Reference

- [1] J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design*. New York, NY: Van Nostrand Reinhold, 1983, pp. 309-312.

DIODEU

user-definable diode model

DIODEU



### Keywords

I diode current  
 Q diode capacitor charge

### Defaults

0  
 0

### Form

```
DIODEU n1 [n2] I = expression1 Q = expression2;
```

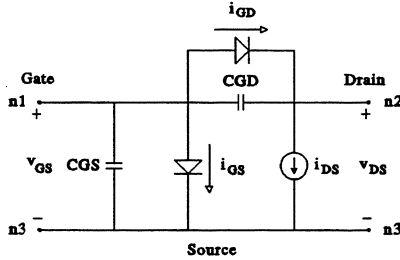
To utilize DIODEU, you can create a set of model parameters of arbitrary names and meaning, and one or more voltage labels to represent the controlling voltages. The diode current and charge are then defined as functions (*expression1* and *expression2*) of these model parameters and voltage labels.

The nonlinear charge source is represented in the schematic diagram by a capacitor to emphasize its capacitive nature. The charge source *Q* is directed from *n1* toward *n2*.

FETC

Curtice and Ettenberg FET model

FETC



Keywords

Defaults

A0	coefficient of the cubic formula for $i_{DS}$	0
A1	coefficient of the cubic formula for $i_{DS}$	0
A2	coefficient of the cubic formula for $i_{DS}$	0
A3	coefficient of the cubic formula for $i_{DS}$	0
GAMMA	hyperbolic tangent function parameter	0
BETA	coefficient for pinch-off change with respect to $v_{DS}$	0
VDS0	$v_{DS}$ at which A0, A1, A2 and A3 were evaluated	0
TAU	transit time under gate	0
IS	gate junction saturation current	1.0E-14
N	gate-drain and gate-source emission coefficient	1.0
CGD0	zero-bias gate-drain junction capacitance	0
CGS0	zero-bias gate-source junction capacitance	0
FC	coefficient for forward bias in capacitance equation	0.5
GMIN	linear conductance associated with the Schottky junctions	1.0E-12
VBI	built-in gate potential	0.8
VBR	gate reverse bias breakdown voltage	+∞
TEMP	temperature	298°K

Form

```
FETC n1 n2 n3 A0 = x1 A1 = x2 A2 = x3 A3 = x4 GAMMA = x5
      BETA = x6 VDS0 = x7 TAU = x8 IS = x9 N = x10
      CGD0 = x11 CGS0 = x12 FC = x13 GMIN = x14 VBI = x15
      VBR = x16 TEMP = x17;
```

Example

```
FETC 1 2 3 A0=0.06 A1=0.1 A2=0.06 GAMMA=2 VDS0=2 IS=3nA;
```



## FETC

## Curtice and Ettenberg FET model

## FETC

## Model Equations

FETC implements the cubic symmetrical Curtice and Ettenberg model based on [1].

All components of the equivalent circuit are assumed to be functions of the gate and drain voltages  $v_{GS}$  and  $v_{DS}$  as shown in the diagram. The nonlinear model equations are as follows.

$$i_{DS} = \begin{cases} (A0+A1 \cdot v1+A2 \cdot v1^2+A3 \cdot v1^3) \cdot \tanh(GAMMA \cdot v_{DS}) & \text{for } v_{DS} \geq 0 \\ 0 & \text{for } v_{DS} < 0 \end{cases}$$

where

$$v1(t) = v_{GS}(t - \text{TAU}) \cdot (1 + \text{BETA} \cdot (\text{VDS0} - v_{DS}(t)))$$

and A0, A1, A2, A3, BETA, GAMMA, TAU and VDS0 are model parameters.

$$i_{GS} = \begin{cases} IS \cdot [\exp(v_{GS}/(N \cdot v_t)) - 1] + \text{GMIN} \cdot v_{GS} & \text{for } v_{GS} \geq -5N \cdot v_t \\ -IS + \text{GMIN} \cdot v_{GS} & \text{for } -\text{VBR} + 50v_t < v_{GS} < -5N \cdot v_t \\ -IS \cdot [\exp(-(v_{GS} + \text{VBR})/v_t) + 1] + \text{GMIN} \cdot v_{GS} & \text{for } v_{GS} \leq -\text{VBR} + 50v_t \end{cases}$$

where  $v_t = k \cdot \text{TEMP} / q$  is the thermal voltage. IS, N, GMIN, VBR and TEMP are model parameters.

The diode current  $i_{GD}$  is similarly defined.

$$C_{GS} = \begin{cases} \text{CGS0} \cdot (1 - v_{GS}/\text{VBI})^{-0.5} & \text{for } v_{GS} < \text{FC} \cdot \text{VBI} \\ \text{CGS0} \cdot (1 - \text{FC})^{-0.5} \cdot (1 + (v_{GS} - \text{FC} \cdot \text{VBI}) / (2 \cdot \text{VBI} \cdot (1 - \text{FC}))) & \text{otherwise} \end{cases}$$

where CGS0, VBI and FC are model parameters.

The gate-drain capacitance  $C_{GD}$  is similarly defined.

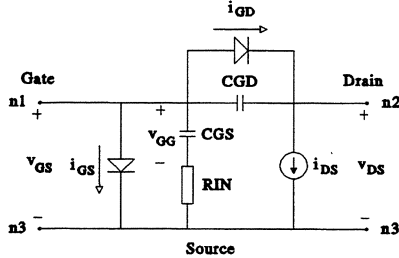
## Reference

- [1] W.R. Curtice and M. Ettenberg, "A nonlinear GaAs FET model for use in the design of output circuits for power amplifiers", *IEEE Trans. Microwave Theory Tech.*, vol. MTT-33, 1985, pp. 1383-1394.

FETCA

Curtice asymmetrical FET model

FETCA



Keywords

Defaults

A0	coefficient of the cubic formula for $i_{DS}$	0
A1	coefficient of the cubic formula for $i_{DS}$	0
A2	coefficient of the cubic formula for $i_{DS}$	0
A3	coefficient of the cubic formula for $i_{DS}$	0
GAMMA	hyperbolic tangent function parameter	0
BETA	coefficient for pinch-off change with respect to $v_{DS}$	0
VDS0	$v_{DS}$ at which A0, A1, A2 and A3 were evaluated	0
TAU	transit time under gate	0
IS	gate junction saturation current	1.0E-14
N	gate-drain and gate-source emission coefficient	1.0
CGD0	zero-bias gate-drain junction capacitance	0
CGS0	zero-bias gate-source junction capacitance	0
FC	coefficient for forward bias in capacitance equation	0.5
GMIN	linear conductance associated with the Schottky junctions	1.0E-12
VBI	built-in gate potential	0.8
VBR	gate reverse bias breakdown voltage	+∞
TEMP	temperature	298°K
RIN	intrinsic channel gate-source resistance	1.0

Form

```
FETCA  n1 n2 n3  A0 = x1    A1 = x2    A2 = x3    A3 = x4    GAMMA = x5
          BETA = x6    VDS0 = x7    TAU = x8    IS = x9    N = x10
          CGD0 = x11  CGS0 = x12  FC = x13  GMIN = x14  VBI = x15
          VBR = x16  TEMP = x17  RIN = x18;
```

Example

```
FETCA  1 2 3  A0=0.06  A1=0.1  A2=0.06  GAMMA=2  VDS0=2  IS=3nA  RIN=2.5;
```

FETCA

Curtice asymmetrical FET model

FETCA

## Model Equations

FETCA implements the cubic asymmetrical Curtice and Ettenberg model based on [1].

The nonlinear components of the equivalent circuit are functions of the voltages  $v_{GS}$ ,  $v_{GG}$  and  $v_{DS}$  as shown in the diagram. The nonlinear model equations are as follows.

$$i_{DS} = \begin{cases} (A0+A1 \cdot v1+A2 \cdot v1^2+A3 \cdot v1^3) \cdot \tanh(GAMMA \cdot v_{DS}) & \text{for } v_{DS} \geq 0 \\ 0 & \text{for } v_{DS} < 0 \end{cases}$$

where

$$v1(t) = v_{GS}(t - \text{TAU}) \cdot (1 + \text{BETA} \cdot (\text{VDS0} - v_{DS}(t)))$$

and A0, A1, A2, A3, BETA, GAMMA, TAU and VDS0 are model parameters.

$$i_{GS} = \begin{cases} IS \cdot [\exp(v_{GS}/(N \cdot v_t)) - 1] + \text{GMIN} \cdot v_{GS} & \text{for } v_{GS} \geq -5N \cdot v_t \\ -IS + \text{GMIN} \cdot v_{GS} & \text{for } -\text{VBR} + 50v_t < v_{GS} < -5N \cdot v_t \\ -IS \cdot [\exp(-(v_{GS} + \text{VBR})/v_t) + 1] + \text{GMIN} \cdot v_{GS} & \text{for } v_{GS} \leq -\text{VBR} + 50v_t \end{cases}$$

where  $v_t = k \cdot \text{TEMP} / q$  is the thermal voltage. IS, N, GMIN, VBR and TEMP are model parameters.

The diode current  $i_{GD}$  is similarly defined.

$$C_{GS} = \begin{cases} \text{CGS0} \cdot (1 - v_{GG}/\text{VBI})^{-0.5} & \text{for } v_{GG} < \text{FC} \cdot \text{VBI} \\ \text{CGS0} \cdot (1 - \text{FC})^{-0.5} \cdot (1 + (v_{GG} - \text{FC} \cdot \text{VBI}) / (2 \cdot \text{VBI} \cdot (1 - \text{FC}))) & \text{otherwise} \end{cases}$$

where CGS0, VBI and FC are model parameters.

The gate-drain capacitance  $C_{GD}$  is similarly defined.

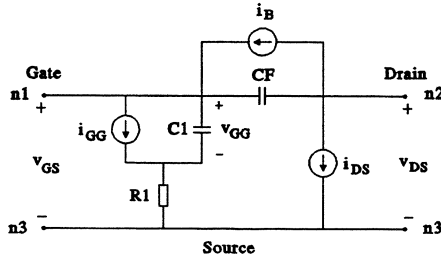
## Reference

- [1] W.R. Curtice and M. Ettenberg, "A nonlinear GaAs FET model for use in the design of output circuits for power amplifiers", *IEEE Trans. Microwave Theory Tech.*, vol. MTT-33, 1985, pp. 1383-1394.

FETM

Materka and Kacprzak FET model

FETM



Keywords

Defaults

IDSS	drain saturation current for $v_{GG}=0$	6.0E-2
VP0	pinch-off voltage for $v_{DS}=0$	-2.0
VBC	channel breakdown voltage	$+\infty$
SL	slope of drain current in the linear region for $v_{GG}=0$	6.0E-2
ALPHAB	slope factor of breakdown current	0
TAU	channel transit-time delay	0
IB0	channel breakdown current for $v_{DS} - v_{GS} = VBC$	0
KG	parameter describing drain current in the linear region	0
ALPHAG	slope factor of gate conduction current	38.696
E	parameter describing drain saturation current	2.0
IG0	saturation current of gate Schottky barrier	0
KR	voltage slope factor of intrinsic channel resistance	0
GAMMA	voltage-slope parameter of pinch-off voltage	0
KE	parameter describing drain saturation current	0
R10	intrinsic channel resistance for zero gate voltage	3.5
SS	slope of drain current in the saturation region for $v_{GG}=0$	0
C10	gate Schottky-barrier capacitance for $v_{GG}=0$	4.0E-4
K1	voltage-slope parameter of gate capacitance	1.25
C1S	linear parasitic gate capacitance	0
CF0	feedback capacitance for $v_{DS}=v_{GS}$	2.0E-5
KF	voltage-slope parameter of feedback capacitance	0

Form

FETM	n1	n2	n3	IDSS = x1	VP0 = x2	VBC = x3	SL = x4	ALPHAB = x5
				TAU = x6	IB0 = x7	KG = x8	ALPHAG = x9	E = x10
				IG0 = x11	KR = x12	GAMMA = x13	KE = x14	R10 = x15
				SS = x16	C10 = x17	K1 = x18	C1S = x19	CF0 = x20
				KF = x21;				

FETM

Materka and Kacprzak FET model

FETM

## Example

```
FETM 1 2 3 IDSS=6.0E-2 VPO=-1.9 SL=0.0676 R10=3.5
      C10=0.42PF K1=1.282 CF0=0.02PF;
```

## Model Equations

FETM implements the Materka and Kacprzak model based on [1].

All components of the equivalent circuit are assumed to be functions of voltages  $v_{GS}$ ,  $v_{GG}$  and  $v_{DS}$  as shown in the diagram. The nonlinear model equations are as follows.

$$C_1 = \begin{cases} C10 \cdot (1 - K1 \cdot v_{GG})^{-0.5} + C1S & \text{for } K1 \cdot v_{GG} < 0.8 \\ C10 \cdot \text{sqrt}(5) + C1S & \text{for } K1 \cdot v_{GG} \geq 0.8 \end{cases}$$

where C10, K1 and C1S are model parameters.

$$C_F = \begin{cases} CF0 \cdot (1 - KF \cdot (v_{GS} - v_{DS}))^{-0.5} & \text{for } KF \cdot (v_{GS} - v_{DS}) < 0.8 \\ CF0 \cdot \text{sqrt}(5) & \text{for } KF \cdot (v_{GS} - v_{DS}) \geq 0.8 \end{cases}$$

where CF0 and KF are model parameters.

$$R_1 = \begin{cases} R10 \cdot (1 - KR \cdot v_{GG}) & \text{for } KR \cdot v_{GG} < 1.0 \\ 0 & \text{for } KR \cdot v_{GG} \geq 1.0 \end{cases}$$

where R10 and KR are model parameters.

$$i_{GG} = IGO \cdot [\exp(\text{ALPHAG} \cdot v_{GG}) - 1]$$

where IGO and ALPHAG are model parameters.

$$i_B = IB0 \cdot \exp(\text{ALPHAB} \cdot (v_{DS} - v_{GS} - \text{VBC}))$$

where IB0, ALPHAB and VBC are model parameters.

$$i_{DS} = \text{IDSS} \cdot [1 - v_{GG}(t - \text{TAU}) / (VPO + \text{GAMMA} \cdot v_{DS})]^{(E + KE \cdot v_{GG}(t - \text{TAU}))} \cdot \frac{\tanh[SL \cdot v_{DS} / (\text{IDSS} \cdot (1 - KG \cdot v_{GG}(t - \text{TAU})))]}{1 + SS \cdot v_{DS} / \text{IDSS}}$$

where IDSS, TAU, VPO, GAMMA, E, KE, SL, KG and SS are model parameters.

FETM

Materka and Kacprzak FET model

FETM

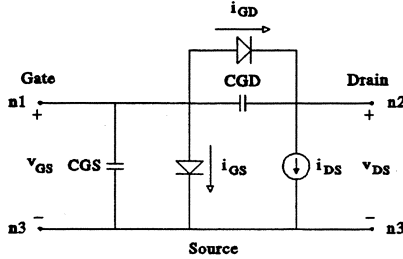
## Reference

- [1] A. Materka and T. Kacprzak, "Computer calculation of large-signal GaAs FET amplifier characteristics", *IEEE Trans. Microwave Theory Tech.*, vol. MTT-33, 1985, pp. 129-135.

FETR

Raytheon FET model

FETR



Keywords

Defaults

ALPHA	hyperbolic tangent function parameter	0
BETA	transconductance parameter	0
LAMBDA	channel length modulation parameter	0
THETA	transconductance parameter for large $v_{GS}$	0
VTO	threshold voltage	-2.0
TAU	transit time under gate	0
IS	gate junction saturation current	1.0E-14
N	gate-drain and gate-source emission coefficient	1.0
CGD0	zero bias gate-drain junction capacitance	0
CGS0	zero bias gate-source junction capacitance	0
FC	coefficient for forward bias in the capacitance equation	0.5
GMIN	linear conductance associated with the Schottky junctions	1.0E-12
VBI	built-in gate potential	0.8
VBR	gate reverse bias breakdown voltage	+∞
TEMP	temperature	298°K

Form

```
FETR n1 n2 n3 ALPHA = x1  BETA = x2  LAMBDA = x3  THETA = x4
      VTO = x5  TAU = x6  IS = x7  N = x8
      CGS0 = x9  CGD0 = x10  FC = x11  GMIN = x12
      VBI = x13  VBR = x14  TEMP = x15;
```

Example

```
FETR 1 2 3 BETA=0.02 VTO=-1.2 IS=3.7nA;
```

FETR

Raytheon FET model

FETR

### Model Equations

FETR implements the Raytheon model based on the work of Statz *et al.* [1].

All components of the equivalent circuit are assumed to be functions of the gate and drain voltages  $v_{GS}$  and  $v_{DS}$  as shown in the diagram. The nonlinear model equations are as follows.

$$i_{DS} = \begin{cases} xx1 \cdot xx2 \cdot (1 - (1 - \text{ALPHA} \cdot v_{DS}/3)^3) & \text{for } 0 < v_{DS} < 3/\text{ALPHA} \\ xx1 \cdot xx2 & \text{for } v_{DS} \geq 3/\text{ALPHA} \end{cases}$$

where

$$xx1 = (\text{BETA} \cdot [v_{GS}(t - \text{TAU}) - \text{VTO}]^2) / (1 + \text{THETA} \cdot [v_{GS}(t - \text{TAU}) - \text{VTO}])$$

$$xx2 = 1 + \text{LAMBDA} \cdot v_{DS}$$

ALPHA, BETA, LAMBDA, THETA, TAU and VTO are model parameters.

$$i_{GS} = \begin{cases} IS \cdot [\exp(v_{GS}/(N \cdot v_t)) - 1] + \text{GMIN} \cdot v_{GS} & \text{for } v_{GS} \geq -5N \cdot v_t \\ -IS + \text{GMIN} \cdot v_{GS} & \text{for } -\text{VBR} + 50v_t < v_{GS} < -5N \cdot v_t \\ -IS \cdot [\exp(-(v_{GS} + \text{VBR})/v_t) + 1] + \text{GMIN} \cdot v_{GS} & \text{for } v_{GS} \leq -\text{VBR} + 50v_t \end{cases}$$

where  $v_t = k \cdot \text{TEMP}/q$  is the thermal voltage. IS, N, GMIN, VBR and TEMP are model parameters.

The diode current  $i_{GD}$  is similarly defined.

$$C_{GS} = \begin{cases} \text{CGS0} \cdot (1 - v_{GS}/\text{VBI})^{-0.5} & \text{for } v_{GS} < \text{FC} \cdot \text{VBI} \\ \text{CGS0} \cdot (1 - \text{FC})^{-0.5} \cdot (1 + (v_{GS} - \text{FC} \cdot \text{VBI}) / (2 \cdot \text{VBI} \cdot (1 - \text{FC}))) & \text{otherwise} \end{cases}$$

where CGS0, VBI and FC are model parameters.

The gate-drain capacitance  $C_{GD}$  is similarly defined.

### Reference

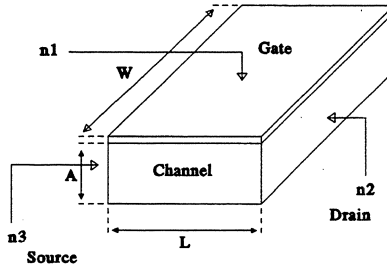
- [1] H. Statz, P. Newman, I.W. Smith, R.A. Pucel and H.A. Haus, "GaAs FET device and circuit simulation in SPICE", *IEEE Trans. Electron Devices*, vol. ED-34, 1987, pp. 160-169.



FETT

physics-based Khatibzadeh and Trew FET model

FETT



Keywords

Defaults

L	gate length in m	0.56E-6
W	gate width in m	1.2E-3
A	channel thickness in m	0.35E-6
U0	low-field mobility in $m^2/(Vns)$	4.0E-10
D	high-field diffusion coefficient in $m^2/ns$	1.0E-12
EPSR	relative dielectric constant	12.5
EC	critical electric field in V/m	3.75E5
VS	saturation electron velocity in m/ns	1.5E-4
VBI	built-in potential in V	0.7
ND	doping density (for uniform doping) in $m^{-3}$	7.5E22
LAMBDA	parameter for the transition function in m	1.0E-8
ALPHA	exponent for calculating the average mobility	0.5
TAU	time delay in ns	0.0
ND1	parameter for doping profile in $m^{-3}$	1.0E22
ND2	parameter for doping profile in $m^{-3}$	1.0E22
Y0	parameter for doping profile in m	1.0E-7
YNOM	parameter for doping profile in m	$+\infty$
YNOM1	parameter for doping profile in m	$+\infty$
AA	parameter for electron velocity curve in V/m	1.0E6
BB	parameter for electron velocity curve	1
CC	parameter for electron velocity curve	1
DD	parameter for electron velocity curve	0

**FETT physics-based Khatibzadeh and Trew FET model FETT**

**Form**

FETT n1 n2 n3 L = x1 W = x2 A = x3 U0 = x4  
 D = x5 EPSR = x6 EC = x7 VS = x8  
 VBI = x9 ND = x10 LAMBDA = x11 ALPHA = x12  
 TAU = x13 ND1 = x14 ND2 = x15 Y0 = x16  
 YNOM = x17 YNOM1 = x18 AA = x19 BB = x20  
 CC = x21 DD = x22;

**Example**

FETT 1 2 3 L=0.42UM W=600UM A=0.3UM VBI=0.7 ND=1.0E23 TAU=3PS;

**Model Equations**

FETT implements the physics-based Khatibzadeh and Trew analytical model based on [1]. It is dynamically integrated into the harmonic balance simulation and optimization as described by Bandler, Zhang and Cai [2].

**Doping Profile**

FETT accommodates uniform and nonuniform doping profiles. Three types of doping profiles, namely uniform, quadratic and piecewise (Khatibzadeh [3]), are available.

Let  $N(y)$  represent the doping density function, where  $y$  is the depth under the FET gate.

If the model parameter YNOM is not specified, then uniform doping is assumed:

$$N(y) = ND$$

In this case, the parameters ND1, ND2, Y0, YNOM and YNOM1 are ignored.

If YNOM is specified but YNOM1 is not, then quadratic doping approximation is used:

$$N(y) = ND \cdot \exp\{-0.5 \cdot [(y - Y0) / YNOM]^2\}$$

In this case, ND1, ND2 and YNOM1 are ignored.

If both YNOM and YNOM1 are specified, then piecewise doping approximation is used:

$$N(y) = \begin{cases} ND & \text{if } y \leq Y0 \\ ND1 \cdot \exp\{(Y0-y)/YNOM\} + ND2 \cdot \exp\{(Y0-y)/YNOM1\} & \text{otherwise} \end{cases}$$

## FETT physics-based Khatibzadeh and Trew FET model FETT

### Electron Velocity - Electric Field Curve

The electron velocity as a function of the electric field is given by

$$v(E) = (1 - DD) \cdot v_1(E) + DD \cdot v_2(E)$$

where  $E$  is the electric field, and  $DD$  is a model parameter.

$v_1(E)$  is the formula originally used by Khatibzadeh and Trew [1]:

$$v_1(E) = \begin{cases} VS & \text{if } E > 2 \cdot EC \\ U0 \cdot E - VS \cdot E^2 / (4 \cdot EC^2) & \text{if } E \leq 2 \cdot EC \end{cases}$$

where  $VS$ ,  $U0$  and  $EC$  are model parameters.

The function  $v_2(E)$  is added to model the overshooting behaviour as experimentally observed:

$$v_2(E) = VS \cdot (BB/CC) \cdot \{(E/AA - CC) / [(E/AA)^4 + BB] + CC/BB\}$$

where  $VS$ ,  $AA$ ,  $BB$  and  $CC$  are model parameters.  $AA$  can be used to adjust the location of the overshoot.  $BB$  and  $CC$  can be used to adjust the sharpness of the curve peak. A larger value of the ratio  $BB/CC$  will lead to a sharper peak.

The parameter  $DD$  serves as a weighting factor for combining  $v_1(E)$  and  $v_2(E)$ . Its value should be between 0 and 1.

### Transition Function

The transition function models the transition of the doping profile from almost zero near the gate ( $y \approx 0$ ) to the background doping concentration in the channel ( $y \approx A$ ):

$$T(d(x), y) = 1 - \frac{1}{1 + \exp\{(y - d(x))/LAMBDA\}}$$

where  $d(x)$  is the effective depletion-layer width calculated internally by the model, and  $LAMBDA$  is a model parameter whose value must be of the order of the Debye length.

### ALPHA

$ALPHA$  is an empirical model parameter used to calculate the weighted average mobility for the evaluation of the conduction currents [3]. Setting  $ALPHA = 0.5$  is experimentally found to give good results for the current level.

## FETT physics-based Khatibzadeh and Trew FET model FETT

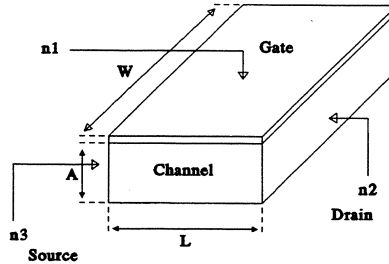
## References

- [1] M.A. Khatibzadeh and R.J. Trew, "A large-signal, analytic model for the GaAs MESFET", *IEEE Trans. Microwave Theory Tech.*, vol. 36, 1988, pp. 231-238.
- [2] J.W. Bandler, Q.J. Zhang and Q. Cai, "Nonlinear circuit optimization with dynamically integrated physical device models", *IEEE MTT-S Int. Microwave Symp. Dig.* (Dallas, TX), 1990, pp. 303-306.
- [3] M.A. Khatibzadeh, "Large-signal modeling of gallium-arsenide field effect transistors", Ph.D. dissertation, North Carolina State University, Raleigh, NC, 1987.

FETT1

modified physics-based FET model

FETT1



## Keywords

		Defaults
L	gate length in m	0.56E-6
W	gate width in m	1.2E-3
A	channel thickness in m	0.35E-6
U0	low-field mobility in $m^2/(Vns)$	4.0E-10
D	high-field diffusion coefficient in $m^2/ns$	1.0E-12
EPSR	relative dielectric constant	12.5
EC	critical electric field in V/m	3.75E5
VS	saturation electron velocity in m/ns	1.5E-4
VBI	built-in potential in V	0.7
ND	doping density (for uniform doping) in $m^{-3}$	7.5E22
LAMBDA	parameter for the transition function in m	1.0E-8
ALPHA	exponent for calculating the average mobility	0.5
TAU	time delay in ns	0.0
AA	parameter for electron velocity curve in V/m	1.0E6
BB	parameter for electron velocity curve	1
CC	parameter for electron velocity curve	1
DD	parameter for electron velocity curve	0

## Form

```

FETT  n1 n2 n3  L = x1      W = x2      A = x3      U0 = x4
        D = x5      EPSR = x6    EC = x7      VS = x8
        VBI = x9    ND = x10    LAMBDA = x11  ALPHA = x12
        TAU = x13  AA = x14    BB = x15    CC = x16
        DD = x17;

```

FETT1

modified physics-based FET model

FETT1

## Example

```
FETT1 1 2 3 L=0.42UM W=600UM A=0.3UM VBI=0.7 ND=1.0E23
      TAU=3PS;
```

## Model Equations

FETT1 implements the physics-based analytical model by Khatibzadeh and Trew [1] and OSA's modifications to improve the model efficiency for uniform doping profiles. Under suitable conditions (uniform doping), FETT1 can be evaluated much faster than FETT. However, it cannot be used for nonuniform doping profiles (use FETT instead).

The model equations are dynamically integrated into the harmonic balance simulation and optimization as described by Bandler, Zhang and Cai [2].

### Doping Density

FETT1 is suitable for uniform doping profile only. The doping density is specified by the model parameter ND.

### Electron Velocity - Electric Field Curve

The electron velocity as a function of the electric field is given by

$$v(E) = (1 - DD) \cdot v_1(E) + DD \cdot v_2(E)$$

where E is the electric field, and DD is a model parameter.

$v_1(E)$  is the formula originally used by Khatibzadeh and Trew [1]:

$$v_1(E) = \begin{cases} VS & \text{if } E > 2 \cdot EC \\ U0 \cdot E - VS \cdot E^2 / (4 \cdot EC^2) & \text{if } E \leq 2 \cdot EC \end{cases}$$

where VS, U0 and EC are model parameters.

The function  $v_2(E)$  is added to model the overshooting behaviour as experimentally observed:

$$v_2(E) = VS \cdot (BB/CC) \cdot \{ (E/AA - CC) / [(E/AA)^4 + BB] + CC/BB \}$$

where VS, AA, BB and CC are model parameters. AA can be used to adjust the location of the overshoot. BB and CC can be used to adjust the sharpness of the curve peak. A larger value of the ratio BB/CC will lead to a sharper peak.

## FETT1

## modified physics-based FET model

## FETT1

The parameter DD serves as a weighting factor for combining  $v_1(E)$  and  $v_2(E)$ . Its value should be between 0 and 1.

**Transition Function**

The transition function models the transition of the doping profile from almost zero near the gate ( $y \approx 0$ ) to the background doping concentration in the channel ( $y \approx A$ ). For uniform doping, FETT1 approximates the transition function by

$$T(d(x), y) = \begin{cases} 0.5 \cdot (1 + \sin(\pi(y-d(x))/(6 \cdot \text{LAMBDA}))) & \text{if } d(x) - 3 \cdot \text{LAMBDA} < y < d(x) + 3 \cdot \text{LAMBDA} \\ 0 & \text{if } y < d(x) - 3 \cdot \text{LAMBDA} \\ 1 & \text{if } y > d(x) + 3 \cdot \text{LAMBDA} \end{cases}$$

where  $d(x)$  is the effective depletion-layer width calculated internally by the model, and LAMBDA is a model parameter whose value must be of the order of the Debye length (for example, LAMBDA = 0.01UM).

Using this formula, the integrations involved in the calculation of currents and charges at the gate, drain and source electrodes are substantially simplified.

**ALPHA**

ALPHA is an empirical model parameter used to calculate the weighted average mobility for the evaluation of the conduction currents [3]. Setting ALPHA = 0.5 is experimentally found to give good results.

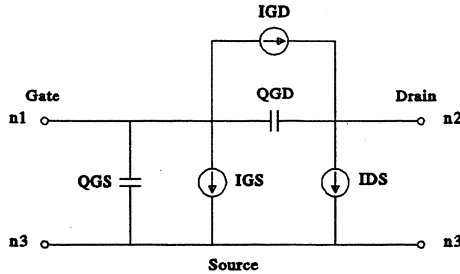
**References**

- [1] M.A. Khatibzadeh and R.J. Trew, "A large-signal, analytic model for the GaAs MESFET", *IEEE Trans. Microwave Theory Tech.*, vol. 36, 1988, pp. 231-238.
- [2] J.W. Bandler, Q.J. Zhang and Q. Cai, "Nonlinear circuit optimization with dynamically integrated physical device models", *IEEE MTT-S Int. Microwave Symp. Dig.* (Dallas, TX), 1990, pp. 303-306.
- [3] M.A. Khatibzadeh, "Large-signal modeling of gallium-arsenide field effect transistors", Ph.D. dissertation, North Carolina State University, Raleigh, NC, 1987.

FETU1

user-definable FET model 1

FETU1



Keywords

Defaults

IGS	gate-source current	0
IDS	drain-source current	0
IGD	gate-drain current	0
QGS	gate-source capacitor charge	0
QGD	gate-drain capacitor charge	0

Form

```
FETU1 n1 n2 [n3] IGS = expression1 IDS = expression2
          IGD = expression3 QGS = expression4
          QGD = expression5;
```

To utilize FETU1, you can create a set of model parameters of arbitrary meaning and names, and a set of voltage labels to represent the controlling voltages. The currents and charges are then defined as functions (expression1, ..., expression5) of these model parameters and voltage labels.

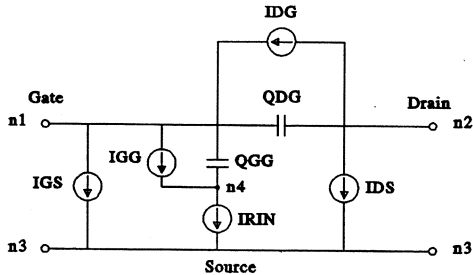
The nonlinear charge sources are represented in the schematic diagram by capacitors to emphasize their capacitive nature. The charge source QGD is directed from the gate (n1) toward the drain (n2). The charge source QGS is directed from the gate (n1) toward the source (n3).



## FETU2

## user-definable FET model 2

## FETU2



## Keywords

## Defaults

IGS	gate-source current	0
IDS	drain-source current	0
IDG	drain-gate current	0
IGG	current between gate and node n4	0
IRIN	current between node n4 and source	0
QGG	gate capacitor charge	0
QDG	drain-gate capacitor charge	0

## Form

```
FETU2 n1 n2 n3 [n4] IGS = expression1   IDS = expression2
                    IDG = expression3   IGG = expression4
                    IRIN = expression5  QGG = expression6
                    QDG = expression7;
```

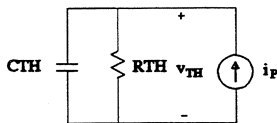
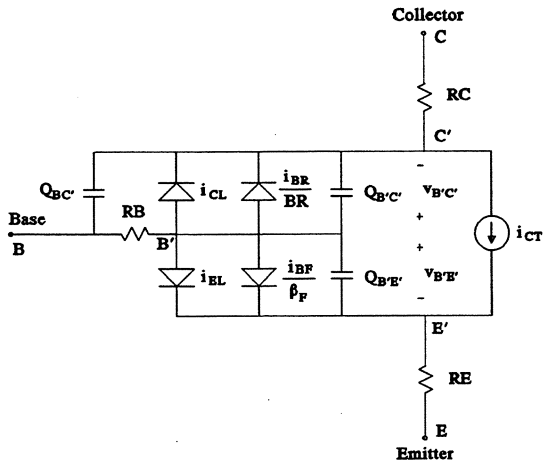
To utilize FETU2, you can create a set of model parameters of arbitrary meaning and names, and a set of voltage labels to represent the controlling voltages. The currents and charges are then defined as functions (expression1, ..., expression7) of these model parameters and voltage labels.

The nonlinear charge sources are represented in the schematic diagram by capacitors to emphasize their capacitive nature. The charge source QDG is directed from the gate (n1) toward the drain (n2). The charge source QGG is directed from n1 toward n4.

HBT

heterojunction bipolar transistor model

HBT



thermal circuit

Keywords

Defaults

CJE	base-emitter zero-bias depletion capacitance	0
VJE	base-emitter junction built-in potential	0.75
MJE	base-emitter junction exponential factor	0.33
TF	ideal forward-transit time	0
CJC	base-collector zero-bias depletion capacitance	0
VJC	base-collector junction built-in potential	0.75
MJC	base-collector junction exponential factor	0.33
XCJC	fraction of CJC that goes to internal base node	1.0
TR	ideal reverse-transit time	0
FC	coefficient for forward-bias depletion capacitance formula	0.5
IS0	transport saturation current	1.0E-23
BF0	ideal maximum forward beta	100
NF	forward-current emission coefficient	1.0
VAF	forward early voltage	$\infty$

**HBT****heterojunction bipolar transistor model****HBT****Keywords****Defaults**

IKF	corner for forward-beta high current roll-off	$\infty$
ISEO	base-emitter leakage saturation current	0
NE	base-emitter leakage emission coefficient	1.5
BR	ideal maximum reverse beta	1.0
NR	reverse-current emission coefficient	1.0
VAR	reverse early voltage	$\infty$
IKR	corner for reverse-beta high-current roll-off	$\infty$
ISCO	base-collector leakage saturation current	0
NC	base-collector leakage emission coefficient	2.0
EG	parameter for calculating the thermal effect	0
EINF	parameter for calculating the thermal effect	0
P	exponential parameter for calculating the thermal effect	0
XTI	exponential parameter for calculating the thermal effect	0
RTH	thermal resistance	1.0
CTH	thermal capacitance	0
RB	base resistance	10
RC	collector resistance	10
RE	emitter resistance	10

**Form**

```

HBT  n1 n2 n3  CJE = x1   VJE = x2   MJE = x3   TF = x4   GJC = x5
      VJC = x6   MJC = x7   XCJC = x8   TR = x9   FC = x10
      ISO = x11  BFO = x12  NF = x13  VAF = x14  IKF = x15
      ISEO = x16 NE = x17  BR = x18  NR = x19  VAR = x20
      IKR = x21  ISCO = x22  NC = x23  RB = x24  RC = x25
      RE = x26  EG = x27   EINF = x28  P = x29   XTI = x30
      RTH = x31  CTH = x32;

```

**Example**

```

HBT  1 2 3  VJE=0.75  MJE=0.33  VJC=0.75  MJC=0.33
      XCJC=1.0  FC=0.5   ISO=8.216E-16  BFO=408.1
      NF=0.983  VAF=105.8  IKF=7.581E-3  ISEO=8.233E-16
      NE=1.283  BR=1.0   NR=1.0   VAR=1000.0
      IKR=1.0E3  NC=2.0   RB=10.3  P=-1
      XTI=6.5   RTH=180  CTH=1.0E-6;

```

HBT

heterojunction bipolar transistor model

HBT

## Model Description

HBT is a modification of the BJT model by introducing the thermal effect following the thermal-electrical model of HBT [1]. A thermal subcircuit is used to evaluate the device temperature which is integrated into the DC, small-signal and large-signal harmonic balance simulation.

The temperature is calculated by

$$T = T_0 + v_{TH}$$

where  $T_0$  is the room temperature. The thermal effect on the device performance is characterized by the following formulas.

$$I_S = IS0 \cdot (T/T_0)^{XTI} \cdot \exp[EG/(kT_0) - EG/(kT)]$$

$$\beta_F = BFO \cdot (T/T_0)^P \cdot \exp[EINF/(kT) - EINF/(kT_0)]$$

where  $IS0$ ,  $XTI$ ,  $EG$ ,  $BFO$ ,  $P$  and  $EINF$  are model parameters.  $k$  is the Boltzmann constant. The saturation current  $I_S$  and forward beta  $\beta_F$  are required for evaluating the nonlinear currents passing through the diodes as shown in the equivalent circuit for HBT.

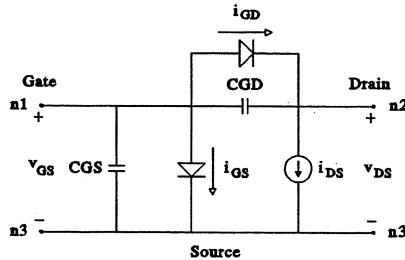
## Reference

- [1] H. Wang, C. Algani, A. Konczykowska and W. Zuberek, "Temperature dependence of DC currents in HBT," *IEEE MTT-S Int. Microwave Symp. Digest* (Albuquerque, NM), 1992, pp. 731-734.
- [2] P.C. Grossman and J. Choma, "Large signal modeling of HBT's including self-heating and transit time effects," *IEEE Trans. Microwave Theory Tech.*, vol. 40, 1992, pp. 449-464.

HEMTAC

advanced Curtice HEMT model

HEMTAC



## Keywords

## Defaults

ALPHA	hyperbolic tangent function parameter	0
BETA	transconductance parameter	0
LAMBDA	channel length modulation parameter	0
VT0	threshold voltage	-1.0
PSI	empirical transconductance degradation parameter (exponential)	0
XI	empirical transconductance degradation parameter (linear)	0
VPF	gate voltage where transconductance begins to degrade	0
GAMMA	voltage-slope parameter of pinch-off voltage	0
UCRIT	parameter representing the critical field for mobility degradation	0
VGEXP	gate voltage exponential parameter	2
VDS0	drain voltage where current saturation occurs with zero gate bias	1
CGD0	gate-drain fixed fringing capacitance	0
CGS0	gate-source fixed fringing capacitance	0
CM0	empirical total gate electrode capacitance parameter	0
CHI	empirical gate electrode coefficient parameter	1.0
TAU	transit time under gate	0
IS	gate junction saturation current	1.0E-14
N	gate-drain and gate-source emission coefficient	1.0
GMIN	linear conductance associated with the Schottky junctions	1.0E-12
VBR	gate reverse bias breakdown voltage	+∞
TEMP	temperature	298°K

## Form

```
HEMTAC n1 n2 n3 ALPHA = x1 BETA = x2 LAMBDA = x3 VT0 = x4
          PSI = x5 XI = x6 VPF = x7 GAMMA= x8
          UCRIT = x9 VGEXP = x10 VDS0 = x11 CGD0 = x12
          CGS0 = x13 CM0 = x14 CHI = x15 TAU = x16
          IS = x17 N = x18 GMIN = x19 VBR = x20
          TEMP = x21;
```

HEMTAC

advanced Curtice HEMT model

HEMTAC

### Example

```
HEMTAC 1 2 3 ALPHA=0.5 BETA=0.02 VTO=-1.2 PSI=2 VPF=-1
        CGD0=0.1PF CGS0=0.2PF;
```

### Model Equations

HEMTAC implements the advanced Curtice HEMT model [1].

All components of the equivalent circuit are assumed to be functions of the gate and drain voltages  $v_{GS}$  and  $v_{DS}$  as shown in the diagram. The nonlinear model equations are as follows.

$$i_{DS} = \begin{cases} i_{DSFET} & \text{for } v_{GS}(t-TAU) \leq VPF \\ i_{DSFET} - XI \cdot \beta_{eff} \cdot (v_{GS}(t-TAU) - VPF)^{PSI} \cdot f(v_{DS}) & \text{for } v_{GS}(t-TAU) > VPF \end{cases}$$

where

$$i_{DSFET} = \begin{cases} 0 & \text{for } v_{GS}(t-TAU) \leq V_T \\ \beta_{eff} \cdot (v_{GS}(t-TAU) - V_T)^{VGEXP} \cdot f(v_{DS}) & \text{for } v_{GS}(t-TAU) > V_T \end{cases}$$

$$f(v_{DS}) = \tanh(ALPHA \cdot v_{DS}) \cdot (1 + LAMBDA \cdot v_{DS})$$

$$\beta_{eff} = BETA / [1 + UCRIT \cdot (v_{GS}(t-TAU) - V_T)]$$

$$V_T = VTO + GAMMA \cdot v_{DS}$$

and ALPHA, BETA, LAMBDA, XI, PSI, VPF, VTO, GAMMA, UCRIT, VGEXP and TAU are model parameters.

$$C_{GS} = \begin{cases} CGS0 + 2 \cdot C_G \cdot [1 - (V_{DSS} - v_{DS})^2 / (2 \cdot V_{DSS} - v_{DS})^2] / 3 & \text{for } v_{DS} < V_{DSS} \\ 2 \cdot C_G / 3 + CGS0 & \text{for } v_{DS} \geq V_{DSS} \end{cases}$$

$$C_{GD} = \begin{cases} CGD0 + 2 \cdot C_G \cdot [1 - V_{DSS}^2 / (2 \cdot V_{DSS} - v_{DS})^2] / 3 & \text{for } v_{DS} < V_{DSS} \\ CGD0 & \text{for } v_{DS} \geq V_{DSS} \end{cases}$$

where

$$C_G = \begin{cases} 0 & \text{for } v_{GS} \leq VTO \\ CM0 \cdot (v_{GS} - VTO)^{1/CHI} & \text{for } v_{GS} > VTO \end{cases}$$

HEMTAC

advanced Curtice HEMT model

HEMTAC

$$V_{DSS} = \begin{cases} 0 & \text{for } v_{GS} \leq VT0 \\ VDS0 \cdot (1 - v_{GS}/VT0) & \text{for } v_{GS} > VT0 \end{cases}$$

and CGS0, CGD0, CM0, VDS0, CHI and VT0 are model parameters.

$$i_{GS} = \begin{cases} IS \cdot [\exp(v_{GS}/(N \cdot v_t)) - 1] + GMIN \cdot v_{GS} & \text{for } v_{GS} \geq -5N \cdot v_t \\ -IS + GMIN \cdot v_{GS} & \text{for } -VBR + 50v_t < v_{GS} < -5N \cdot v_t \\ -IS \cdot [\exp(-(v_{GS} + VBR)/v_t) + 1] + GMIN \cdot v_{GS} & \text{for } v_{GS} \leq -VBR + 50v_t \end{cases}$$

where  $v_t = k \cdot \text{TEMP} / q$  is the thermal voltage. IS, N, GMIN, VBR and TEMP are model parameters.

The diode current  $i_{GD}$  is similarly defined.

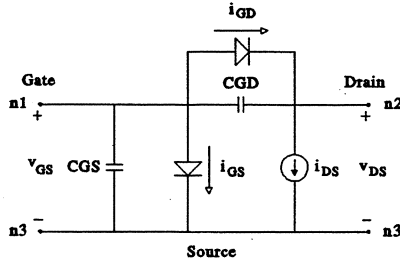
## Reference

- [1] J.M. Golio, *Microwave MESFETs and HEMTs*, Norwood, MA: Artech House, 1991.

HEMTC

Curtice HEMT model

HEMTC



Keywords

Defaults

ALPHA	hyperbolic tangent function parameter	0
BETA	transconductance parameter	0
LAMBDA	channel length modulation parameter	0
VT0	threshold voltage	-1.0
PSI	empirical transconductance degradation parameter (exponential)	0
XI	empirical transconductance degradation parameter (linear)	0
VPF	gate voltage where transconductance begins to degrade	0
VDS0	drain voltage where current saturation occurs with zero gate bias	1
CGD0	gate-drain fixed fringing capacitance	0
CGS0	gate-source fixed fringing capacitance	0
CM0	empirical total gate electrode capacitance parameter	0
CHI	empirical gate electrode coefficient parameter	1.0
TAU	transit time under gate	0
IS	gate junction saturation current	1.0E-14
N	gate-drain and gate-source emission coefficient	1.0
GMIN	linear conductance associated with the Schottky junctions	1.0E-12
VBR	gate reverse bias breakdown voltage	+∞
TEMP	temperature	298°K

Form

HEMTC n1 n2 n3 ALPHA = x1 BETA = x2 LAMBDA = x3 VT0 = x4  
 PSI = x5 XI = x6 VPF = x7 VDS0 = x8  
 CGD0 = x9 CGS0 = x10 CM0 = x11 CHI = x12  
 TAU = x13 IS = x14 N = x15 GMIN = x16  
 VBR = x17 TEMP = x18;



HEMTC

Curtice HEMT model

HEMTC

Example

HEMTC 1 2 3 BETA=0.02 VTO=-1.2 PSI=2;

Model Equations

HEMTC implements the Curtice HEMT model [1].

All components of the equivalent circuit are assumed to be functions of the gate and drain voltages  $v_{GS}$  and  $v_{DS}$  as shown in the diagram. The nonlinear model equations are as follows.

$$i_{DS} = \begin{cases} i_{DSFET} & \text{for } v_{GS}(t-TAU) \leq VPF \\ i_{DSFET} - XI \cdot (v_{GS}(t-TAU) - VPF)^{PSI+1} \cdot f(v_{DS}) / (PSI+1) & \text{for } v_{GS}(t-TAU) > VPF \end{cases}$$

where

$$i_{DSFET} = \begin{cases} 0 & \text{for } v_{GS}(t-TAU) \leq VTO \\ BETA \cdot (v_{GS}(t-TAU) - VTO)^2 \cdot f(v_{DS}) & \text{for } v_{GS}(t-TAU) > VTO \end{cases}$$

$$f(v_{DS}) = \tanh(ALPHA \cdot v_{DS}) \cdot (1 + LAMBDA \cdot v_{DS})$$

and ALPHA, BETA, LAMBDA, XI, PSI, VPF, VTO and TAU are model parameters.

$$C_{GS} = \begin{cases} CGS0 + 2 \cdot C_G \cdot [1 - (V_{DSS} - v_{DS})^2 / (2 \cdot V_{DSS} - v_{DS})^2] / 3 & \text{for } v_{DS} < V_{DSS} \\ 2 \cdot C_G / 3 + CGS0 & \text{for } v_{DS} \geq V_{DSS} \end{cases}$$

$$C_{GD} = \begin{cases} CGD0 + 2 \cdot C_G \cdot [1 - V_{DSS}^2 / (2 \cdot V_{DSS} - v_{DS})^2] / 3 & \text{for } v_{DS} < V_{DSS} \\ CGD0 & \text{for } v_{DS} \geq V_{DSS} \end{cases}$$

where

$$C_G = \begin{cases} 0 & \text{for } v_{GS} \leq VTO \\ CM0 \cdot (v_{GS} - VTO)^{1/CHI} & \text{for } v_{GS} > VTO \end{cases}$$

$$V_{DSS} = \begin{cases} 0 & \text{for } v_{GS} \leq VTO \\ VDS0 \cdot (1 - v_{GS}/VTO) & \text{for } v_{GS} > VTO \end{cases}$$

and CGS0, CGD0, CM0, VDS0, CHI and VTO are model parameters.

## HEMTC

## Curtice HEMT model

## HEMTC

$$i_{GS} = \begin{cases} IS \cdot [\exp(v_{GS}/(N \cdot v_t)) - 1] + GMIN \cdot v_{GS} & \text{for } v_{GS} \geq -5N \cdot v_t \\ -IS + GMIN \cdot v_{GS} & \text{for } -VBR + 50v_t < v_{GS} < -5N \cdot v_t \\ -IS \cdot [\exp(-(v_{GS} + VBR)/v_t) + 1] + GMIN \cdot v_{GS} & \text{for } v_{GS} \leq -VBR + 50v_t \end{cases}$$

where  $v_t = k \cdot \text{TEMP} / q$  is the thermal voltage. IS, N, GMIN, VBR and TEMP are model parameters.

The diode current  $i_{GD}$  is similarly defined.

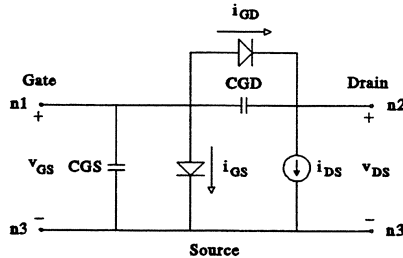
## Reference

[1] J.M. Golio, *Microwave MESFETs and HEMTs*, Norwood, MA: Artech House, 1991.

HEMTG1

high order beta degradation HEMT model

HEMTG1



Keywords

Defaults

ALPHA	hyperbolic tangent function parameter	0
BETA	transconductance parameter	0
LAMBDA	channel length modulation parameter	0
VTO	threshold voltage	-1.0
GAMMA	voltage-slope parameter of pinch-off voltage	0
UCRIT	parameter representing the critical field for mobility degradation	0
VGEXP	gate voltage exponential parameter	2
GMEXP	beta degradation exponential parameter	1
VDS0	drain voltage where current saturation occurs with zero gate bias	1
CGD0	gate-drain fixed fringing capacitance	0
CGS0	gate-source fixed fringing capacitance	0
CM0	empirical total gate electrode capacitance parameter	0
CHI	empirical gate electrode coefficient parameter	1.0
TAU	transit time under gate	0
IS	gate junction saturation current	1.0E-14
N	gate-drain and gate-source emission coefficient	1.0
GMIN	linear conductance associated with the Schottky junctions	1.0E-12
VBR	gate reverse bias breakdown voltage	+∞
TEMP	temperature	298°K

Form

HEMTG1 n1 n2 n3 ALPHA = x1 BETA = x2 LAMBDA = x3 VTO = x4  
 GAMMA = x5 UCRIT = x6 VGEXP = x7 GMEXP = x8  
 VDS0 = x9 CGD0 = x10 CGS0 = x11 CM0 = x12  
 CHI = x13 TAU = x14 IS = x15 N = x16  
 GMIN = x17 VBR = x18 TEMP = x19;

**HEMTG1 high order beta degradation HEMT model HEMTG1**

**Example**

```
HEMTG1 1 2 3 ALPHA=0.5 BETA=0.02 VTO=-1.2 GMEXP=2 CGDO=0.1PF
        CGSO=0.2PF;
```

**Model Equations**

HEMTG1 implements the high order beta degradation HEMT model [1].

All components of the equivalent circuit are assumed to be functions of the gate and drain voltages  $v_{GS}$  and  $v_{DS}$  as shown in the diagram. The nonlinear model equations are as follows.

$$i_{DS} = \begin{cases} 0 & \text{for } v_{GS}(t-TAU) \leq V_T \\ \beta_{eff} \cdot (v_{GS}(t-TAU) - V_T)^{VGEXP} \cdot f(v_{DS}) & \text{for } v_{GS}(t-TAU) > V_T \end{cases}$$

where

$$f(v_{DS}) = \tanh(ALPHA \cdot v_{DS}) \cdot (1 + LAMBDA \cdot v_{DS})$$

$$\beta_{eff} = BETA / [1 + UCRIT \cdot (v_{GS}(t-TAU) - V_T)^{GMEXP}]$$

$$V_T = VTO + GAMMA \cdot v_{DS}$$

and ALPHA, BETA, LAMBDA, VTO, GAMMA, UCRIT, VGEXP, GMEXP and TAU are model parameters.

$$C_{GS} = \begin{cases} CGSO + 2 \cdot C_G \cdot [1 - (V_{DSS} - v_{DS})^2 / (2 \cdot V_{DSS} - v_{DS})^2] / 3 & \text{for } v_{DS} < V_{DSS} \\ 2 \cdot C_G / 3 + CGSO & \text{for } v_{DS} \geq V_{DSS} \end{cases}$$

$$C_{GD} = \begin{cases} CGDO + 2 \cdot C_G \cdot [1 - V_{DSS}^2 / (2 \cdot V_{DSS} - v_{DS})^2] / 3 & \text{for } v_{DS} < V_{DSS} \\ CGDO & \text{for } v_{DS} \geq V_{DSS} \end{cases}$$

where

$$C_G = \begin{cases} 0 & \text{for } v_{GS} \leq VTO \\ CM0 \cdot (v_{GS} - VTO)^{1/CHI} & \text{for } v_{GS} > VTO \end{cases}$$

$$V_{DSS} = \begin{cases} 0 & \text{for } v_{GS} \leq VTO \\ VDS0 \cdot (1 - v_{GS}/VTO) & \text{for } v_{GS} > VTO \end{cases}$$

and CGSO, CGDO, CM0, VDS0, CHI and VTO are model parameters.

## HEMTG1      high order beta degradation HEMT model      HEMTG1

$$i_{GS} = \begin{cases} IS \cdot [\exp(v_{GS}/(N \cdot v_t)) - 1] + GMIN \cdot v_{GS} & \text{for } v_{GS} \geq -5N \cdot v_t \\ -IS + GMIN \cdot v_{GS} & \text{for } -VBR + 50v_t < v_{GS} < -5N \cdot v_t \\ -IS \cdot [\exp(-(v_{GS} + VBR)/v_t) + 1] + GMIN \cdot v_{GS} & \text{for } v_{GS} \leq -VBR + 50v_t \end{cases}$$

where  $v_t = k \cdot \text{TEMP} / q$  is the thermal voltage. IS, N, GMIN, VBR and TEMP are model parameters.

The diode current  $i_{GD}$  is similarly defined.

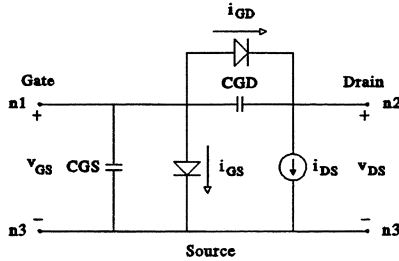
## Reference

- [1] D. Halchin, M. Miller, M. Golio and S. Tehrani, "HEMT models for large signal circuit simulation," *IEEE MTT-S Int. Microwave Symp. Digest* (San Diego, CA), 1994, pp. 985-988.

HEMTG2

double parabolic HEMT model

HEMTG2



Keywords

Defaults

ALPHA	hyperbolic tangent function parameter	0
BETA	transconductance parameter	0
LAMBDA	channel length modulation parameter	0
VTO	threshold voltage	-1.0
GAMMA	voltage-slope parameter of pinch-off voltage	0
VC	gate voltage where the peak of transconductance occurs	0
VDS0	drain voltage where current saturation occurs with zero gate bias	1
CGD0	gate-drain fixed fringing capacitance	0
CGS0	gate-source fixed fringing capacitance	0
CM0	empirical total gate electrode capacitance parameter	0
CHI	empirical gate electrode coefficient parameter	1.0
TAU	transit time under gate	0
IS	gate junction saturation current	1.0E-14
N	gate-drain and gate-source emission coefficient	1.0
GMIN	linear conductance associated with the Schottky junctions	1.0E-12
VBR	gate reverse bias breakdown voltage	+∞
TEMP	temperature	298°K

Form

```
HEMTG2 n1 n2 n3 ALPHA = x1 BETA = x2 LAMBDA = x3 VTO = x4
GAMMA= x5 VC = x6 VDS0 = x7 CGD0 = x8
CGS0 = x9 CM0 = x10 CHI = x11 TAU = x12
IS = x13 N = x14 GMIN = x15 VBR = x16
TEMP = x17;
```

HEMTG2

double parabolic HEMT model

HEMTG2

Example

HEMTG2 1 2 3 ALPHA=0.5 BETA=0.02 VT0=-1.2 VDS0=-1 CGD0=0.1PF  
CGS0=0.2PF;

Model Equations

HEMTG2 implements the double parabolic HEMT model [1].

All components of the equivalent circuit are assumed to be functions of the gate and drain voltages  $v_{GS}$  and  $v_{DS}$  as shown in the diagram. The nonlinear model equations are as follows.

$$i_{DS} = \begin{cases} 0 & \text{for } v_{GS}(t-TAU) < V_T \\ BETA \cdot (v_{GS}(t-TAU) - GAMMA \cdot v_{DS} - VT0)^3 \cdot f(v_{DS})/3 & \text{for } V_T \leq v_{GS}(t-TAU) < V_{FF} \\ BETA \cdot [(VC-VT0)^2 \cdot (v_{GS}(t-TAU) - V_{FF})/2 - (v_{GS}(t-TAU) - GAMMA \cdot v_{DS} - VC)^3/3] \cdot f(v_{DS}) & \text{for } v_{GS}(t-TAU) \geq V_{FF} \end{cases}$$

where

$$f(v_{DS}) = \tanh(ALPHA \cdot v_{DS}) \cdot (1 + LAMBDA \cdot v_{DS})$$

$$V_{FF} = (VT0 + VC)/2 + GAMMA \cdot v_{DS}$$

$$V_T = VT0 + GAMMA \cdot v_{DS}$$

and ALPHA, BETA, LAMBDA, VT0, GAMMA, VC and TAU are model parameters.

$$C_{GS} = \begin{cases} CGS0 + 2 \cdot C_G \cdot [1 - (V_{DSS} - v_{DS})^2 / (2 \cdot V_{DSS} - v_{DS})^2] / 3 & \text{for } v_{DS} < V_{DSS} \\ 2 \cdot C_G / 3 + CGS0 & \text{for } v_{DS} \geq V_{DSS} \end{cases}$$

$$C_{GD} = \begin{cases} CGD0 + 2 \cdot C_G \cdot [1 - V_{DSS}^2 / (2 \cdot V_{DSS} - v_{DS})^2] / 3 & \text{for } v_{DS} < V_{DSS} \\ CGD0 & \text{for } v_{DS} \geq V_{DSS} \end{cases}$$

where

$$C_G = \begin{cases} 0 & \text{for } v_{GS} \leq VT0 \\ CM0 \cdot (v_{GS} - VT0)^{1/CHI} & \text{for } v_{GS} > VT0 \end{cases}$$

HEMTG2

double parabolic HEMT model

HEMTG2

$$V_{DSS} = \begin{cases} 0 & \text{for } v_{GS} \leq VT0 \\ VDS0 \cdot (1 - v_{GS}/VT0) & \text{for } v_{GS} > VT0 \end{cases}$$

and CGS0, CGD0, CM0, VDS0, CHI and VT0 are model parameters.

$$i_{GS} = \begin{cases} IS \cdot [\exp(v_{GS}/(N \cdot v_t)) - 1] + GMIN \cdot v_{GS} & \text{for } v_{GS} \geq -5N \cdot v_t \\ -IS + GMIN \cdot v_{GS} & \text{for } -VBR + 50v_t < v_{GS} < -5N \cdot v_t \\ -IS \cdot [\exp(-(v_{GS} + VBR)/v_t) + 1] + GMIN \cdot v_{GS} & \text{for } v_{GS} \leq -VBR + 50v_t \end{cases}$$

where  $v_t = k \cdot TEMP / q$  is the thermal voltage. IS, N, GMIN, VBR and TEMP are model parameters.

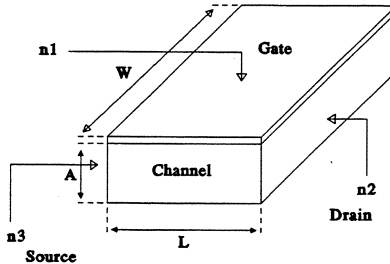
The diode current  $i_{GD}$  is similarly defined.

Reference

- [1] D. Halchin, M. Miller, M. Golio and S. Tehrani, "HEMT models for large signal circuit simulation," *IEEE MTT-S Int. Microwave Symp. Digest* (San Diego, CA), 1994, pp. 985-988.



## KTL physics-based bias-dependent small-signal FET model KTL



## Keywords

## Defaults

L	gate length in m	0.56E-6
W	gate width in m	1.2E-3
A	channel thickness in m	0.35E-6
EPSR	relative dielectric constant	12.5
VS	saturation electron velocity in m/ns	1.5E-4
VBI	built-in potential in V	0.7
U0	low-field mobility in $m^2/(Vns)$	4.0E-10
D	high-field diffusion coefficient in $m^2/ns$	1.0E-12
ND	doping density (for uniform doping) in $m^{-3}$	7.5E22
LAMBDA	parameter for the transition function in m	1.0E-8
ALPHA	exponent for calculating the average mobility	0.5
AA	parameter for electron velocity curve in V/m	1.0E6
BB	parameter for electron velocity curve	1
CC	parameter for electron velocity curve	1
DD	parameter for electron velocity curve	0
A0	proportional constant	1
R01	parameter for output resistance in Ohm/V <sup>2</sup>	0.01
R02	parameter for output resistance in V	14
R03	parameter for output resistance in Ohm	600
M	number of gate fingers of the MESFET	4
LGG0	parameter for gate inductance in nH	0.02
CDS	drain-source capacitance	0
GDS	drain-source conductance	0
RG	gate resistance	0
RD	drain resistance	0
RS	source resistance	0
CGE	external gate capacitance	0
CDE	external drain capacitance	0
CX	drain-source capacitance	1.5pF
LD	drain lead inductance	0
LS	source lead inductance	0

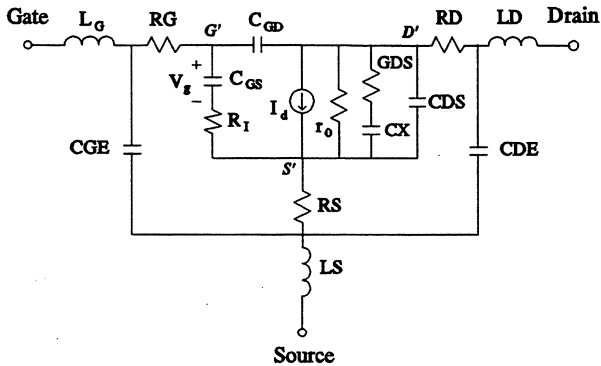
**KTL physics-based bias-dependent small-signal FET model KTL**

**Form**

```
KTL(n1, n2, n3,
    L, W, A, EPSR, EC, VS, VBI, UO, D, ND, LAMBDA, ALPHA,
    AA, BB, CC, DD, A0, R01, R02, R03, LGG0, M,
    LD, LS, RG, RD, RSO, GDS, CX, CDS, CGE, CDE, $)
```

**Notes**

- ▷ KTL is implemented in an include file named "ktl.inc" which defines both the intrinsic and extrinsic parameters of the MESFET. To use the KTL model, you can incorporate the contents of "ktl.inc" directly into the input file, or using the "#include ktl.inc" (see Chapter 3 for further details on the #include preprocessor directive).
- ▷ KTL combines the Khatibzadeh and Trew physics-based nonlinear model (FETT1, for solving for the DC operating point) and the Ladbroke small-signal equivalent circuit.
- ▷ KTL assumes uniform doping.
- ▷ KTL can only be used for small-signal simulation.



*The small-signal equivalent circuit for the KTL model.*

## KTL physics-based bias-dependent small-signal FET model KTL

## Model Equations

The model equations for defining doping density, electron velocity and transition function are the same as those in FETT1.

In the small-signal equivalent circuit we have

$$I_d = g_m \cdot V_g \cdot e^{-j\omega \tau}.$$

The bias-dependent small-signal parameters are  $L_G$ ,  $C_{GS}$ ,  $C_{GD}$ ,  $R_I$ ,  $g_m$ ,  $\tau$  and  $r_0$ . They are derived using the modified Ladbrooke formulas once the DC operating point is solved for.

$$L_G = 4 \times 10^2 \cdot \pi \cdot d \cdot W / (M^2 \cdot L) + L_{G0} \quad (\text{nH})$$

$$\epsilon = 8.854 \times 10^{-12} \cdot \text{EPSR}$$

$$C_{GS} = 10^9 \cdot \epsilon \cdot W \cdot L \cdot (1 + X/(2 \cdot L) - 2 \cdot d/(L + 2 \cdot X)) / d \quad (\text{nF})$$

$$C_{GD} = 2 \times 10^9 \cdot \epsilon \cdot W / (1 + 2 \cdot X/L) \quad (\text{nF})$$

$$R_I = VS \cdot L / (U0 \cdot I_{CH}) \quad (\text{Ohm})$$

$$g_m = \epsilon \cdot VS \cdot W / d \quad (1/\text{Ohm})$$

$$\tau = (X/2 - 2 \cdot d/(1 + 2 \cdot X/L)) / VS \quad (\text{nS})$$

$$r_0 = R01 \cdot V_{D'S'} \cdot (R02 - V_{G'S'}) + R03 \quad (\text{Ohm})$$

where the equivalent depletion depth, space-charge layer extension and channel current are

$$d = [2 \cdot \epsilon \cdot (V_{BI} - V_{G'S'}) / (q \cdot ND)]^{0.5} \quad (\text{m})$$

$$X = A0 \cdot (2 \cdot \epsilon / [q \cdot ND (V_{BI} - V_{G'S'})])^{0.5} (V_{D'S'} - V_{G'S'} + V_{BI}) \quad (\text{m})$$

$$I_{CH} = q \cdot ND \cdot VS \cdot 10^9 \cdot (A - d) \cdot W \quad (\text{A})$$

$V_{D'S'}$  and  $V_{G'S'}$  are DC intrinsic voltages from  $D'$  to  $S'$  and from  $G'$  to  $S'$ , respectively, as shown in the small-signal equivalent circuit diagram.

KTL physics-based bias-dependent small-signal FET model KTL

Include File "ktl.inc"

```
#define KTL(gate, drain, source,
    Lx, Wx, Ax, EPSRx, VSx, VBIx, UOx, Dx, NDx, LAMBDAx, ALPHAx,
    AAx, BBx, CCx, DDx, A0x, R01x, R02x, R03x, LGG0x, Mx,
    LDx, LSx, RGx, RDx, RSx, GDSx, CXx, CDSx, CGEx, CDEx, S) {

    EPSI_$ = 8.854E-12 * EPSRx;
    VB0_$ = VBIx;
    EC_$ = VSx / UOx;

    VLABEL @int_gate$ @int_source$ NAME=VGS_DC_$;
    VLABEL @int_drain$ @int_source$ NAME=VDS_DC_$;

    ! equivalent depletion depth d = [2 * EPS * (-vgs + Vb0) / (q * N)]^0.5

    TMP_$ = (EPSI_$ + EPSI_$) / (1.6E-19 * NDx);
    D_$ = SQRT(TMP_$ * (-VGS_DC_$ + VB0_$));

    ! space-charge layer extension x = a0 * [2*EPS/[q*N*(-vgs+Vb0)]]^0.5 * (vdg+Vb0)

    X_$ = A0x * SQRT(TMP_$/(-VGS_DC_$ + VB0_$))*(VDS_DC_$ - VGS_DC_$ + VB0_$);

    ! the channel current Ich = q N Vsat (W - d) Zg

    ICH_$ = 1.6E-19 * NDx * VSx * (Ax - D_$) * Wx * 1.0E9;

    ! LGG_$ = 4*PI*1.0E-7 * D_$ * W / (M * M * L) * 1.0E9 + LGG0;
    LGG_$ = 400 * PI * D_$ * Wx / (Mx * Mx * Lx) + LGG0x;

    CGS_$ = 1.0E9 * EPSI_$ * Wx * Lx / D_$ * (1 + X_$ / (Lx + Lx)
        - (D_$ + D_$) / (Lx + X_$ + X_$));

    CGD_$ = 1.0E9 * 2 * EPSI_$ * Wx / (1 + (X_$ + X_$) / Lx);
    RI_$ = VSx * Lx / (UOx * ICH_$);
    TAU_$ = (X_$ / 2 - (D_$ + D_$) / (1 + (X_$ + X_$) / Lx)) / VSx;

    EXTRINSIC2 @int_gate$ @int_drain$ @int_source$ gate drain source
        LG=LGG_$ LD=LDx LS=LSx RG=RGx RD=RDx RS=RSx
        GDS=GDSx CX=CXx CDS=CDSx CGE=CGEx CDE=CDEx;

    SRC @int_gate$ @int_source$ R=RI_$ C=CGS_$;
    CAP @int_gate$ @int_drain$ C=CGD_$;

    FEILT @int_gate$ @int_drain$ @int_source$
        TAU=TAU_$ L=Lx W=Wx A=Ax EPSR=EPSRx EC=EC_$ VS=VSx
        VBI=VBIx UO=UOx D=Dx ND=NDx LAMBDA=LAMBDAx ALPHA=ALPHAx AA=AAx
        BB=BBx CC=CCx DD=DDx A0=A0x R01=R01x R02=R02x R03=R03x;
}
```

## KTL physics-based bias-dependent small-signal FET model KTL

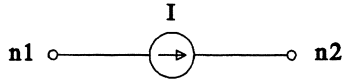
## References

- [1] M.A. Khatibzadeh and R.J. Trew, "A large-signal, analytic model for the GaAs MESFET", *IEEE Trans. Microwave Theory Tech.*, vol. 36, 1988, pp. 231-238.
- [2] P.H. Ladbrooke, *MMIC Design: GaAs FETs and HEMTs*. Norwood, MA: Artech House, 1989.
- [3] J.W. Bandler, R.M. Biernacki, S.H. Chen, J. Song, S. Ye and Q.J. Zhang, "Statistical modeling of GaAs MESFETs", *IEEE MTT-S Int. Microwave Symp. Dig.* (Boston, MA), 1991, pp. 87-90.
- [4] M.A. Khatibzadeh, "Large-signal modeling of gallium-arsenide field effect transistors", Ph.D. dissertation, North Carolina State University, Raleigh, NC, 1987.

NLCCS

nonlinear controlled current source

NLCCS



## Keyword

## Defaults

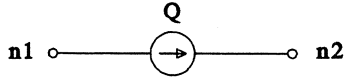
I current

0

## Form

```
NLCCS n1 [n2] I = expression;
```

To utilize NLCCS, you can create a set of model parameters of arbitrary meaning and names, and one or more voltage labels to represent the controlling voltages. The current is then defined as a function (expression) of these model parameters and voltage labels.

**NLCQ****nonlinear controlled charge source****NLCQ****Keyword**

Q charge

**Defaults**

0

**Form**

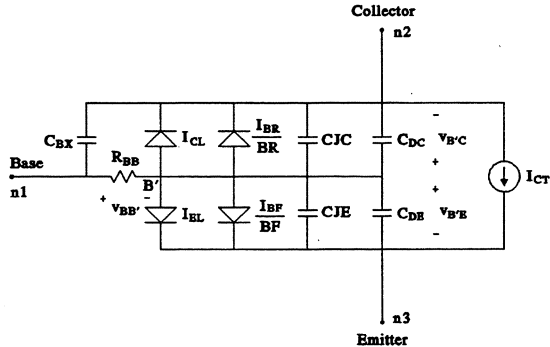
```
NLCQ n1 [n2] Q = expression;
```

To utilize NLCQ, you can create a set of model parameters of arbitrary meaning and names, and one or more voltage labels to represent the controlling voltages. The charge is then defined as a function (expression) of these model parameters and voltage labels.

NPN

NPN Gummel and Poon bipolar transistor model

NPN



Keywords

Defaults

CJE	base-emitter zero-bias depletion capacitance	0
VJE	base-emitter junction built-in potential	0.75
MJE	base-emitter junction exponential factor	0.33
TF	ideal forward-transit time	0
CJC	base-collector zero-bias depletion capacitance	0
VJC	base-collector junction built-in potential	0.75
MJC	base-collector junction exponential factor	0.33
XCJC	fraction of CJC that goes to internal base node	1.0
TR	ideal reverse-transit time	0
FC	coefficient for forward-bias depletion capacitance formula	0.5
IS	transport saturation current	1.0E-16
BF	ideal maximum forward beta	100
NF	forward-current emission coefficient	1.0
VAF	Forward Early voltage	∞
IKF	corner for forward-beta high current roll-off	∞
ISE	base-emitter leakage saturation current	0
NE	base-emitter leakage emission coefficient	1.5
BR	ideal maximum reverse beta	1.0
NR	reverse-current emission coefficient	1.0
VAR	Reverse Early voltage	∞
IKR	corner for reverse-beta high-current roll-off	∞
ISC	base-collector leakage saturation current	0
NC	base-collector leakage emission coefficient	2.0
RB	zero-bias base resistance	10
RBM	minimum base resistance at high currents	10
IRB	current when base resistance falls half way to RBM	∞
TEMP	temperature	298°K



NPN

NPN Gummel and Poon bipolar transistor model

NPN

Form

```

NPN  n1 n2 n3  CJE = x1  VJE = x2  MJE = x3  TF = x4  CJC = x5
      VJC = x6  MJC = x7  XCJC = x8  TR = x9  FC = x10
      IS = x11  BF = x12  NF = x13  VAF = x14  IKF = x15
      ISE = x16  NE = x17  BR = x18  NR = x19  VAR = x20
      IKR = x21  ISC = x22  NC = x23  RB = x24  RBM = x25
      IRB = x26  TEMP = x27;

```

Example

```

NPN  1 2 3  VJE=0.75  MJE=0.33  VJC=0.75  MJC=0.33
      XCJC=1.0  FC=0.5  IS=8.216E-16  BF=408.1
      NF=0.983  VAF=105.8  IKF=7.581E-3  ISE=8.233E-16
      NE=1.283  BR=1.0  NR=1.0  VAR=1000.0
      IKR=1.0E3  NC=2.0  RB=199.3  RBM=52.05
      IRB=2.001E-3;

```

Model Description

NPN is adapted from the integral charge control model of Gummel and Poon [1]. It becomes the simpler Ebers-Moll model when certain parameters required for the Gummel and Poon model are not specified.

Model parameters CJE, MJE and VJE determine the nonlinear depletion-layer capacitance for the base-emitter junction. CJC, MJC and VJC determine the nonlinear depletion-layer capacitance for the base-collector junction. TR and TF along with the depletion-layer capacitances model the base charge storage effects.

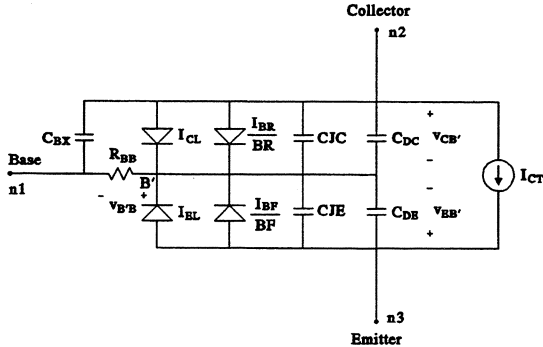
Reference

- [1] I.E. Getreu, *CAD of Electronic Circuits 1: Modeling the Bipolar Transistor*. Amsterdam: Elsevier Scientific Publishing Company, 1978, pp. 69-123.

PNP

PNP Gummel and Poon bipolar transistor model

PNP



Keywords

Defaults

CJE	base-emitter zero-bias depletion capacitance	0
VJE	base-emitter junction built-in potential	0.75
MJE	base-emitter junction exponential factor	0.33
TF	ideal forward-transit time	0
CJC	base-collector zero-bias depletion capacitance	0
VJC	base-collector junction built-in potential	0.75
MJC	base-collector junction exponential factor	0.33
XCJC	fraction of CJC that goes to internal base node	1.0
TR	ideal reverse-transit time	0
FC	coefficient for forward-bias depletion capacitance formula	0.5
IS	transport saturation current	1.0E-16
BF	ideal maximum forward beta	100
NF	forward-current emission coefficient	1.0
VAF	Forward Early voltage	∞
IKF	corner for forward-beta high current roll-off	∞
ISE	base-emitter leakage saturation current	0
NE	base-emitter leakage emission coefficient	1.5
BR	ideal maximum reverse beta	1.0
NR	reverse-current emission coefficient	1.0
VAR	Reverse Early voltage	∞
IKR	corner for reverse-beta high-current roll-off	∞
ISC	base-collector leakage saturation current	0
NC	base-collector leakage emission coefficient	2.0
RB	zero-bias base resistance	10
RBM	minimum base resistance at high currents	10
IRB	current when base resistance falls half way to RBM	∞
TEMP	temperature	298°K

PNP

PNP Gummel and Poon bipolar transistor model

PNP

## Form

```

PNP  n1 n2 n3  CJE = x1   VJE = x2   MJE = x3   TF = x4   CJC = x5
      VJC = x6   MJC = x7   XCJC = x8   TR = x9   FC = x10
      IS = x11  BF = x12   NF = x13   VAF = x14  IKF = x15
      ISE = x16  NE = x17   BR = x18   NR = x19  VAR = x20
      IKR = x21  ISC = x22  NC = x23   RB = x24  RBM = x25
      IRB = x26  TEMP = x27;

```

## Example

```

PNP  1 2 3  VJE=0.75  MJE=0.33  VJC=0.75  MJC=0.33
      XCJC=1.0  FC=0.5    IS=8.216E-16  BF=408.1
      NF=0.983  VAF=105.8  IKF=7.581E-3  ISE=8.233E-16
      NE=1.283  BR=1.0    NR=1.0     VAR=1000.0
      IKR=1.0E3  NC=2.0    RB=199.3   RBM=52.05
      IRB=2.001E-3;

```

## Model Description

PNP is adapted from the integral charge control model of Gummel and Poon [1]. It becomes the simpler Ebers-Moll model when certain parameters required for the Gummel and Poon model are not specified.

Model parameters CJE, MJE and VJE determine the nonlinear depletion-layer capacitance for the base-emitter junction. CJC, MJC and VJC determine the nonlinear depletion-layer capacitance for the base-collector junction. TR and TF along with the depletion-layer capacitances model the base charge storage effects.

## Reference

- [1] I.E. Getreu, *CAD of Electronic Circuits 1: Modeling the Bipolar Transistor*. Amsterdam: Elsevier Scientific Publishing Company, 1978, pp. 69-123.



## 8

## Linear Elements

CAP	capacitor	8-2
CCCS	current controlled current source	8-3
CCVS	current controlled voltage source	8-4
CIR3	ideal three-port circulator	8-5
CLIN	ideal coupled transmission lines	8-6
CLINP	coupled transmission lines, physical model	8-7
EXTRINSIC1	extrinsic parasitic model 1	8-8
EXTRINSIC2	extrinsic parasitic model 2	8-9
EXTRINSIC3	extrinsic parasitic model 3	8-10
EXTRINSIC4	extrinsic parasitic model 4	8-11
IND	inductor	8-12
MAGAP	microstrip asymmetric gap	8-13
MBEND1	microstrip 90 degree bend	8-14
MBEND2	microstrip chamfered 90 degree bend	8-15
MBEND3	microstrip optimally-chamfered 90 degree bend	8-16
MBEND3A	microstrip optimally-chamfered 90 degree bend	8-17
MCROSS	microstrip cross-junction	8-18
MGAP	microstrip symmetric gap	8-19
MLANG4	four-finger microstrip Lange coupler	8-20
MLANG6	six-finger microstrip Lange coupler	8-22
MLANG8	eight-finger microstrip Lange coupler	8-24
MOPEN	microstrip open stub	8-26
MRSTUB	microstrip radial stub	8-27
MSACL	asymmetrical coupled microstrip lines	8-28
MSCL	two-conductor symmetrical coupled microstrip lines	8-29
MSHORT	microstrip short stub	8-30
MSL	microstrip line	8-31
MSLIT	narrow transverse slit in microstrip	8-33
MSTEP	microstrip step	8-34
MSUB	microstrip substrate definition	8-35
MTEE	microstrip T-junction	8-36
OPEN	open circuit	8-37
PRC	parallel connection of resistor and capacitor	8-38
RES	resistor	8-39
SRC	series connection of resistor and capacitor	8-40
SRL	series connection of resistor and inductor	8-41
SRLC	series connection of resistor, inductor and capacitor	8-42
TEM	ideal transmission line	8-43
TRL	transmission line, physical model	8-44
VCCS	voltage controlled current source	8-45
VCVS	voltage controlled voltage source	8-46
DATAPORT	imported n-port subcircuit	8-47
SPORT	n-port subcircuit defined by S matrix	8-48
YPORT	n-port subcircuit defined by Y matrix	8-54
ZPORT	n-port subcircuit defined by Z matrix	8-60



## 8

# Linear Elements

This chapter describes OSA90's library of linear element models.

The description for each element includes a schematic diagram, the number of nodes, the set of parameters and their default values, an example of the syntax, and some application notes if appropriate.

The nodes for each element can be divided into two subsets. The first subset represents nodes which must be nonzero for the element to be meaningfully connected (e.g., the element RES requires at least one nonzero node). The other set represents nodes which are optional. In the description, the optional nodes are enclosed within square brackets. The default for optional nodes that are not explicitly specified is the ground node.

All the element parameters are optional except where noted. Element parameters can be specified by constant values, optimization variables and labels. Parameters that are omitted will be assigned the default values.

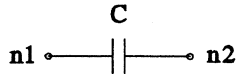
You can create customized elements as user-defined linear models, as described in Chapter 6.

The Empire library of microstrip structures is documented in the Empire User's Manual.

CAP

capacitor

CAP



## Keywords

C capacitance

## Defaults

0

## Form

CAP n1 [n2] C = x1;

## Example

CAP 1 2 C = 35PF;

## Notes

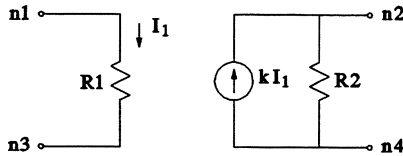
- ▷ If  $C = 0$  or is not specified the element becomes an open circuit.



CCCS

current controlled current source

CCCS



## Keywords

## Defaults

R1	input resistance	0
R2	output resistance	$\infty$
M	DC value of the controlling coefficient k	0
A	angle of the controlling coefficients	0
F	3 dB cut-off frequency of the controlling coefficient	$\infty$
T	time delay of the controlling coefficient	0

## Form

CCCS n1 [n2 n3 n4] R1 = x1 R2 = x2 M = x3 A = x4 F = x5 T = x6;

## Example

CCCS 1 2 3 4 R1=50KOH R2=5KOH M=50;

## Notes

- ▷ The controlling coefficient k is evaluated as

$$k = M \cdot \exp(j(A - 2\pi fT)) / (1 + jf/F)$$

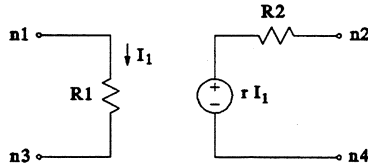
where M, A (converted to radians, if necessary), T and F are model parameters, and  $f$  is the current frequency.

- ▷ F, if specified, must be greater than 0.
- ▷ An ideal CCCS is defined by the default values for R1 and R2.
- ▷ A source of reverse polarity can be specified with a negative value for M.

CCVS

current controlled voltage source

CCVS



Keywords

Defaults

R1	input resistance	0
R2	output resistance	0
M	DC value of the transimpedance $r$	0
A	angle of the transimpedance $r$	0
F	3 dB cut-off frequency of the transimpedance $r$	$\infty$
T	time delay of the transimpedance $r$	0

Form

CCVS n1 [n2 n3 n4] R1 = x1 R2 = x2 M = x3 A = x4 F = x5 T = x6;

Example

CCVS 1 2 3 4 R1=50KOH R2=5KOH M=50;

Notes

- ▷ The transimpedance  $r$  is evaluated as

$$r = M \cdot \exp(j(A - 2\pi fT)) / (1 + jf/F)$$

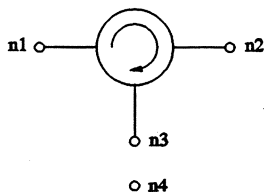
where  $M$ ,  $A$  (converted to radians, if necessary),  $T$  and  $F$  are model parameters, and  $f$  is the current frequency.

- ▷  $F$ , if specified, must be greater than 0.
- ▷ An ideal CCVS is defined by the default values for  $R1$  and  $R2$ .
- ▷ A source of reverse polarity can be specified with a negative value for  $M$ .

CIR3

ideal three-port circulator

CIR3



## Keywords

Defaults

RREF reference impedance

50

## Form

```
CIR3 n1 n2 n3 [n4] RREF = x1;
```

## Example

```
CIR3 1 2 3;
```

## Notes

- ▷ The ideal three-port circulator is modelled by the following scattering matrix:

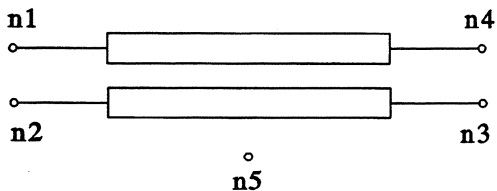
$$S = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

The reference impedance of the  $S$  parameters can be specified by RREF.

CLIN

ideal coupled transmission lines

CLIN



Keywords

Defaults

ZE	even-mode impedance	50
ZO	odd-mode impedance	50
E	electrical length in degrees	90
F	frequency at which the electrical length is E	1GHz

Form

CLIN n1 n2 [n3 n4 n5] ZE = x1 ZO = x2 E = x3 F = x4;

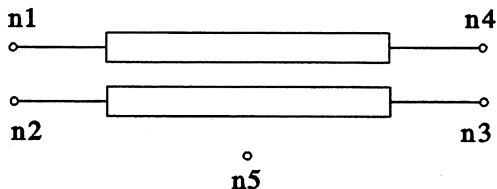
Example

CLIN 1 2 3 4 ZE=140 ZO=20 E=90 F=10GHz;

CLINP

coupled transmission lines, physical model

CLINP



## Keywords

## Defaults

ZE	even-mode impedance	50
ZO	odd-mode impedance	50
L	physical length	0.01
KE	even-mode effective dielectric constant	1
KO	odd-mode effective dielectric constant	1
AE	even-mode attenuation in dB per meter	0
AO	odd-mode attenuation in dB per meter	0

## Form

```
CLINP n1 n2 [n3 n4 n5] ZE = x1 ZO = x2 L = x3 KE = x4
      KO = x5 AE = x6 AO = x7;
```

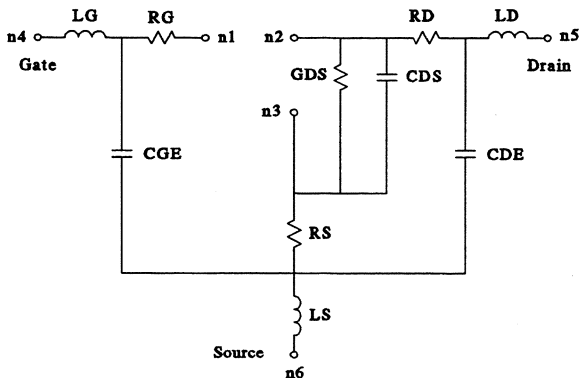
## Example

```
CLINP 1 2 3 4 ZE=140 ZO=20 L=115MIL KE=2.6 KO=2 AE=3 AO=3;
```

EXTRINSIC1

extrinsic parasitic model 1

EXTRINSIC1



Keywords

Defaults

CDS	drain-source (collector-emitter) capacitance	0
GDS	drain-source (collector-emitter) conductance	0
RG	gate (base) resistance	0
RD	drain (collector) resistance	0
RS	source (emitter) resistance	0
CGE	external gate (base) capacitance	0
CDE	external drain (collector) capacitance	0
LG	gate (base) lead inductance	0
LD	drain (collector) lead inductance	0
LS	source (emitter) lead inductance	0

Form

```
EXTRINSIC1 n1 n2 n3 n4 n5 [n6] CDS = x1  GDS = x2  RG = x3  RD = x4
RS = x5  CGE = x6  CDE = x7  LG = x8
LD = x9  LS = x10;
```

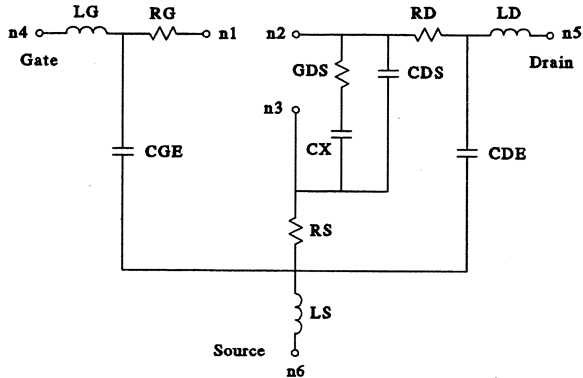
Example

```
EXTRINSIC1 1 2 3 4 5 RG=2.4 RD=1.54 RS=1.26
CDS=0.09PF LG=0.001NH;
```

## EXTRINSIC2

## extrinsic parasitic model 2

## EXTRINSIC2



## Keywords

## Defaults

CDS	drain-source (collector-emitter) capacitance	0
GDS	drain-source (collector-emitter) conductance	0
RG	gate (base) resistance	0
RD	drain (collector) resistance	0
RS	source (emitter) resistance	0
CGE	external gate (base) capacitance	0
CDE	external drain (collector) capacitance	0
CX	drain-source (collector-emitter) capacitance	1.5pF
LG	gate (base) lead inductance	0
LD	drain (collector) lead inductance	0
LS	source (emitter) lead inductance	0

## Form

```
EXTRINSIC2  n1 n2 n3 n4 n5 [n6]  CDS = x1  GDS = x2  RG = x3  RD = x4
           RS = x5  CGE = x6  CDE = x7  CX = x8
           LG = x9  LD = x10  LS = x11;
```

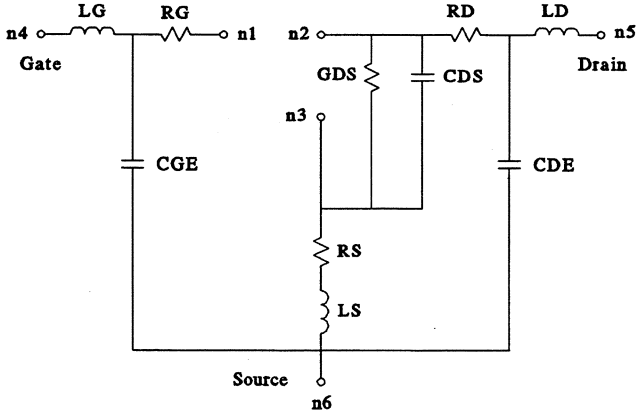
## Example

```
EXTRINSIC2  1 2 3 4 5  RG=2.4  RD=1.54  RS=1.26
              CX=2PF  LG=0.001NH;
```

EXTRINSIC3

extrinsic parasitic model 3

EXTRINSIC3



Keywords

Defaults

CDS	drain-source (collector-emitter) capacitance	0
GDS	drain-source (collector-emitter) conductance	0
RG	gate (base) resistance	0
RD	drain (collector) resistance	0
RS	source (emitter) resistance	0
CGE	external gate (base) capacitance	0
CDE	external drain (collector) capacitance	0
LG	gate (base) lead inductance	0
LD	drain (collector) lead inductance	0
LS	source (emitter) lead inductance	0

Form

```
EXTRINSIC3  n1 n2 n3 n4 n5 [n6]  CDS = x1  GDS = x2  RG = x3  RD = x4
           RS = x5  CGE = x6  CDE = x7  LG = x8
           LD = x9  LS = x10;
```

Example

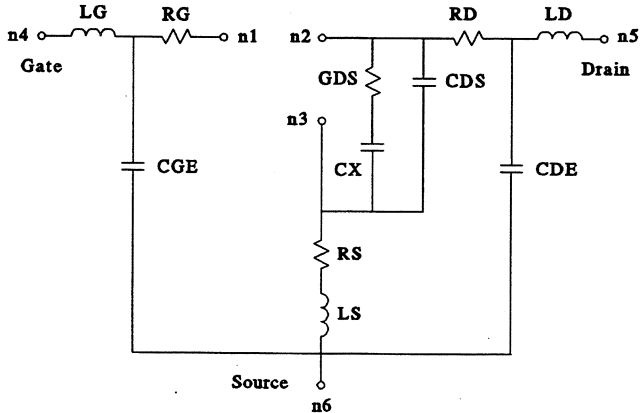
```
EXTRINSIC3  1 2 3 4 5  RG=2.4  RD=1.54  RS=1.26
                CDS=0.09PF  LG=0.001NH;
```



## EXTRINSIC4

## extrinsic parasitic model 4

## EXTRINSIC4



## Keywords

## Defaults

CDS	drain-source (collector-emitter) capacitance	0
GDS	drain-source (collector-emitter) conductance	0
RG	gate (base) resistance	0
RD	drain (collector) resistance	0
RS	source (emitter) resistance	0
CGE	external gate (base) capacitance	0
CDE	external drain (collector) capacitance	0
CX	drain-source (collector-emitter) capacitance	1.5pF
LG	gate (base) lead inductance	0
LD	drain (collector) lead inductance	0
LS	source (emitter) lead inductance	0

## Form

```
EXTRINSIC4 n1 n2 n3 n4 n5 [n6] CDS = x1 GDS = x2 RG = x3 RD = x4
RS = x5 CGE = x6 CDE = x7 CX = x8
LG = x9 LD = x10 LS = x11;
```

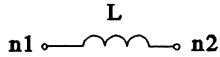
## Example

```
EXTRINSIC4 1 2 3 4 5 RG=2.4 RD=1.54 RS=1.26 CX=2PF LG=0.001NH;
```

IND

inductor

IND



## Keywords

## Defaults

L inductance

0

## Form

```
IND n1 [n2] L = x1;
```

## Example

```
IND 1 2 L=7NH;
```

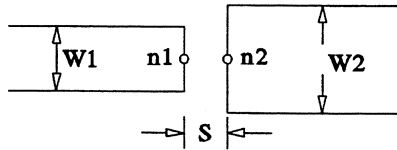
## Notes

- ▷ If  $L = 0$  or is not specified the element becomes a short circuit and the nodes  $n1$  and  $n2$  are effectively short circuited. However, short circuits are preferably defined by naming the two nodes identically throughout the circuit file.

MAGAP

microstrip asymmetric gap

MAGAP



## Keywords

defaults

W1 line width of conductor 1  
 W2 line width of conductor 2  
 S gap spacing

required  
 required  
 required

## Form

```
MAGAP n1 n2 W1 = x1 W2 = x2 S = x3;
```

## Example

```
MAGAP 1 2 W1=0.65mm W2=1mm S=0.2mm;
```

## Notes

- ▷ An MSUB statement should be placed before MAGAP to provide the substrate definition.
- ▷ MAGAP is valid for the range of parameters
  - $0.2 \leq S/H$
  - $0.1 \leq W1/H \leq 3.0$
  - $0.1 \leq W2/H \leq 3.0$
  - $1 \leq W2/W1$  (or  $W1/W2$ )  $\leq 3$
  - $6 \leq \text{EPSR} \leq 13$
 where H and EPSR are substrate parameters.

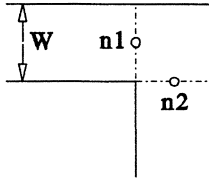
## Reference

- [1] M. Kirschning, R.H. Jansen and N.H.L. Koster, "Measurement and computer-aided modeling of microstrip discontinuities by an improved resonator method", *IEEE MTT-S Int. Microwave Symp. Dig.* (Boston, MA), May 1983, pp. 495-497.

MBEND1

microstrip 90 degree bend

MBEND1



## Keywords

defaults

W line width

required

## Form

```
MBEND1 n1 n2 W = x1;
```

## Example

```
MBEND1 1 2 W=1mm;
```

## Notes

- ▷ An MSUB statement should be placed before MBEND1 to provide the substrate definition.
- ▷ MBEND1 is valid for the range of parameters
 
$$0.2 \leq W/H \leq 6.0$$

$$2 \leq \text{EPSR} \leq 13$$
 where  $H$  and  $\text{EPSR}$  are substrate parameters.
- ▷ MBEND1 was validated in [1] for the frequencies up to 14GHZ.

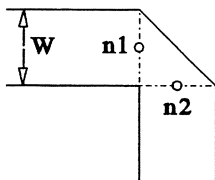
## Reference

- [1] M. Kirschning, R. Jansen and N. Koster, "Measurement and computer-aided modeling of microstrip discontinuities by an improved resonator method", *IEEE MTT-S Int. microwave Symp. Dig.* (Boston, MA), 1983, pp. 495-497.

MBEND2

microstrip chamfered 90 degree bend

MBEND2



## Keywords

defaults

W line width

required

## Form

```
MBEND2 n1 n2 W = x1;
```

## Example

```
MBEND2 1 2 W=1mm;
```

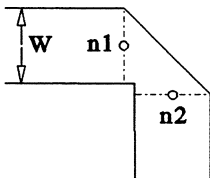
## Notes

- ▷ An MSUB statement should be placed before MBEND2 to provide the substrate definition.
- ▷ MBEND2 is valid for the range of parameters
 
$$0.2 \leq W/H \leq 6.0$$

$$2 \leq \text{EPSR} \leq 13$$
 where H and EPSR are substrate parameters.
- ▷ MBEND2 was validated in [1] for the frequencies up to 14GHZ.

## Reference

- [1] M. Kirschning, R. Jansen and N. Koster, "Measurement and computer-aided modeling of microstrip discontinuities by an improved resonator method", *IEEE MTT-S Int. microwave Symp. Dig.* (Boston, MA), 1983, pp. 495-497.

**MBEND3 microstrip optimally-chamfered 90 degree bend MBEND3****Keywords****defaults**

W line width

**required****Form**

```
MBEND3 n1 n2 W = x1;
```

**Example**

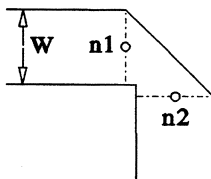
```
MBEND3 1 2 W=1mm;
```

**Notes**

- ▷ An MSUB statement should be placed before MBEND3 to provide the substrate definition.

**Reference**

- [1] E. Hammerstad, "Computer-aided design of microwave couplers with accurate discontinuity models", *IEEE MTT-S Int. Microwave Symp. Dig.* (Los Angeles, CA), 1981, pp. 54-56.

**MBEND3A microstrip optimally-chamfered 90 degree bend MBEND3A****Keywords****defaults**

W line width

**required****Form**

```
MBEND3A n1 n2 W = x1;
```

**Example**

```
MBEND3A 1 2 W=1mm;
```

**Notes**

- ▷ An MSUB statement should be placed before MBEND3A to provide the substrate definition.

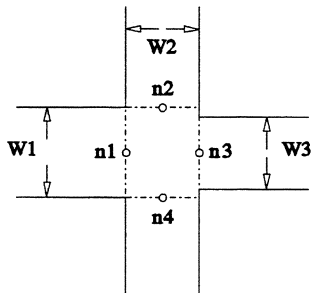
**Reference**

- [1] I.J. Bahl and P. Bhartia, *Microwave Solid State Circuit Design*, New York, NY: John Wiley & Sons, 1989.

**MCROSS**

**microstrip cross-junction**

**MCROSS**



**Keywords**

**defaults**

W1 line width at node n1  
 W2 line width at node n2 and node n4  
 W3 line width at node n3

required  
 required  
 required

**Form**

MCROSS n1 n2 n3 n4 W1 = x1 W2 = x2 W3 = x3;

**Example**

MCROSS 1 2 3 4 W1=1mm W2=1mm W3=0.7mm;

**Notes**

- ▷ An MSUB statement should be placed before MCROSS to provide the substrate definition.
- ▷ MCROSS is valid for the range of parameters:  $0.5 \leq (W1/H, W2/H, W3/H) \leq 2.0$ , where H is a substrate parameter.
- ▷ Scaling is used for  $EPSR \neq 9.9$ , where EPSR is a substrate parameter.

**References**

[1] K.C. Gupta, R. Garg and I.J. Bahl, *Microstrip Lines and Slotlines*, Dedham, MA: Artech House, 1979.

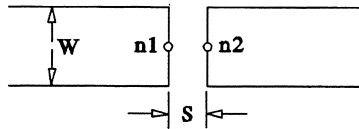
[2] K. Chang (ed.), *Handbook of Microwave and Optical Components*, vol. 1, Microwave Passive and Antenna Components, New York, NY: John Wiley & Sons, 1989.



MGAP

microstrip symmetric gap

MGAP



## Keywords

defaults

W line width  
S gap spacing

required  
required

## Form

```
MGAP n1 n2 W = x1 S = x2;
```

## Example

```
MGAP 1 2 W=0.65mm S=0.2mm;
```

## Notes

- ▷ An MSUB statement should be placed before MGAP to provide the substrate definition.
- ▷ MGAP was formulated in [1] by fitting data from [2] for the range of parameters
  - $0.5 \leq W/H \leq 2.0$
  - $1.0 \leq \text{EPSR} \leq 15$
  - $0.1 \leq S/H \leq 2.0$
 where H and EPSR are substrate parameters.

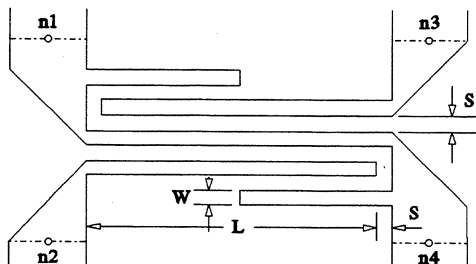
## References

- [1] E. Hammerstad, "Computer-aided design of microwave couplers with accurate discontinuity models", *IEEE MTT-S Int. Microwave Symp. Dig.* (Los Angeles, CA), 1981, pp. 54-56.
- [2] P. Benedek and P. Silvester, "Equivalent capacitances for microstrip gaps and steps", *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-20, 1972, pp. 729-733.

MLANG4

four-finger microstrip Lange coupler

MLANG4



## Keywords

W finger width  
 L finger length  
 S finger spacing

defaults

required  
 required  
 required

## Form

```
MLANG4 n1 n2 n3 n4 W = x1 L = x2 S = x3;
```

## Example

```
MLANG4 1 2 3 4 W=0.6mm L=10mm S=0.3mm;
```

## Notes

- ▷ An MSUB statement should be placed before MLANG4 to provide the substrate definition.
- ▷ MLANG4 was validated in [2] for the range of parameters
  - $0.1 \leq W/H \leq 3.0$
  - $1.0 \leq \text{EPSR} \leq 20$
  - $0.068 \leq S/H \leq 0.9$
 where H and EPSR are substrate parameters.

MLANG4

four-finger microstrip Lange coupler

MLANG4

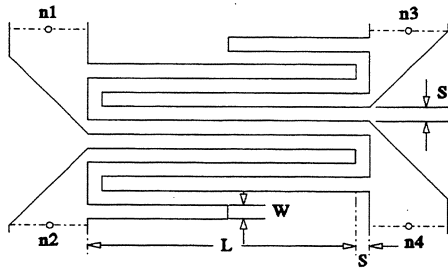
## References

- [1] D. Kajfez, Z. Paunovic and S. Pavlin, "Simplified design of Lange coupler", *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-26, 1977, pp. 806-808.
- [2] F. Sellberg, "Simple and accurate algorithms to include Lange and asymmetric microstrip couplers in CAD-packages", *Proc. 18th European Microwave Conf.* (Stockholm, Sweden), 1988, pp. 1157-1162.

MLANG6

six-finger microstrip Lange coupler

MLANG6



Keywords

W finger width  
 L finger length  
 S finger spacing

defaults

required  
 required  
 required

Form

MLANG6 n1 n2 n3 n4 W = x1 L = x2 S = x3;

Example

MLANG6 1 2 3 4 W=0.6mm L=10mm S=0.3mm;

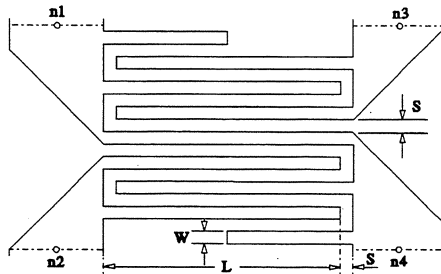
Notes

- ▷ An MSUB statement should be placed before MLANG6 to provide the substrate definition.
- ▷ MLANG6 was validated in [2] for the range of parameters
  - $0.1 \leq W/H \leq 3.0$
  - $1.0 \leq \text{EPSR} \leq 20$
  - $0.068 \leq S/H \leq 0.9$
 where H and EPSR are substrate parameters.

**MLANG6****six-finger microstrip Lange coupler****MLANG6**

## References

- [1] D. Kajfez, Z. Paunovic and S. Pavlin, "Simplified design of Lange coupler", *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-26, 1977, pp. 806-808.
- [2] F. Sellberg, "Simple and accurate algorithms to include Lange and asymmetric microstrip couplers in CAD-packages", *Proc. 18th European Microwave Conf.* (Stockholm, Sweden), 1988, pp. 1157-1162.

**MLANG8****eight-finger microstrip Lange coupler****MLANG8****Keywords**

W finger width  
 L finger length  
 S finger spacing

**defaults**

required  
 required  
 required

**Form**

```
MLANG8 n1 n2 n3 n4 W = x1 L = x2 S = x3;
```

**Example**

```
MLANG8 1 2 3 4 W=0.6mm L=10mm S=0.3mm;
```

**Notes**

- ▷ An MSUB statement should be placed before MLANG8 to provide the substrate definition.
- ▷ MLANG8 was validated in [2] for the range of parameters
  - $0.1 \leq W/H \leq 3.0$
  - $1.0 \leq \text{EPSR} \leq 20$
  - $0.068 \leq S/H \leq 0.9$
 where H and EPSR are substrate parameters.

MLANG8

eight-finger microstrip Lange coupler

MLANG8

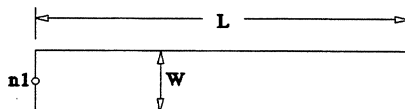
## References

- [1] D. Kajfez, Z. Paunovic and S. Pavlin, "Simplified design of Lange coupler", *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-26, 1977, pp. 806-808.
- [2] F. Sellberg, "Simple and accurate algorithms to include Lange and asymmetric microstrip couplers in CAD-packages", *Proc. 18th European Microwave Conf.* (Stockholm, Sweden), 1988, pp. 1157-1162.

MOPEN

microstrip open stub

MOPEN



## Keywords

W line width  
L line length

defaults

required  
0

## Form

```
MOPEN n1 W = x1 L = x2;
```

## Example

```
MOPEN 1 W=0.5mm L=2mm;
```

## Notes

- ▷ An MSUB statement should be placed before MOPEN to provide the substrate definition.
- ▷ The valid range of the parameters is given by [2] as
  - $0.1 \leq W/H \leq 10$
  - $1.0 \leq \text{EPSR} \leq 18$
  - $f(\text{GHz}) \cdot H(\text{mm}) \leq 30$
 where H and EPSR are substrate parameters, and f is the frequency.

## Reference

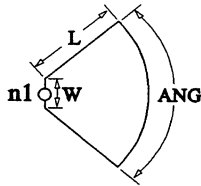
- [1] M. Kirschning, R. Jansen and N. Koster, "Accurate model for open end effect of microstrip lines", *Electronics Letters*, vol. 17, Feb. 5, 1981, pp. 123-125.
- [2] R. Jansen and M. Kirschning, "Arguments and an accurate model for the power-current formulation of microstrip characteristic impedance", *Arch. Elek. Übertragung (AEU)*, vol. 37, 1983, pp.108-112.



MRSTUB

microstrip radial stub

MRSTUB



### Keywords

W stub width at node n1  
 L stub edge length  
 ANG stub angle

defaults

required  
 required  
 required

### Form

```
MRSTUB n1 W = x1 L = x2 ANG = x3;
```

### Example

```
MRSTUB 1 W=0.5mm L=4mm ANG=35deg;
```

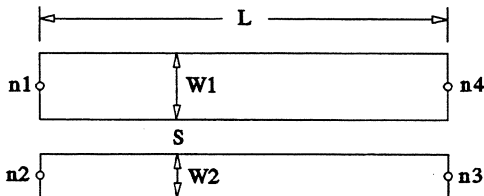
### Notes

- ▷ An MSUB statement should be placed before MRSTUB to provide the substrate definition.

MSACL

asymmetrical coupled microstrip lines

MSACL



## Keywords

W1 line width at node 1  
 W2 line width at node 2  
 L line length  
 S line spacing

## Defaults

required  
 required  
 required  
 required

## Form

```
MSACL n1 n2 n3 n4 W1 = x1 W2 = x2 L = x3 S = x4;
```

## Example

```
MSACL 1 2 3 4 W1=0.5mm W2=0.3mm L=2mm S=0.2mm;
```

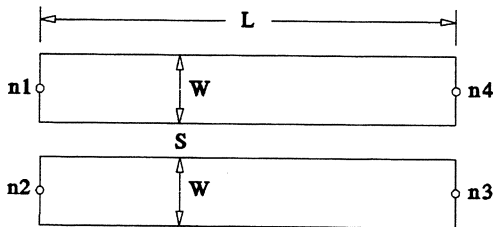
## Notes

- ▷ An MSUB statement should be placed before MSACL to provide the substrate definition.

## References

- [1] E. Hammerstad and O. Jensen, "Accurate models for microstrip computer-aided design", *IEEE MTT-S Int. Microwave Symp. Dig.* (Washington, DC), 1980, pp. 407-409.
- [2] D. Kajfez, Z. Paunovic and S. Pavlin, "Simplified design of Lange coupler", *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-26, 1977, pp. 806-808.
- [3] F. Sellberg, "Simple and accurate algorithms to include Lange and asymmetric microstrip couplers in CAD-packages", *Proc. 18th European Microwave Conf.* (Stockholm, Sweden), 1988, pp. 1157-1162.

## MSCL two-conductor symmetrical coupled microstrip lines MSCL



### Keywords

W line width  
L line length  
S line spacing

### Defaults

required  
required  
required

### Form

MSCL n1 n2 n3 n4 W = x1 L = x2 S = x3;

### Example

MSCL 1 2 3 4 W=0.5mm L=2mm S=0.2mm;

### Notes

- ▷ An MSUB statement should be placed before MSCL to provide the substrate definition.
- ▷ MSCL is accurate for the range of parameters
  - $0.1 \leq W/H \leq 10$
  - $1.0 \leq \text{EPSR} \leq 18$
  - $f(\text{GHz}) \cdot H(\text{mm}) \leq 30$
 where H and EPSR are substrate parameters, and f is the frequency.

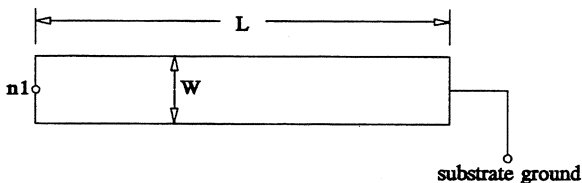
### References

- [1] S. March, "Microstrip packaging: watch the last step", *Microwaves*, Dec. 1981, pp. 83-94.
- [2] M. Kirschning and R. Jansen, "Accurate wide-range design equations for the frequency-dependent characteristics of parallel coupled microstrip lines", *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-32, 1984, pp. 83-90. Corrections: *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-33, 1985, p. 288.

MSHORT

microstrip short stub

MSHORT



## Keywords

defaults

W line width  
L line length

required  
required

## Form

```
MSHORT n1 W = x1 L = x2;
```

## Example

```
MSHORT 1 W=0.5mm L=3mm;
```

## Notes

- ▷ An MSUB statement should be placed before MSHORT to provide the substrate definition.
- ▷ The valid range of the parameters is given by [1] as
  - $0.1 \leq W/H \leq 10$
  - $1.0 \leq \text{EPSR} \leq 18$
  - $f(\text{GHz}) \cdot H(\text{mm}) \leq 30$
  - where H and EPSR are substrate parameters, and f is the frequency.
- ▷ Ideal short circuit is assumed for this component.

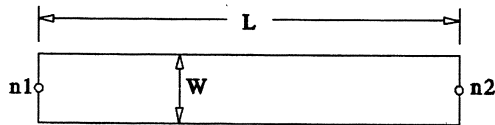
## References

- [1] R. Jansen and M. Kirschning, "Arguments and an accurate model for the power-current formulation of microstrip characteristic impedance", *Arch. Elek. Übertragung (AEU)*, vol. 37, 1983, pp.108-112.

MSL

microstrip line

MSL



## Keywords

W line width  
L line length

## Defaults

required  
required

## Form

```
MSL n1 n2 W = x1 L = x2;
```

## Example

```
MSL 1 2 W=0.5mm L=3.5mm;
```

## Notes

- ▷ An MSUB statement should be placed before MSL to provide the substrate definition.
- ▷ MSL is accurate for the range of parameters
  - $0.1 \leq W/H \leq 10$
  - $1.0 \leq \text{EPSR} \leq 18$
  - $f(\text{GHz}) \cdot H(\text{mm}) \leq 30$
 where H and EPSR are substrate parameters, and f is the frequency.

## References

- [1] E. Hammerstad and O. Jensen, "Accurate models for microstrip computer-aided design", *IEEE MTT-S Int. Microwave Symp. Dig.* (Washington, DC), 1980, pp. 407-409.
- [2] S. March, "Microstrip packaging: watch the last step", *Microwaves*, Dec. 1981, pp. 83-94.
- [3] M. Kirschning and R. Jansen, "Accurate model for effective dielectric constant of microstrip with validity up to millimetre-wave frequencies", *Electronics Letters*, vol. 18, no. 6, 1982, pp. 272-273.

MSL

microstrip line

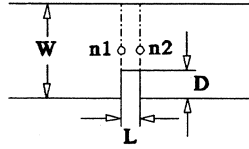
MSL

- [4] R. Jansen and M. Kirschning, "Arguments and an accurate model for the power-current formulation of microstrip characteristic impedance", *Arch. Elek. Übertragung (AEU)*, vol. 37, 1983, pp.108-112.

MSLIT

narrow transverse slit in microstrip

MSLIT



## Keywords

defaults

W line width  
 D slit depth  
 L slit length

required  
 0  
 required

## Form

```
MSLIT n1 n2 W = x1 D = x2 L = x3;
```

## Example

```
MSLIT 1 2 W=0.5mm D=0.15mm L=0.1mm;
```

## Notes

- ▷ An MSUB statement should be placed before MSLIT to provide the substrate definition.
- ▷ MSLIT is valid for the range of parameters
  - $0.1 \leq W/H \leq 10$
  - $0.0 \leq D/W \leq 0.9$
  - $1.0 \leq \text{EPSR} \leq 18$
  - $f(\text{GHz}) \cdot H(\text{mm}) \leq 30$
  - where H and EPSR are substrate parameters, and f is the frequency.
- ▷ L should be smaller than H and should be much smaller than the propagation wavelength.

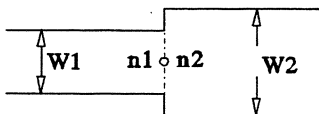
## References

- [1] W.J.R. Hofer, "Theoretical and experimental characterization of narrow transverse slits in microstrip", *NTZ*, Bd. 30, 1977, pp. 582-585.
- [2] W.J.R. Hofer, "Equivalent series inductivity of a narrow transverse slit in microstrip", *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-25, 1977, pp. 822-824.

MSTEP

microstrip step

MSTEP



## Keywords

W1 line width at node n1  
 W2 line width at node n2

defaults

required  
 required

## Form

```
MSTEP n1 n2 W1 = x1 W2 = x2;
```

## Example

```
MSTEP 1 2 W1=0.6mm W2=1mm;
```

## Notes

- ▷ An MSUB statement should be placed before MSTEP to provide the substrate definition.

## References

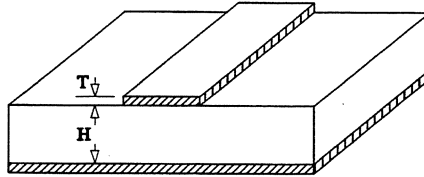
- [1] G. Kompas, "Design of stepped microstrip components", *The Radio and Electronic Engineer*, vol. 48, no. 1/2, Jan./Feb. 1978, pp.53-63.
- [2] I.J. Bahl and P. Bhartia, *Microwave Solid State Circuit Design*, New York, NY: John Wiley & Sons, 1989.



MSUB

microstrip substrate definition

MSUB



## Keywords

EPSR dielectric constant of the substrate  
 H substrate height  
 H2 shielding cover height  
 T conducting metal thickness  
 TAND substrate dielectric loss tangent  
 ROC resistivity of the conducting strip in  $\Omega\text{m}$   
 RHS surface roughness of the conducting strip in m

## Defaults

2.35  
 0.635mm  
 $\infty$   
 0  
 0  
 0  
 0

## Form

MSUB [n] EPSR = x1 H = x2 H2 = x3 T = x4 TAND = x5 ROC = x6 RHS = x7;

## Example

MSUB EPSR=9.7 H=0.635mm T=0.006mm TAND=0.0002 ROC=0.0172E-6 RHS=0.5E-6;

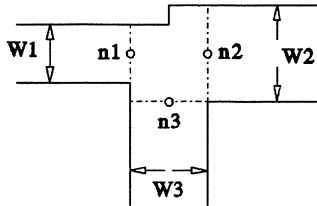
## Notes

- ▷ MSUB is required to define the substrate information for microstrip components.
- ▷ MSUB should be placed before any microstrip components.
- ▷ Multiple MSUB can be used in one circuit file with the current one overriding the previous one.
- ▷ The node n is used to define the connection of the ground plane. If not defined, the ground plane of the substrate is by default connected to the global ground node 0.

MTEE

microstrip T-junction

MTEE



Keywords

W1 line width at node n1  
 W2 line width at node n2  
 W3 line width at node n3

defaults

required  
 required  
 required

Form

MTEE n1 n2 n3 W1 = x1 W2 = x2 W3 = x3;

Example

MTEE 1 2 3 W1=0.65mm W2=0.65mm W3=0.5mm;

Notes

- ▷ An MSUB statement should be placed before MTEE to provide the substrate definition.

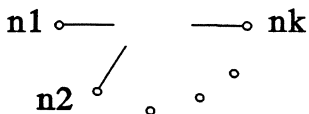
Reference

- [1] E. Hammerstad, "Computer aided design of microstrip couplers with accurate discontinuity models", *IEEE MTT-S Int. Microwave Symp. Dig.* (Los Angeles, CA), 1981, pp. 54-56.

OPEN

open circuit

OPEN

**Form**

```
OPEN n1 [n2 ... nk];
```

**Example**

```
OPEN 1 7;
```

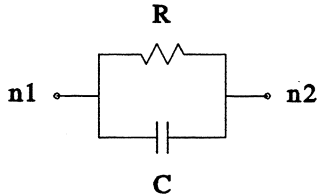
**Notes**

- ▷ OPEN is typically used to mark intentionally introduced floating nodes, for example, at the open end of a transmission line. Without it, the circuit file parser will consider floating nodes an error.
- ▷ If the circuit contains more than one open-circuited node, one OPEN element is sufficient to include all such nodes (although multiple OPEN elements are allowed).

PRC

parallel connection of resistor and capacitor

PRC



## Keywords

R resistance  
C capacitance

## Defaults

$\infty$   
0

## Form

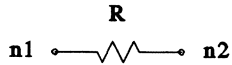
```
PRC n1 [n2] R = x1 C = x2;
```

## Example

```
PRC 1 2 R=3.5 C=100PF;
```

## Notes

- ▷ R, if specified, must be greater than 0.
- ▷ If the keyword R or C is missing the element defaults to the CAP or RES element, respectively.

**RES****resistor****RES****Keywords**

R resistance

**Defaults**

0.01ohm

**Form**

```
RES n1 [n2] R = x1;
```

**Example**

```
RES .1 2 R=3.5;
```

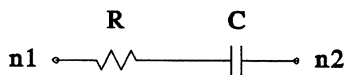
**Notes**

- ▷ R, if specified, must be greater than 0.

SRC

series connection of resistor and capacitor

SRC



## Keywords

R resistance  
C capacitance

## Defaults

0.01ohm  
 $\infty$

## Form

```
SRC n1 [n2] R = x1 C = x2;
```

## Example

```
SRC 1 2 R=3.5 C=100PF;
```

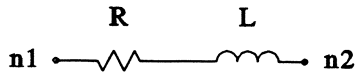
## Notes

- ▷ If the keyword C is missing the element defaults to the RES element.

SRL

series connection of resistor and inductor

SRL

**Keywords**

R resistance  
L inductance

**Defaults**

0.01ohm  
0

**Form**

```
SRL n1 [n2] R = x1 L = x2;
```

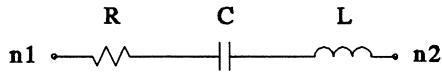
**Example**

```
SRL 1 2 R=3.5 L=10UH;
```

**Notes**

- ▷ R, if specified, must be greater than 0.
- ▷ If the keyword L is missing the element defaults to the RES element.

**SRLC series connection of resistor, inductor and capacitor SRLC**



**Keywords**

R resistance  
 L inductance  
 C capacitance

**Defaults**

0.01ohm  
 0  
 ∞

**Form**

SRLC n1 [n2] R = x1 L = x2 C = x3;

**Example**

SRLC 1 2 R=3.5 L=10UH C=100PF;

**Notes**

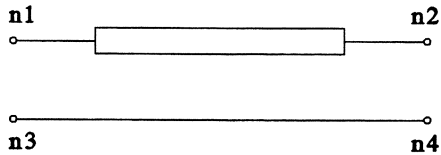
- ▷ R, if specified, must be greater than 0.
- ▷ If the keyword L or C is missing the element defaults to the SRC or SRL element, respectively.



TEM

ideal transmission line

TEM



### Keywords

Z characteristic impedance  
 E electrical length  
 F frequency at which E is specified

### Defaults

50ohm  
 90deg  
 1GHz

### Form

```
TEM n1 [n2 n3 n4] Z = x1 E = x2 F = x3;
```

### Example

```
TEM 1 2 Z=45 E=1RAD F=10GHZ;
```

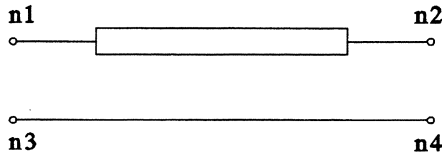
### Notes

- ▷ Z, E and F, if specified, must be greater than 0.
- ▷ The default unit for electrical length is degree.

TRL

transmission line, physical model

TRL



### Keywords

Z characteristic impedance  
 L physical length  
 K effective dielectric constant  
 A attenuation per unit length  
 F frequency at which A is specified

### Defaults

50ohm  
 0.01m  
 1  
 0  
 0

### Form

TRL n1 [n2 n3 n4] Z = x1 L = x2 K = x3 A = x4 F = x5;

### Example

TRL 1 2 Z=45 L=15MM K=9.7 A=3 F=10GHZ;

### Notes

- ▷ Z and L, if specified, must be greater than 0.
- ▷ K, if specified, must be greater than or equal to 1.
- ▷ A and F, if specified, must be greater than or equal to 0.
- ▷ Attenuation A is scaled with the square root of the current frequency  $f$  relative to the frequency F, i.e.,

$$A(f) = A \cdot \sqrt{f/F} \quad \text{if } F > 0,$$

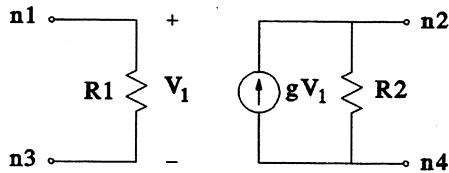
and is constant with frequency if  $F = 0$  or is not specified.

- ▷ The default unit for attenuation is dB per meter.

VCCS

voltage controlled current source

VCCS



## Keywords

R1	input resistance
R2	output resistance
M	DC value of the transadmittance $g$
A	angle of the transadmittance $g$
F	3 dB cut-off frequency of the transadmittance $g$
T	time delay of the transadmittance $g$

## Defaults

$\infty$   
 $\infty$   
 0  
 0  
 $\infty$   
 0

## Form

VCCS n1 [n2 n3 n4] R1 = x1 R2 = x2 M = x3 A = x4 F = x5 T = x6;

## Example

VCCS 1 2 3 4 R1=50KOH R2=5KOH M=50;

## Notes

- ▷ The transadmittance  $g$  is evaluated as

$$g = M \cdot \exp(j(A - 2\pi fT)) / (1 + jf/F)$$

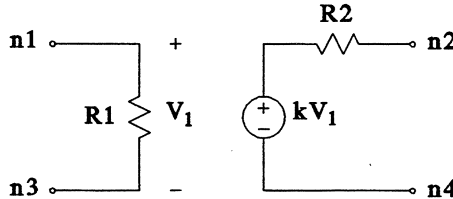
where  $M$ ,  $A$  (converted to radians, if necessary),  $T$  and  $F$  are model parameters, and  $f$  is the current frequency.

- ▷  $F$ , if specified, must be greater than 0.
- ▷ An ideal VCCS is defined by the default values for  $R1$  and  $R2$ .
- ▷ A source of reverse polarity can be specified with a negative value for  $M$ .

VCVS

voltage controlled voltage source

VCVS



Keywords

Defaults

R1	input resistance	$\infty$
R2	output resistance	0
M	DC value of the controlling coefficient k	0
A	angle of the controlling coefficients	0
F	3 dB cut-off frequency of the controlling coefficient	$\infty$
T	time delay of the controlling coefficient	0

Form

VCVS n1 [n2 n3 n4] R1 = x1 R2 = x2 M = x3 A = x4 F = x5 T = x6;

Example

VCVS 1 2 3 4 R1=50KOH R2=50 M=10;

Notes

- ▷ The controlling coefficient k is evaluated as

$$k = M \cdot \exp(j(A - 2\pi fT)) / (1 + jf/F)$$

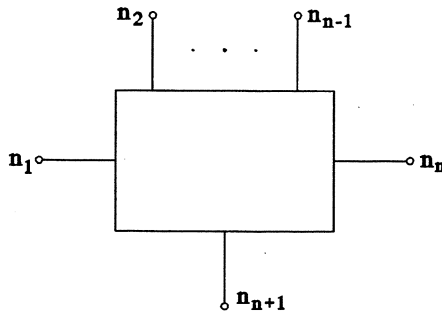
where M, A (converted to radians, if necessary), T and F are model parameters, and  $f$  is the current frequency.

- ▷ F, if specified, must be greater than 0.
- ▷ An ideal VCVS is defined by the default values for R1 and R2.
- ▷ A source of reverse polarity can be specified with a negative value for M.

DATAPORT

imported  $n$ -port subcircuit

DATAPORT



## Keywords

DATA     name of the imported data

## Defaults

required

## Form

```
DATAPORT n1 ... nn [n_{n+1}] DATA = name;
```

## Examples

```
DATAPORT 1 0 DATA=import_impedance;
```

```
DATAPORT 1 2 3 DATA=import_2port;
```

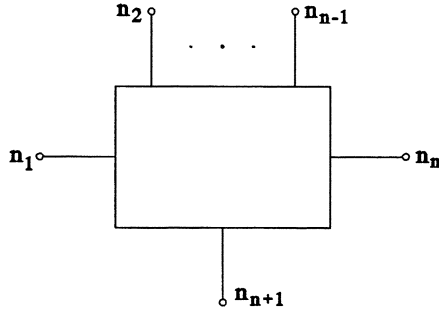
## Notes

- ▷ The number of ports  $n$  must be between 1 and 31.
- ▷ The name specified after the keyword DATA must have been defined in the ImportData block (see Chapter 6).
- ▷ The imported data must represent an  $n$ -port with the  $(n+1)$ th node being the common node for all ports (ground, or reference node), i.e., the data must be an  $n \times n$   $S$ ,  $Y$  or  $Z$  matrix. When the data is used as a DATAPORT element the reference node, i.e., the  $(n+1)$ th node, can be connected to a nonzero node.

SPORT

$n$ -port subcircuit defined by  $S$  matrix

SPORT



Keywords

Defaults

DATA	$S$ matrix at a single frequency	none
REAL	real part of $S$ matrix at a single frequency	0
IMAG	imaginary part of $S$ matrix at a single frequency	0
FMP	$S$ matrix at multiple frequencies, in magnitude and phase	none
FRI	$S$ matrix at multiple frequencies, in real and imaginary parts	none
DC	$S$ matrix at DC	none

Notes

- ▷ The number of ports must be between 1 and 31.
- ▷ The imported data must represent an  $n$ -port with the  $(n+1)$ th node being the common node for all ports (ground, or reference node). When the data is used as an SPORT element the reference node, i.e., the  $(n+1)$ th node, can be connected to a nonzero node.
- ▷ The reference impedance for the  $S$  parameters specified through an SPORT element is assumed to be 50 ohm. Please make sure that your  $S$  parameters conform to this requirement.
- ▷ The format of SPORT is very flexible. The  $S$  parameters can be supplied for each individual frequency or for all the frequencies at once. The data can be in polar or rectangular format, and the DC and AC data can be specified separately. The different variations of data format are described separately as follows.

**SPORT****n-port subcircuit defined by S matrix****SPORT****Format 1: S Matrix at a Single Frequency**

```
SPORT n1 ... nn [nn+1] DATA = array_name;
```

**Example**

```
SPORT 1 2 3 DATA=S_Array;
```

**Notes**

- ▷ Array\_name must refer to a  $2 \times n \times n$  array (see Chapter 4 for further details on arrays).
- ▷ It is assumed that the array contains the S parameters in the following order:

```
RS11 IS11 RS12 IS12 ... RS1n IS1n
RS21 IS21 RS22 IS22 ... RS2n IS2n
...
RSn1 ISn1 RSn2 ISn2 ... RSn n ISn n
```

SPORT

n-port subcircuit defined by S matrix

SPORT

## Format 2: Separate Real and Imaginary Matrices at a Single Frequency

```
SPORT n1 ... nn [nn+1] REAL = array1 IMAG = array2;
```

## Example

```
SPORT 1 2 3 REAL=S_real IMAG=S_imag;
```

## Notes

- ▷ The arrays identified by `array1` and `array2` must be of the size  $n \times n$ .
- ▷ It is assumed that `array1` contains the real part of the  $S$  parameters in the following order:

```
RS11 RS12 ... RS1n
RS21 RS22 ... RS2n
...
```

```
RSn1 RSn2 ... RSn
```

and `array2` contains the imaginary part of the  $S$  parameters in the following order:

```
IS11 IS12 ... IS1n
IS21 IS22 ... IS2n
...
```

```
ISn1 ISn2 ... ISn
```

- ▷ If the keyword `REAL` is omitted, then the real part of the  $S$  parameters is zero by default, i.e., the  $n$ -port is assumed to be purely reactive (lossless).
- ▷ If the keyword `IMAG` is omitted, then the imaginary part of the  $S$  parameters is zero by default, i.e., the  $n$ -port is assumed to be purely resistive.



SPORT

n-port subcircuit defined by S matrix

SPORT

### Format 3: S Matrix for Multiple Frequencies, in Magnitude and Phase

```
SPORT n1 ... nn [nn+1] FMP = array_name;
```

#### Example

```
SPORT 1 2 3 FMP=S_data;
```

#### Notes

- ▷ The array identified by `array_name` must contain a set of frequencies and for each frequency the  $n$ -port  $S$  parameters in polar form (i.e., magnitude and phase).
- ▷ The number of rows in the array is the number of different frequencies, and the number of columns must be  $2 \times n \times n + 1$ . For example, if the array is defined as `S_Data[20,9]`, then it is assumed to contain the  $S$  matrix of a 2-port for 20 different frequencies, and the data is in the following order:

```
f1  MS11  PS11  MS12  PS12  MS21  PS21  MS22  PS22
f2  MS11  PS11  MS12  PS12  MS21  PS21  MS22  PS22
...  ...
f20 MS11  PS11  MS12  PS12  MS21  PS21  MS22  PS22
```

- ▷ If a frequency involved in circuit simulation is not included in the supplied data, OSA90 will perform linear interpolations.

**SPORT****n-port subcircuit defined by S matrix****SPORT****Format 4: S Matrix for Multiple Frequencies, in Real and Imaginary Parts**

```
SPORT n1 ... nn [nn+1] FRI = array_name;
```

**Example**

```
SPORT 1 2 3 FRI=S_data;
```

**Notes**

- ▷ The array identified by `array_name` must contain a set of frequencies and for each frequency the  $n$ -port  $S$  parameters in rectangular form (i.e., real and imaginary parts).
- ▷ The number of rows in the array is the number of different frequencies, and the number of columns must be  $2n \times n + 1$ . For example, if the array is defined as `S_Data[20,9]`, then it is assumed to contain the  $S$  matrix of a 2-port for 20 different frequencies, and the data is in the following order:

```
f1  RS11  IS11  RS12  IS12  RS21  IS21  RS22  IS22
f2  RS11  IS11  RS12  IS12  RS21  IS21  RS22  IS22
...  ...
f20 RS11  IS11  RS12  IS12  RS21  IS21  RS22  IS22
```

- ▷ If a frequency involved in circuit simulation is not included in the supplied data, OSA90 will perform linear interpolations.

SPORT

**n-port subcircuit defined by S matrix**

SPORT

**Format 5: Supplying AC and DC Data Separately**

```
SPORT n1 ... nn [nn+1] FMP = array_ac DC = array_dc;
```

```
SPORT n1 ... nn [nn+1] FRI = array_ac DC = array_dc;
```

**Example**

```
SPORT 1 2 3 FMP=S_ac DC=S_dc;
```

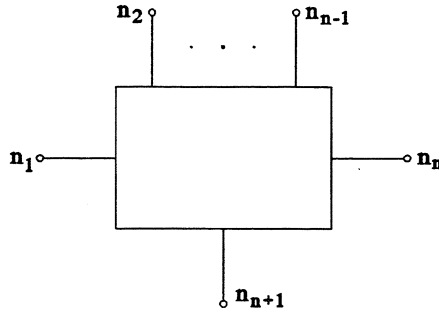
**Notes**

- ▷ The array identified by `array_ac` must contain a set of frequencies and for each frequency the  $n$ -port  $S$  parameters in polar or rectangular form.
- ▷ The number of rows in the `array_ac` is the number of different frequencies, and the number of columns must be  $2 \times n \times n + 1$ . For example, if the array is defined as `S_ac[20,9]`, then it is assumed to contain 20 different frequencies and the  $S$  matrix of a 2-port for those frequencies.
- ▷ It is assumed that the frequencies contained in `array_ac` do not include DC. The DC data is supplied separately in `array_dc`. The size of `array_dc` must be  $n \times n$ , since the  $S$  parameters for DC are real numbers.

YPORT

**n-port subcircuit defined by Y matrix**

YPORT



**Keywords**

**Defaults**

DATA	Y matrix at a single frequency	none
REAL	real part of Y matrix at a single frequency	0
IMAG	imaginary part of Y matrix at a single frequency	0
FMP	Y matrix at multiple frequencies, in magnitude and phase	none
FRI	Y matrix at multiple frequencies, in real and imaginary parts	none
DC	Y matrix at DC	none

**Notes**

- ▷ The number of ports must be between 1 and 31.
- ▷ The imported data must represent an  $n$ -port with the  $(n+1)$ th node being the common node for all ports (ground, or reference node). When the data is used as a YPORT element the reference node, i.e., the  $(n+1)$ th node, can be connected to a nonzero node.
- ▷ The format of YPORT is very flexible. The Y parameters can be supplied for each individual frequency or for all the frequencies at once. The data can be in polar or rectangular format, and the DC and AC data can be specified separately. The different variations of data format are described separately as follows.

**YPORT****n-port subcircuit defined by Y matrix****YPORT****Format 1: Y Matrix at a Single Frequency**

```
YPORT n1 ... nn [nn+1] DATA = array_name;
```

**Example**

```
YPORT 1 2 3 DATA=Y_Array;
```

**Notes**

- ▷ **Array\_name** must refer to a  $2 \times n \times n$  array (see Chapter 4 for further details on arrays).
- ▷ It is assumed that the array contains the *Y* parameters in the following order:

```
RY11  IY11  RY12  IY12  ...  RY1n  IY1n
RY21  IY21  RY22  IY22  ...  RY2n  IY2n
...
RYn1  IYn1  RYn2  IYn2  ...  RYnn  IYnn
```

YPORT

n-port subcircuit defined by Y matrix

YPORT

## Format 2: Separate Real and Imaginary Matrices at a Single Frequency

```
YPORT n1 ... nn [nn+1] REAL = array1 IMAG = array2;
```

## Example

```
YPORT 1 2 3 REAL=Y_real IMAG=Y_imag;
```

## Notes

- ▷ The arrays identified by `array1` and `array2` must be of the size  $n \times n$ .
- ▷ It is assumed that `array1` contains the real part of the  $Y$  parameters in the following order:

```
RY11  RY12  ...  RY1n
RY21  RY22  ...  RY2n
...
```

```
RYn1  RYn2  ...  RYnn
```

and `array2` contains the imaginary part of the  $Y$  parameters in the following order:

```
IY11  IY12  ...  IY1n
IY21  IY22  ...  IY2n
...
```

```
IYn1  IYn2  ...  IYnn
```

- ▷ If the keyword `REAL` is omitted, then the real part of the  $Y$  parameters is zero by default, i.e., the  $n$ -port is assumed to be purely reactive (lossless).
- ▷ If the keyword `IMAG` is omitted, then the imaginary part of the  $Y$  parameters is zero by default, i.e., the  $n$ -port is assumed to be purely resistive.

YPORT

n-port subcircuit defined by Y matrix

YPORT

### Format 3: Y Matrix for Multiple Frequencies, in Magnitude and Phase

```
YPORT n1 ... nn [nn+1] FMP = array_name;
```

### Example

```
YPORT 1 2 3 FMP=Y_data;
```

### Notes

- ▷ The array identified by `array_name` must contain a set of frequencies and for each frequency the  $n$ -port  $Y$  parameters in polar form (i.e., magnitude and phase).
- ▷ The number of rows in the array is the number of different frequencies, and the number of columns must be  $2 \times n \times n + 1$ . For example, if the array is defined as `Y_Data[20,9]`, then it is assumed to contain the  $Y$  matrix of a 2-port for 20 different frequencies, and the data is in the following order:

```
f1 MY11 PY11 MY12 PY12 MY21 PY21 MY22 PY22
f2 MY11 PY11 MY12 PY12 MY21 PY21 MY22 PY22
... ...
f20 MY11 PY11 MY12 PY12 MY21 PY21 MY22 PY22
```

- ▷ If a frequency involved in circuit simulation is not included in the supplied data, OSA90 will perform linear interpolations.

YPORT

n-port subcircuit defined by Y matrix

YPORT

## Format 4: Y Matrix for Multiple Frequencies, in Real and Imaginary Parts

```
YPORT n1 ... nn [nn+1] FRI = array_name;
```

## Example

```
YPORT 1 2 3 FRI=Y_data;
```

## Notes

- ▷ The array identified by `array_name` must contain a set of frequencies and for each frequency the n-port  $Y$  parameters in rectangular form (i.e., real and imaginary parts).
- ▷ The number of rows in the array is the number of different frequencies, and the number of columns must be  $2 \times n \times n + 1$ . For example, if the array is defined as `Y_Data[20,9]`, then it is assumed to contain the  $Y$  matrix of a 2-port for 20 different frequencies, and the data is in the following order:

```
f1  RY11  IY11  RY12  IY12  RY21  IY21  RY22  IY22
f2  RY11  IY11  RY12  IY12  RY21  IY21  RY22  IY22
...  ...
f20 RY11  IY11  RY12  IY12  RY21  IY21  RY22  IY22
```

- ▷ If a frequency involved in circuit simulation is not included in the supplied data, OSA90 will perform linear interpolations.



YPORT

n-port subcircuit defined by Y matrix

YPORT

## Format 5: Supplying AC and DC Data Separately

```
YPORT n1 ... nn [nn+1] FMP = array_ac DC = array_dc;
```

```
YPORT n1 ... nn [nn+1] FRI = array_ac DC = array_dc;
```

## Example

```
YPORT 1 2 3 FMP=Y_ac DC=Y_dc;
```

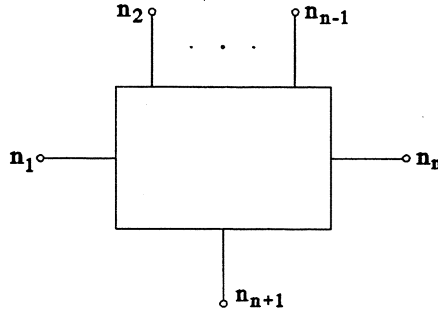
## Notes

- ▷ The array identified by `array_ac` must contain a set of frequencies and for each frequency the n-port  $Y$  parameters in polar or rectangular form.
- ▷ The number of rows in the `array_ac` is the number of different frequencies, and the number of columns must be  $2 \times n \times n + 1$ . For example, if the array is defined as `Y_ac[20,9]`, then it is assumed to contain 20 different frequencies and the  $Y$  matrix of a 2-port for those frequencies.
- ▷ It is assumed that the frequencies contained in `array_ac` do not include DC. The DC data is supplied separately in `array_dc`. The size of `array_dc` must be  $n \times n$ , since the  $Y$  parameters for DC are real numbers.

ZPORT

**n-port subcircuit defined by Z matrix**

ZPORT



## Keywords

## Defaults

DATA	Z matrix at a single frequency	none
REAL	real part of Z matrix at a single frequency	0
IMAG	imaginary part of Z matrix at a single frequency	0
FMP	Z matrix at multiple frequencies, in magnitude and phase	none
FRI	Z matrix at multiple frequencies, in real and imaginary parts	none
DC	Z matrix at DC	none

## Notes

- ▷ The number of ports must be between 1 and 31.
- ▷ The imported data must represent an  $n$ -port with the  $(n+1)$ th node being the common node for all ports (ground, or reference node). When the data is used as a ZPORT element the reference node, i.e., the  $(n+1)$ th node, can be connected to a nonzero node.
- ▷ The format of ZPORT is very flexible. The Z parameters can be supplied for each individual frequency or for all the frequencies at once. The data can be in polar or rectangular format, and the DC and AC data can be specified separately. The different variations of data format are described separately as follows.

**ZPORT****n-port subcircuit defined by Z matrix****ZPORT****Format 1: Z Matrix at a Single Frequency**

```
ZPORT n1 ... nn [nn+1] DATA = array_name;
```

**Example**

```
ZPORT 1 2 3 DATA=Z_Array;
```

**Notes**

- ▷ Array\_name must refer to a  $2 \times n \times n$  array (see Chapter 4 for further details on arrays).
- ▷ It is assumed that the array contains the Z parameters in the following order:

```
RZ11  IZ11  RZ12  IZ12  ...  RZ1n  IZ1n
RZ21  IZ21  RZ22  IZ22  ...  RZ2n  IZ2n
...
RZn1  IZn1  RZn2  IZn2  ...  RZnn  IZnn
```

ZPORT

n-port subcircuit defined by Z matrix

ZPORT

## Format 2: Separate Real and Imaginary Matrices at a Single Frequency

```
ZPORT n1 ... nn [nn+1] REAL = array1 IMAG = array2;
```

### Example

```
ZPORT 1 2 3 REAL=Z_real IMAG=Z_imag;
```

### Notes

- ▷ The arrays identified by `array1` and `array2` must be of the size  $n \times n$ .
- ▷ It is assumed that `array1` contains the real part of the  $Z$  parameters in the following order:

```
RZ11 RZ12 ... RZ1n
RZ21 RZ22 ... RZ2n
...
```

```
RZn1 RZn2 ... RZnn
```

and `array2` contains the imaginary part of the  $Z$  parameters in the following order:

```
IZ11 IZ12 ... IZ1n
IZ21 IZ22 ... IZ2n
...
```

```
IZn1 IZn2 ... IZnn
```

- ▷ If the keyword `REAL` is omitted, then the real part of the  $Z$  parameters is zero by default, i.e., the  $n$ -port is assumed to be purely reactive (lossless).
- ▷ If the keyword `IMAG` is omitted, then the imaginary part of the  $Z$  parameters is zero by default, i.e., the  $n$ -port is assumed to be purely resistive.

**ZPORT****n-port subcircuit defined by Z matrix****ZPORT****Format 3: Z Matrix for Multiple Frequencies, in Magnitude and Phase**

```
ZPORT n1 ... nn [nn+1] FMP = array_name;
```

**Example**

```
ZPORT 1 2 3 FMP=Z_data;
```

**Notes**

- ▷ The array identified by `array_name` must contain a set of frequencies and for each frequency the  $n$ -port  $Z$  parameters in polar form (i.e., magnitude and phase).
- ▷ The number of rows in the array is the number of different frequencies, and the number of columns must be  $2 \times n \times n + 1$ . For example, if the array is defined as `Z_Data[20,9]`, then it is assumed to contain the  $Z$  matrix of a 2-port for 20 different frequencies, and the data is in the following order:

```
f1  MZ11  PZ11  MZ12  PZ12  MZ21  PZ21  MZ22  PZ22
f2  MZ11  PZ11  MZ12  PZ12  MZ21  PZ21  MZ22  PZ22
...  ...
f20 MZ11  PZ11  MZ12  PZ12  MZ21  PZ21  MZ22  PZ22
```

- ▷ If a frequency involved in circuit simulation is not included in the supplied data, OSA90 will perform linear interpolations.

**ZPORT****n-port subcircuit defined by Z matrix****ZPORT****Format 4: Z Matrix for Multiple Frequencies, in Real and Imaginary Parts**

```
ZPORT n1 ... nn [nn+1] FRI = array_name;
```

**Example**

```
ZPORT 1 2 3 FRI=Z_data;
```

**Notes**

- ▷ The array identified by `array_name` must contain a set of frequencies and for each frequency the  $n$ -port  $Z$  parameters in rectangular form (i.e., real and imaginary parts).
- ▷ The number of rows in the array is the number of different frequencies, and the number of columns must be  $2n \times n + 1$ . For example, if the array is defined as `Z_Data[20,9]`, then it is assumed to contain the  $Z$  matrix of a 2-port for 20 different frequencies, and the data is in the following order:

```
f1  RZ11  IZ11  RZ12  IZ12  RZ21  IZ21  RZ22  IZ22
f2  RZ11  IZ11  RZ12  IZ12  RZ21  IZ21  RZ22  IZ22
...  ...
f20 RZ11  IZ11  RZ12  IZ12  RZ21  IZ21  RZ22  IZ22
```

- ▷ If a frequency involved in circuit simulation is not included in the supplied data, OSA90 will perform linear interpolations.

ZPORT

n-port subcircuit defined by Z matrix

ZPORT

### Format 5: Supplying AC and DC Data Separately

```
ZPORT n1 ... nn [nn+1] FMP = array_ac DC = array_dc;
```

```
ZPORT n1 ... nn [nn+1] FRI = array_ac DC = array_dc;
```

### Example

```
ZPORT 1 2 3 FMP=Z_ac DC=Z_dc;
```

### Notes

- ▷ The array identified by `array_ac` must contain a set of frequencies and for each frequency the  $n$ -port  $Z$  parameters in polar or rectangular form.
- ▷ The number of rows in the `array_ac` is the number of different frequencies, and the number of columns must be  $2 \times n \times n + 1$ . For example, if the array is defined as `Z_ac[20,9]`, then it is assumed to contain 20 different frequencies and the  $Z$  matrix of a 2-port for those frequencies.
- ▷ It is assumed that the frequencies contained in `array_ac` do not include DC. The DC data is supplied separately in `array_dc`. The size of `array_dc` must be  $n \times n$ , since the  $Z$  parameters for DC are real numbers.





# 9

## Simulation

9.1 Overview .....	9-1
9.2 Sweep Block .....	9-1
9.3 Simulation Types .....	9-3
9.4 Parameter Sweeps .....	9-7
9.5 Parameter Labels .....	9-13
9.6 Output Labels .....	9-14
9.7 OSA90 Display Menu .....	9-15
9.8 Xsweep Display .....	9-17
9.9 Numerical Display .....	9-22
9.10 Graphics Zoom .....	9-24
9.11 Draw Types .....	9-27
9.12 Parametric Display .....	9-28
9.13 Array Display .....	9-31
9.14 Visualization and Contour Plotting .....	9-32
9.15 Host Acceleration Files .....	9-43
9.16 Report Generation .....	9-44



## 9

## Simulation

## 9.1 Overview

Simulation refers to the calculation of responses and functions for graphical display and numerical output.

This chapter discusses the input file Sweep block and the “Display” menu option.

The Sweep block allows you to define simulation types, sweep parameters, sweep ranges and output labels.

The “Display” menu allows you to select different display formats and options.

## 9.2 Sweep Block

Syntax:

```
Sweep
  sweep_set;
  sweep_set;
  ...
  sweep_set;
End
```

where each *sweep\_set* is a statement delimited by a semicolon “;”.

Syntax of sweep sets:

```
type: sweep_label_1: sweep_range
      sweep_label_2: sweep_range
      label=expression label=expression ... label=expression
      output_label output_label ... output_label;
```

where *sweep\_label\_1* and *sweep\_label\_2* will sweep through multiple values defined by the corresponding *sweep\_range*, each *label* will be assigned a value given by the corresponding *expression*, and each *output\_label* will be calculated and displayed.

Example:

```
Sweep
HB:  Signal Power: -10dBm -5dBm 0dBm
      FREQ: from 5GHz to 20GHz step=1GHz
      Omega=(2.0 * PI * FREQ)
      Bias_Voltage=0.75V
      Gain, Efficiency;
End
```

**Simulation type:** if the calculation of output labels involves circuit simulation, you must also indicate the simulation type (DC, small-signal AC or harmonic balance). In the above example, the simulation type is HB (harmonic balance). You can omit the simulation type if the sweep set does not require circuit analysis.

**Sweep labels** represent parameter sweeps (i.e., simulation loops). A sweep label loops through a set of (multiple) values defined in the sweep range. One or two independent sweep labels can be defined for each sweep set. Two sweep labels are defined in the above example, namely Signal\_Power and FREQ.

**Parameter labels** can also be defined in addition to sweep labels. A parameter label is assigned a single value or an expression which may or may not depend on the sweep labels. The above example has two parameter labels, namely Omega and Bias\_Voltage. Omega is given by an expression which is a function of the sweep label FREQ. Bias\_Voltage is assigned a single value.

**Output labels:** you are required to specify a set of output labels for each sweep set to represent the responses and functions which you wish to be calculated and displayed. In the above example, the output labels are Gain and Efficiency.

**Views:** sweep sets may also include definitions of user-defined display formats called “Graphical Views” which are discussed in Chapter 10.

## 9.3 Simulation Types

To calculate and display output labels which represent circuit responses or functions of circuit responses, you must indicate the type of circuit simulation using one of the keywords listed in Table 9.1.

TABLE 9.1 SIMULATION TYPES

Keyword	Type of Simulation
DC	DC
AC	small-signal AC
HB	large-signal harmonic balance

The simulation type keyword, followed by a colon, must be the first entry in a sweep set:

DC: ..... ;

AC: ..... ;

HB: ..... ;

### Built-In Circuit Responses

The simulation type determines which set of built-in circuit responses will be calculated. For instance, if the keyword DC is specified for a sweep set, then DC responses will be calculated for that sweep set.

➤ DC, small-signal and large-signal circuit responses are described in Chapter 6.

The output labels of a sweep set can involve only one type of circuit responses. For example, if the simulation type of a sweep set is DC, then the output labels of that set may include DC responses and functions of DC responses, but not small-signal or harmonic balance responses.

### Generic Numerical Simulation

If a sweep set does not begin with a simulation type keyword, then generic numerical simulation is assumed. All the output labels must be defined by mathematical expressions and/or Datapipe. In other words, the output labels cannot be functions of circuit responses.

## Frequency Labels

For AC and HB simulation types, you must specify the frequency values by means of the label `FREQ`.

For AC, `FREQ` specifies the frequency for the small-signal circuit analysis. For HB, `FREQ` specifies the fundamental frequency of the AC sources. During DC analysis, the value of `FREQ` is automatically set to zero.

You can sweep the label `FREQ` (see Section 9.4):

```
FREQ: from 2GHZ to 16GHZ step=1GHZ
```

or, you can assign it a single value, such as

```
FREQ = 7GHZ
```

## Two-Tone Frequencies

In the case of two-tone excitations, `FREQ` specifies the first tone frequency, and the second tone frequency is specified by a keyword `FREQ2`.

Example:

```
HB: FREQ=900MHZ FREQ2=907MHZ ...;
```

where the tone frequencies are specified as 900MHZ and 907MHZ.

The keyword `FREQ2` cannot be directly defined as a sweep label. However, you can still sweep the second tone frequency, by tying `FREQ2` to another label:

```
Second_Tone_Freq: from 1GHZ to 5GHZ step=1GHZ
FREQ2 = Second_Tone_Freq
```

The following example ties the two tone frequencies together:

```
FREQ: from 1GHZ to 15GHZ step=2GHZ
FREQ2 = (FREQ + IM_Freq)
```

where `IM_Freq` is a user-defined label representing the difference between the two tone frequencies.

See Chapter 6 for a discussion on the generation of intermodulation frequencies in two-tone simulation.

## The HARM Keyword

In the CIRCUIT statement in the Model block, you can specify the highest harmonic to be included in harmonic balance simulation (see Chapter 6 for further details).

### Syntax:

```
CIRCUIT HARM=m ...;
```

*m* is the highest harmonic to be included in harmonic balance simulation,  $m0 \leq m \leq 16$ , where *m0* is the highest harmonic among all the sources.

### Example:

```
Model
...
CIRCUIT ... HARM=5;
...
End
```

This becomes the “global” default which applies to all sweep sets of the HB type.

You can also specify the highest harmonic to be included in harmonic balance simulation for individual sweep sets of the HB type using the keyword HARM.

### Syntax:

```
HB: HARM=m1 ...;
```

*m1* specifies the highest harmonic for the sweep set,  $m0 \leq m1 \leq m$ , where *m0* is the highest harmonic among all the sources and *m* is the global default.

### Example:

```
Sweep
HB: HARM=4 ...;
...
End
```

Sweep sets for which the HARM keyword is not explicitly specified will assume the global default.

## The RREF Keyword

For small-signal AC sweep sets (i.e., the simulation type is AC), the  $S$  parameters are, by default, calculated with respect to the actual port terminations (impedances) as specified in the PORT definitions in the Model block of the input file (see Chapter 6). You can override this default by specifying the reference impedance using the keyword RREF.

### Syntax:

```
AC: RREF= $x$  ...;
```

where  $x$  specifies the  $S$ -parameter reference impedance for the sweep set, and  $x$  must be a positive constant value or constant label.

### Example:

```
Sweep  
AC: RREF=60 ...;  
...  
End
```

This will work much the same way as though the impedance of 60 ohm (purely resistive) were specified for each port as its terminating impedance. The difference is that this definition takes effect only for the sweep set in which it is specified.



## 9.4 Parameter Sweeps

Parameter sweeps are defined by means of sweep labels and sweep ranges.

### Syntax:

```
sweep_label: x1, x2, ... xn
```

where *sweep\_label* is an existing label and *x1*, *x2*, ..., *xn* are numerical values.

### Example:

```
Width: 1.1, 1.2, 1.7, 3.5, 5
```

where the label *Width* must have already been defined.



The sweep label must be followed by a colon ":". You cannot use an equal sign "=" instead of the colon. Colon is chosen to designate sweep labels, and equal sign is reserved for assignment of a single value. For example,

```
Width = 1.1, 1.2, 1.7, 3.5, 5
```

would be mistaken as an assignment of a single value, namely 1.1, to *Width*.

### Sweep an Interval with Uniform Step Size

### Syntax:

```
sweep_label: from x1 to x2 step=x3
```

where *sweep\_label* is an existing label and *x1*, *x2* and *x3* are numerical values.

This translates into a set of values as: *x1*, *x1+x3*, *x1+2×x3*, ..., *x2*.

### Example:

```
Width: from 1.5 to 5 step=0.5
```

This translates into a set of values for *Width* as 1.5, 2, 2.5, ..., 5.



The upper limit of the interval, namely *x2*, is always included in the translated values, even if the step size does not divide the interval evenly. For example,

```
Width: from 1.5 to 2.75 step=0.5
```

The translated values are 1.5, 2, 2.5, 2.75, which include the upper limit 2.75.

You can specify a negative step size, such as

Width: from 5 to 1.5 step=-0.5

The translated values are 5, 4.5, 4, ..., 1.5. In this case, the lower limit must be greater than the upper limit, i.e.,  $x1 > x2$ .

The number of subintervals is  $n = (x2 - x1) / x3$ , if  $x3$  divides  $(x2 - x1)$  evenly, or else  $n = (x2 - x1) / x3 + 1$ , (the number of values is  $n + 1$ ). The limit is  $n \leq 1023$ .

## Specify the Number of Steps

### Syntax:

*sweep\_label*: from  $x1$  to  $x2$  N= $n$

where *sweep\_label* is an existing label,  $x1$  and  $x2$  are numerical values,  $n$  is an integer,  $1 \leq n \leq 1023$ . The step size is  $(x2 - x1) / n$ . The number of values is  $n + 1$ .

Example:

Width: from 1 to 5 N=8

This translates into a step size of  $(5 - 1) / 8 = 0.5$ , and a set of values as 1, 1.5, 2, ..., 5. Note that the total number of values is 9.

## Sweep an Interval with Exponential Step Size

### Syntax:

*sweep\_label*: from  $x1$  to  $x2$  ratio= $x3$

where *sweep\_label* is an existing label and  $x1$ ,  $x2$  and  $x3$  are numerical values. This translates into a set of values as:  $x1$ ,  $x1 \cdot x3$ ,  $x1 \cdot x3^2$ , ...,  $x2$ .

Example:

Width: from 3 to 96 ratio=2

This translates into a set of values for Width as 3, 6, 12, 24, 48, 96.



The upper limit of the interval, namely  $x_2$ , is always included in the translated values, even if it does not coincide with the prescribed ratio. For example,

Width: from 3 to 30 ratio=2

The translated values are 3, 6, 12, 24, 30.

The values  $x_1$ ,  $x_2$  and  $x_3$  must be consistent, i.e.,

$x_3 > 1$  if  $x_2 > x_1$

$x_3 < 1$  if  $x_2 < x_1$

## Specify the Number of Exponential Steps

Syntax:

```
sweep_label: from x1 to x2 NEXP=n
```

where *sweep\_label* is an existing label,  $x_1$  and  $x_2$  are numerical values,  $n$  is an integer,  $1 \leq n \leq 1023$ . The step ratio is  $(x_2 / x_1)^{1/n}$ . The number of values is  $n + 1$ .

Example:

Width: from 3 to 96 NEXP=5

This is equivalent to specifying a step ratio of  $(96 / 3)^{1/5} = 2$ . The values for sweeping Width are 3, 6, 12, 24, 48, 96. Note that the total number of values is 6.

## Combine Sweep Intervals and Points

For a sweep label, you can specify an arbitrary combination of uniformly spaced sweep intervals, exponentially spaced sweep intervals and discrete points. For example:

```
Width: 0.75 0.8 from 1 to 5 step=1
        from 7 to 15 step=2 19.99
```

which translates into the following set of values:

```
0.75 0.8 1 2 3 4 5 7 9 11 13 15 19.99
```



The set of values assigned to a sweep label will not be sorted by the file parser. During simulation, the values are assigned in the exact order as they appear in the input file. If the values need to be processed in a particular order (ascending, descending, etc.), they must be entered in that order.

## What Can Be Sweep Labels

You can use the predefined label `FREQ` as a sweep label:

```
FREQ: from 4GHZ to 20GHZ step=2GHZ
```

You can also sweep any labels which are defined in the `Expression` and `Model` blocks to represent numerical constants or optimization variables.

For example, suppose the following labels are defined in the `Expression` (or `Model`) block:

```
Width: 3;
Length: ?10cm?;
Area = Width * Length;
```

You can use `Width` and `Length` as sweep labels, because they represent a constant and an optimization variable, respectively. But `Area` cannot be a sweep label because it represents a formula.

Once a parameter sweep is finished during simulation, the value of the sweep label is reset to its original value, i.e., the constant or nominal value as defined in the `Expression` or `Model` block.



You cannot "create" sweep labels that have not been defined in the `Expression` or `Model` block.

## Macro Constants and Expressions

Macro constants, labels and expressions can be used in the definition of sweep ranges (values). For example,

```
#define MIN_X 1.0
#define X_RANGE 8.0

X_Step = 2;
...

X: from MIN_X to (MIN_X + X_RANGE) step=X_Step
```

where `MIN_X` and `X_RANGE` are macro constants, `X_Step` is a label, and the upper limit of the sweep interval is defined by an expression.

Only constant labels can be used. Expressions, if used, must be enclosed in a pair of parentheses to avoid ambiguity.

## Two-Dimensional Sweep

OSA90 permits up to two independent sweep labels per sweep set. For a two-dimensional sweep set, the first sweep label is called the **outer sweep label**, and the second one is called the **inner sweep label**.

Example:

```
Width: from 2 to 8 step=2
Length: 3, 4, 5
```

Width is the outer sweep label and Length is the inner sweep label.

The two-dimensional sweep defined in the above example translates into pairs of values for the pair of labels (Width, Length) in the following order:

```
(2, 3), (2, 4), (2, 5)
(4, 3), (4, 4), (4, 5)
(6, 3), (6, 4), (6, 5)
(8, 3), (8, 4), (8, 5)
```

It is equivalent to a double loop in the "C" programming language as

```
for (Width = 2; Width <= 8; Width += 2) {
    for (Length = 3; Length <= 5; Length += 1) {
        ...
    }
}
```

## The Order of Sweeps in Circuit Simulation

The order of the two sweep labels can be significant in circuit simulation. For instance, if the following two-dimensional sweep is requested of small-signal AC simulation

```
Bias_Voltage: from 1V to 3V step=0.5V
FREQ: from 6GHZ to 18GHZ step=2GHZ
```

it is desirable to have the bias voltage as the outer sweep label and the frequency as the inner sweep label (as shown). Because, the nonlinear devices, if used, need to be linearized for small-signal analysis by solving the DC nonlinear equations. A new solution of the DC nonlinear equations is needed only when the bias voltage changes, but not when the frequency changes. Therefore, having the bias voltage as the outer sweep label minimizes the number of nonlinear DC solutions required for the sweep set.

If a large-signal harmonic balance simulation involves a power sweep, it is preferable to have the power sweep as the inner sweep, because once the nonlinear harmonic balance equations are solved at one point, it is relatively easy to obtain a new solution after an incremental change in the power level. It is usually more time-consuming to obtain a new solution after changing other parameters.

## Sweeping Array Indices

One very useful application of parameter sweeps is to sweep indices of arrays. For example,

```

Model
  A[10] = [a1 a2 ... a10];
  B[10] = [b1 b2 ... b10];


  K: 1;

  C = A[K] + B[K] ...;
  ...
end

Sweep
  K: from 1 to 10 step=1 ...;
  ...
end

```

where the sweep label *K* is used as an index of reference to the arrays *A* and *B* in the expression by which *C* is defined.

 See Chapter 4 for further details on arrays.

An example of sweeping the indices of matrices:

```

Model
  MS_Data[3,3] = [ ... ];

  ....

  CIRCUIT ...;

  I: 1;
  J: 1;

  MS_Error = MS[I,J] - MS_Data[I,J];
  ...
end

Sweep
  AC: I: from 1 to 3 step=1
      J: from 1 to 3 step=1 ...;
  ...
end

```

where the matrix *MS* contains the calculated small-signal *S* parameter magnitudes (built-in response) and *MS\_Data* may represent measured data.

## 9.5 Parameter Labels

In addition to sweep labels, you can also define parameter labels for a sweep set. Parameter labels are not to be swept, rather, they are assigned a single value or to be evaluated from an expression.

**Syntax:**

```
parameter_label = x
```

where *parameter\_label* is a label and *x* can be a value, a label or an expression.

**Examples:**

```
Drain_Bias = 4V
Omega = (2 * PI * FREQ)
```

where Drain\_Bias is a parameter label which is assigned a constant value, and Omega is a parameter label specified by an expression.

Up to 64 parameter labels can be defined in each sweep set.

Similar to sweep labels, parameter labels must be defined in the Model or Expression block and represent numerical constants or optimization variables.

### Parameter Labels as Functions of Sweep Labels

Parameter labels can be functions of the sweep labels defined in the same sweep set.

**Example:**

```
FREQ: from 1GHZ to 10GHZ step=1GHZ
Omega = (2 * PI * FREQ)
```

The formula (expression) must be enclosed within parentheses to avoid ambiguity.

Using this concept, you can create synchronous sweeps, such as

```
K: from 0 to 20 step=1
XK = (X0 + K * DX)
YK = (Y0 + K * DY)
ZK = (Z0 + K * DZ)
```

where a number of parameter labels are tied together through a sweep label. In other words, this creates a one-dimensional sweep of several parameters.

## 9.6 Output Labels

Output labels are required components of a sweep set. They represent the responses and functions to be calculated during simulation and subsequently displayed.

Output labels may include any of the labels defined in the Model and Expression blocks: constants, variables, circuit responses, postprocessed functions, outputs from external programs via Datapipe, etc.

Example:

```
Sweep
  DC: ...  Voutput_DC, Ioutput_DC;


  AC: ...  MS11 PS11 MS21;

  HB: ...  MVoutput, Output_Power_dBm;

  ... AA, BB, CC;
end
```

where the output labels of the first sweep set include Voutput\_DC and Ioutput\_DC (DC circuit responses), the output labels of the second sweep set include MS11, PS11 and MS21 (small-signal  $S$  parameters), and the output labels of the third sweep set include MVoutput (large-signal voltage spectrum) and Output\_Power\_dBm (a user-defined function from response postprocessing). The last sweep set does not have an explicitly specified simulation type, which indicates that it involves abstract simulation only. Therefore, the output labels of the last sweep set, namely AA, BB and CC, cannot include any built-in circuit responses or functions of the built-in circuit responses. They must be labels defined by abstract mathematical expressions and/or outputs from external simulators via Datapipe.



If any circuit responses are included in a sweep set, they must be consistent with the simulation type (DC, AC, HB).  See Chapter 6 for the built-in response labels corresponding to each simulation type.

### Arrays as Output Labels

Output labels can be arrays (vectors and matrices), in which case all the array elements will be included. In the preceding example, the output label MVoutput of the HB sweep set is an array whose elements are voltage spectrum components at different harmonics. Having MVoutput as an output array is equivalent to specifying all its elements, namely MVoutput[0], MVoutput[1], ..., MVoutput[ $m$ ], as output labels, where  $m$  is the highest harmonic index.

**The maximum number of output labels** is limited to 512 per sweep set. An output array is equivalent to as many output labels as the number of its elements. If more is needed, you can divide and spread the output labels to a number of sweep sets of the same simulation type.



## 9.7 OSA90 Display Menu

The “Display” menu option is designed to manage the simulation and display features

☞ see Chapter 2 for a general discussion on the OSA90 user interface.

The steps needed for simulation and display are outlined as follows.

- ▶ Prepare an appropriate input file.
- ▶ Select a display format from the “Display” menu as listed in Table 9.2.

TABLE 9.2 DISPLAY FORMATS

Menu Option	Brief Description
Xsweep	displays responses versus parameter sweeps
Parametric	displays parametric plots of responses
Array	displays elements of arrays
Waveform	displays time-domain waveforms
SmithChart	displays Smith charts and polar plots
Visual	displays 3D visualization and contours
Report	generates a report file

To invoke any of the “Display” menu options requires the Sweep block of the input file, since the simulation and displays are organized according to the Sweep block.


### Simulation

When you choose a display option, OSA90 first performs the necessary simulation to calculate the output labels according to the Sweep block.

An intelligent “host acceleration file” feature is built into OSA90 to avoid duplicate and unnecessary simulation. If simulation has been performed for an input file, it will not be repeated in a subsequent session unless you have made changes to the file. Instead, the results from the previous simulation will be automatically retrieved for display. This feature is discussed in greater detail in Section 9.15.

## Interrupt Simulation

During simulation, OSA90 displays a progress bar on top of the status bar, located at the bottom of the OSA90 window.

While the simulation is in progress, you can interrupt it at any time by clicking on the “Stop” button (  ) located on the toolbar. This will terminate the simulation without prompting for confirmation.

## Sequence of Calculations

The simulator organizes the calculations in the following sequence:

- 1 Start from the first sweep set in the Sweep block. After the first sweep set is done, go on to the second sweep set, and so on. The simulation is complete when all the sweep sets are processed in the natural order.
- 2 The outer sweep loop. Set the outer sweep label to its next available value. After all the values have been assigned, go to step 1 (i.e., go to the next sweep set).
- 3 The inner sweep loop. Set the inner sweep label to its next available value. After all the values have been assigned, go to step 2.
- 4 Set the parameter labels. Each parameter label is either specified with a single value or evaluated from an expression (which may depend on the sweep labels).
- 5 Perform circuit analysis if required. Calculate the output labels. Go to step 3.



Output labels are calculated on an "as necessary" basis. Once an output label is calculated, it will not be recalculated until it is affected (directly or indirectly) by a change in the sweep labels.

For instance, if an output label is a function of the outer sweep label but independent of the inner sweep label, then it is recalculated for each and every step of the outer sweep loop, but it will not be recalculated when the value of the inner sweep label changes.

## 9.8 Xsweep Display

The “Xsweep” option on the “Display” menu provides the most common format of presentation: a rectangular plot of one or more output labels versus a sweep label. This is illustrated in Fig. 9.1, where an output label MS21 is plotted versus the sweep label FREQ.

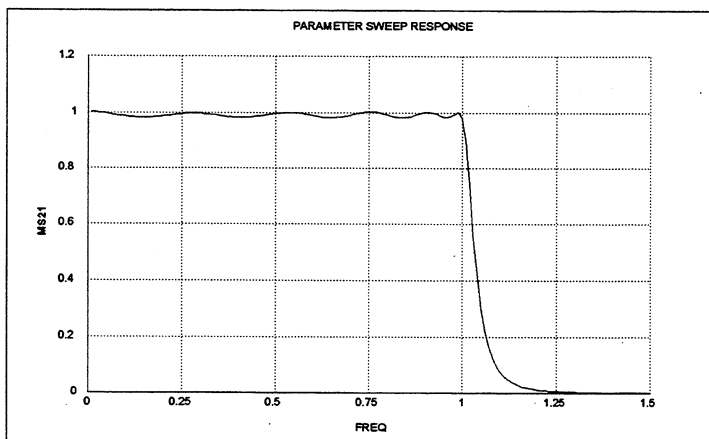
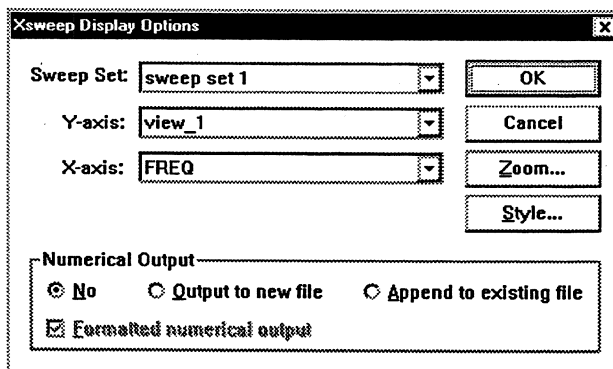



Fig. 9.1 Xsweep display.

### Xsweep Display Options Dialog Box

When you select “Xsweep” from the “Display” menu, a dialog box will appear.



## Select a Sweep Set

If you have defined more than one sweep set in the Sweep block, the sweep sets can be selected for display one at a time. Click on the  button, on the same line as “Sweep Set” and select the desired sweep set.

## Define the Y-axis

Xsweep is a rectangular X-Y plot, and you need to define the X-axis (horizontal) and the Y-axis (vertical).

The Y-axis represents a function or functions of the sweep label. If a sweep set contains only one output label, then it is the only choice for the Y-axis.

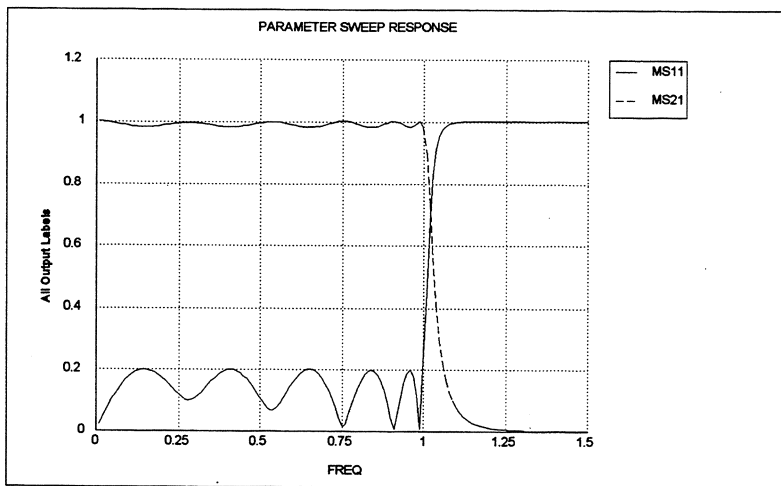
For sweep sets which contain a number of output labels, you can choose to display individual labels by using the “Y-axis” option.

You can also choose to display all the output labels together in one diagram by selecting “All Output Labels” for the “Y-axis” option.

For example, consider the sweep set defined as

```
AC:  FREQ:  from 0.01GHZ to 1.5GHZ step=0.01GHZ
      MS11 MS21;
```

which contains two output labels, namely MS11 and MS21. You can select MS11 or MS21 to be displayed individually, or you can choose to display both MS11 and MS21 in the same diagram, as illustrated in Fig. 9.2.



*Fig. 9.2 Xsweep display of all output labels.*

If there are arrays among the output labels, you can select an array for the Y-axis, then all the elements of that array will appear in one display.

## Define the X-axis

If a sweep set contains one sweep label, then it is the only choice for the X-axis.

For sweep sets which contain two sweep labels, either one can be chosen as X-axis. You can select either label as the "X-axis".

For example, consider the sweep set

```
DC:  VG: from -2 to 0 step=0.5
      VD: from 0 to 15 step=1
      ID_MA;
```

which has two sweep labels and either can be chosen for the X-axis.

Suppose that VD is chosen for the X-axis, then you can select for VG a value from those available, namely -2, -1.5, -1, -0.5 or 0.

Or, you can choose to include all the values of VG in the same display. In this case, a family of curves will be displayed. Each of the curves represents the output label ID\_MA versus the sweep label VD when VG is held at a given value, as illustrated in Fig. 9.3.

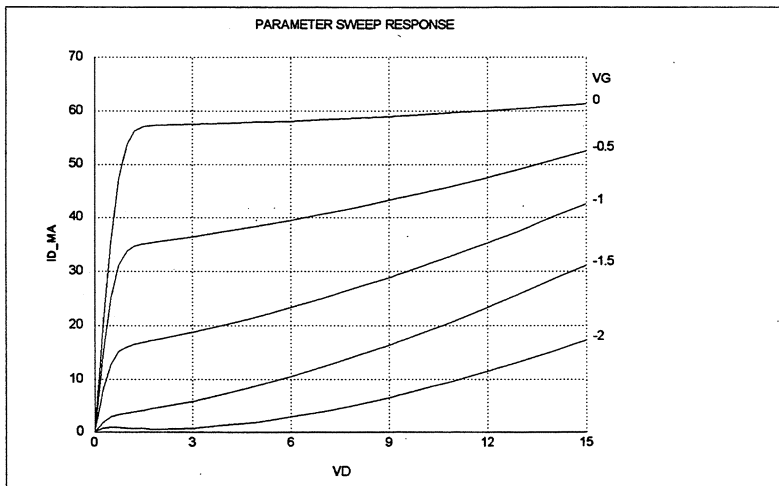


Fig. 9.3 Xsweep display including all the values of both sweep labels.

## Display All Sweep and Output Labels

You can also choose to include all the output labels and all the sweep labels in one display.

Example:

```
DC:  VG: from -2 to 0 step=0.5
      VD: from 0 to 15 step=1
      ID_MA ID_DATA;
```

You can choose to display both ID\_MA and ID\_DATA versus VD (or VG) *and* include all the values of VG (or VD), as illustrated in Fig. 9.4.

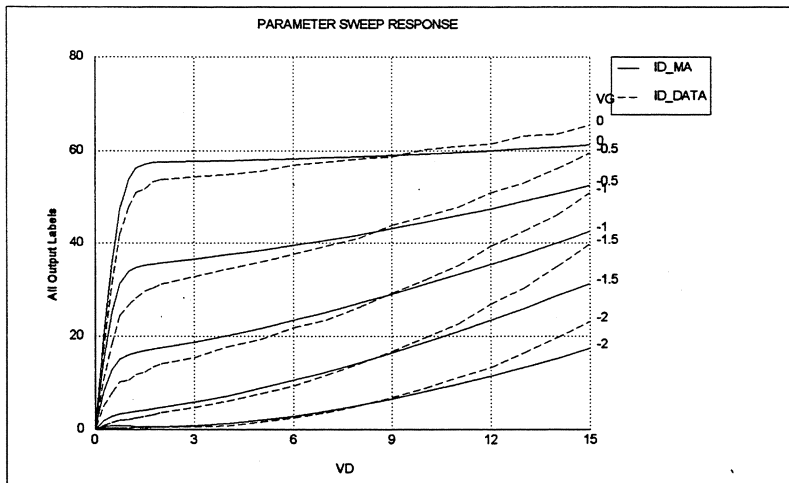
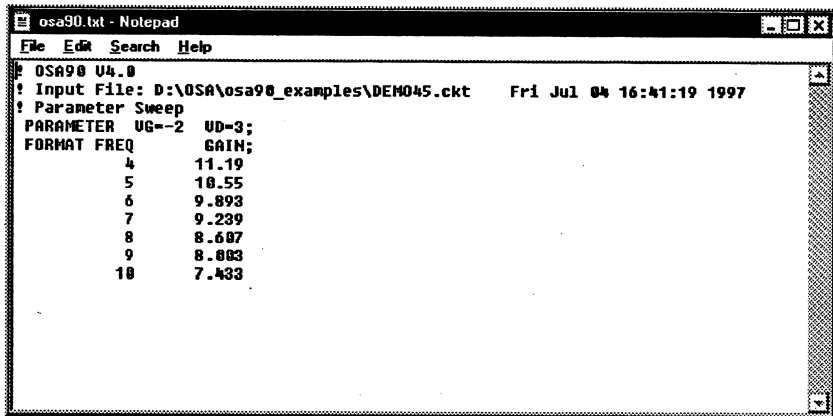


Fig. 9.4 Xsweep display including all the sweep and output labels.

## 9.9 Numerical Display

In addition to graphical display, “Xsweep” can also display the output labels numerically.

The values of the sweep label and output labels are printed in ASCII text format. The program automatically launches “Notepad”, which allows you to view, edit and save the numerical outputs.



```

osa90.txt - Notepad
File Edit Search Help
! OSA90 U4.0
! Input File: D:\OSA\osa90_examples\DEMO45.ckt   Fri Jul 04 16:41:19 1997
! Parameter Sweep
PARAMETER UD=-2 UD=3;
FORNAT  FREQ      GAIN;
         4        11.19
         5        10.55
         6         9.893
         7         9.239
         8         8.607
         9         8.003
        10         7.433
  
```

*Fig. 9.5 Numerical Display.*

### Numerical and Numerical Append

The “Numerical Output” option in the dialog box offers three different forms of numerical display, namely “No”, “Output to new file” and “Append to existing file”.

The default option “No” does not create any numerical output.

If you choose “Output to new file”, the current numerical outputs will overwrite any previous ones. In other words, the file holds the current numerical outputs only.

The choice “Append to existing file”, on the other hand, will append the current numerical outputs to any previous ones. In other words, all numerical outputs are accumulated in one file with the current outputs at the end.



The numerical output file is created in the current directory under the name “osa90.txt”. You can make a copy of the file under a different name so it will not be overwritten by the next numerical output.



## Define the X-axis and Y-axis

Consistent with the graphical Xsweep display, the numerical display also allows you to choose labels for the X-axis and Y-axis.

You can choose any one or all of the output labels for the Y-axis. If the sweep set does not contain any sweep label, then no X-axis needs to be defined. If the sweep set contains one sweep label, then it is the only choice for the X-axis. If the sweep set contains two sweep labels, either of them can be chosen for the X-axis.

The numerical display is in a multi-column format: the first column represents the X-axis (if any), and each of the other columns corresponds to one of the output labels (Y-axis).

## Formatted and Unformatted Displays

When you select numerical display, the “Formatted numerical output” option will become active..

Formatted display (checkmark present in check box) automatically adjusts the spacing between the columns of data so that the data will be aligned with the corresponding label names on the header line. Unformatted display (checkmark absent), on the other hand, does not make such an adjustment.

Normally, the output labels are all displayed on one line for each value of the sweep label. But, if there are many output labels, then the formatted display will divide the output labels into subsets and organize the display into sections; each section shows a subset of the output labels such that the lines of data are of reasonable length. This does not apply to unformatted display.

## 9.10 Graphics Zoom

By default, graphics displays are scaled to the range of the output values. You can change the display scale ("zooming") by clicking on the "Zoom" button in the "Xsweep Display Options" dialog box. The "Zoom" button leads you to another dialog box.

The "Display Zoom" dialog box contains the following controls:

- X-min: [ ]
- Y-min: [ ]
- X-max: [ ]
- Y-max: [ ]
- N X-ticks: [ 0 ]
- N Y-ticks: [ 0 ]
- Log scale for X
- Log scale for Y
- Auto scale
- OK
- Cancel

The various parameters of the dialog box can be illustrated using Fig. 9.6.

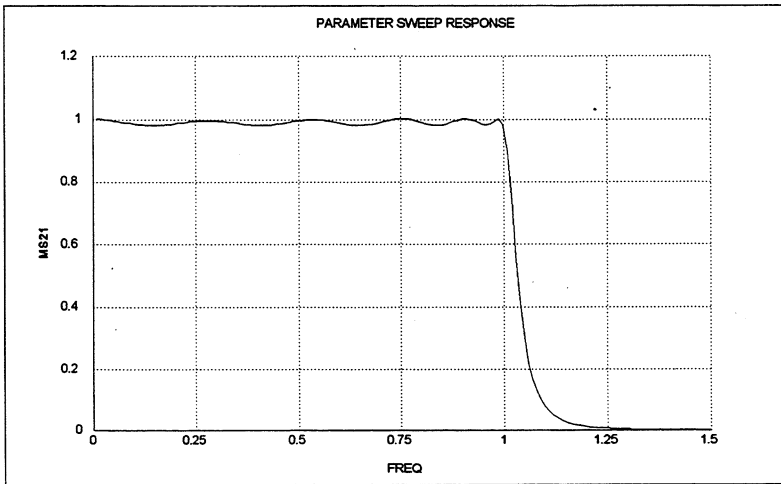


Fig. 9.6 An example illustrating the zoom scale.

The parameters "Y-min", "Y-max", "X-min" and "X-max" define the corners of the display. The parameters "N X-ticks" and "N Y-ticks" specify the number of ticks (grids) on the X-axis and Y-axis, respectively.



Before you can change any values, you must make sure that the "Auto scale" check box does not contain a check mark.

For example, in Fig. 9.6, we have Y-min = 0, Y-max = 1.2, X-min = 0 and X-max = 1.5. The number of ticks on the X-axis is 6 (notice that the origin (X-min, Y-min) does not count as a tick). The number of ticks on the Y-axis is also 6.

Each of the parameter values can be changed independently. For example, if after viewing the display shown in Fig. 9.6, you change the zoom parameters to Y-min = 0.97, Y-max = 1, X-max = 1, N X-ticks = 5 and N Y-ticks = 3 (X-min = 0 remains unchanged), then the zoomed display will be as illustrated in Fig. 9.7.

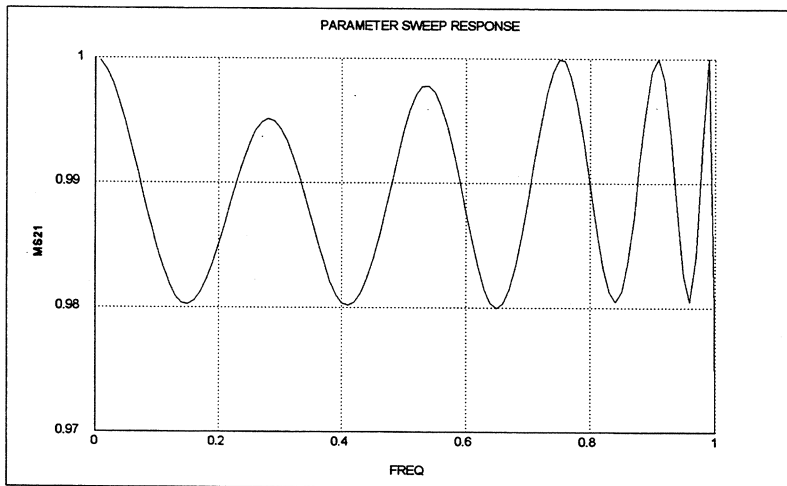


Fig. 9.7 Zoomed display.



You can turn the display upside down by specifying Y-min > Y-max. You can also produce a mirror image of the display by specifying X-min > X-max.

## Log-Scale Display Using the Zoom Option

By default, OSA90 displays graphics in linear scales. You can choose to have the graphics displayed in logarithmic scale using the "Zoom" button. You can also define log-scale display using Graphical Views (Chapter 10).

In the "Display Zoom" dialog box, you can check the "Log scale for X" option to specify logarithmic scale for the X-axis of the graphics display. Similarly, you can check the "Log scale for Y" option to specify logarithmic scale for the Y-axis of the graphics display.

In Fig. 9.8, both the X- and Y-axes are displayed in logarithmic scale.

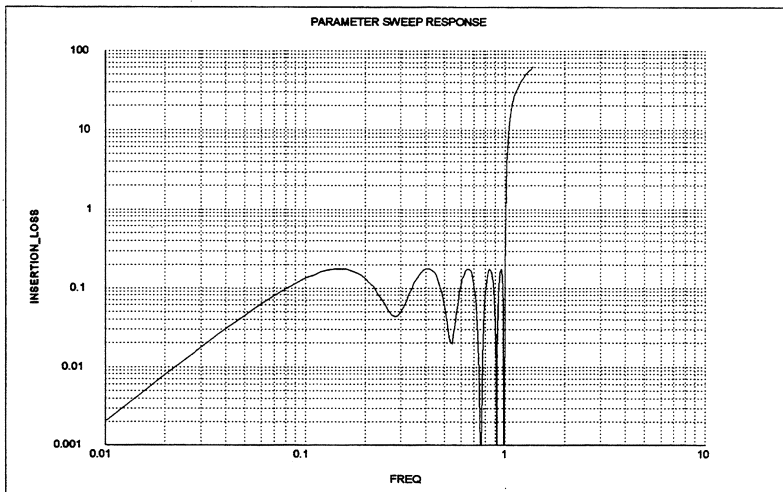


Fig. 9.8 Log-scale display of the insertion loss of a low-pass filter.



If log-scale is specified, the limits for the corresponding axis must be positive. For instance, if you specify log-scale for the X-axis, you must ensure that both "X-min" and "X-max" have positive values. An error message will be displayed if either limit is found to be zero or negative.

## 9.11 Draw Types

Normally, data is displayed graphically as curves. This can be changed by clicking on the “Style” button in the dialog box. This button presents you with another dialog box titled “Display Styles”.

**Curve:** all the points of an output label are connected to form a continuous curve.

**Bars:** data is plotted as vertical bars to form a bar chart.

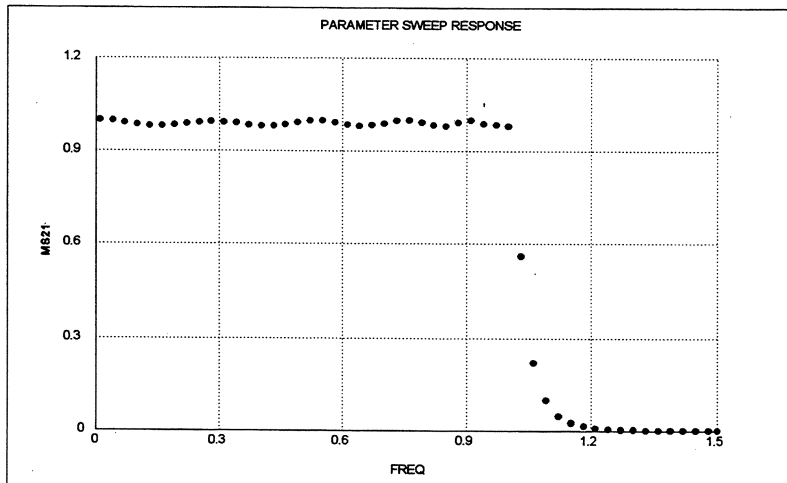
**Points:** data is plotted as discrete, unconnected points.

**Dots:** similar to Points except that the points are thicker.

**Bins:** similar to Bars except that bins are not filled.

**Circles:** Similar to Dots except that circles are used.

**Example:**



*Fig. 9.9 Display after the draw type is set to "Dots".*



The draw style specified in the dialog box applies to all the output labels, i.e., they are all displayed as curves, bars, points or dots. The feature of user-defined views (Chapter 10) allows you greater control in terms of setting different draw types for individual output labels.

## 9.12 Parametric Display

The parametric display featured by the option “Parametric” under the “Display” menu plots one output label versus another output label.

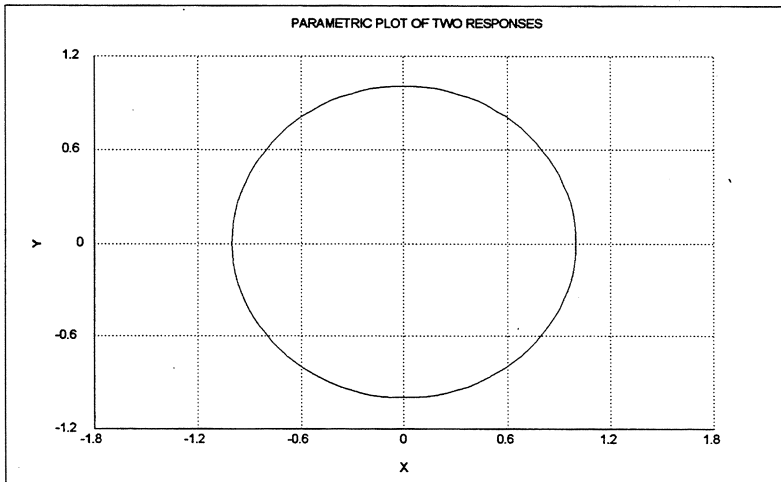
For the parametric display to be applicable, a sweep set must contain one or two sweep labels. One sweep label is chosen as the “Parametric” label. But, the sweep label does not actually appear in the graphics. Rather, both the X-axis and the Y-axis represent output labels which are functions of the parametric label.

Example:

```
Expression
  Angle: 0;
  X = cos(Angle);
  Y = sin(Angle);
end

Sweep
  Angle: from 0 to (2 * PI) N=100 X, Y;
end
```

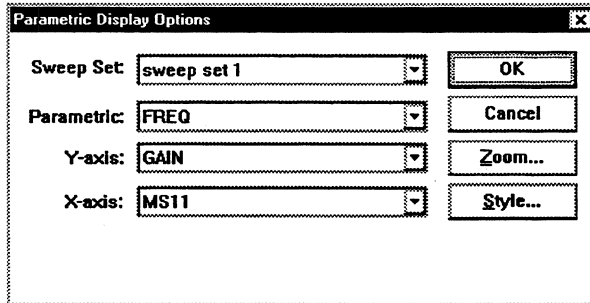
As the label Angle sweeps from 0 to  $2\pi$ , the output labels X and Y become Cartesian coordinates of the points on a full circle. You can see the circle by plotting Y versus X with Angle as the parametric label, as illustrated in Fig. 9.10.




*Fig. 9.10 Displaying a circle using parametric plot.*

## Parametric Display Options Dialog Box

Once you select “Parametric” from the “Display” menu, a dialog box will appear.



### Select a Sweep Set

If you have defined more than one sweep set in the Sweep block, the sweep sets can be selected for display one at a time. Click on the  button on the same line as “Sweep Set” and select the desired sweep set.

### Select the Parametric Label

The parametric display requires that the sweep set contains at least one sweep label. If the sweep set contains one sweep label, then it is automatically selected as parametric label.

If there are two sweep labels, then either one can be chosen as the parametric label. The other sweep label will be appear in the dialog box so you can select one or all of the values available for that label. If you select all the values for the sweep label which is not chosen as the parametric, a family of curves is displayed. Each curve is generated by sweeping the parametric label while holding the other sweep label at a given value.

### Define the X-axis and Y-axis

Since the parametric display plots one output label versus another output label, the sweep set must contain two or more output labels. Any one of the output labels can be selected for the X-axis.

For the Y-axis, you can select individual output labels or all the output labels to be plotted together in one display. If there are arrays among the outputs, you can also select an array for the Y-axis, then all the elements of that array will be plotted in one display.

## Log-Scale Plots

To further illustrate parametric display, consider using it to generate a log-scale plot by defining the X-axis as the logarithm of a sweep label:

```

Model
...
  LOG10_FREQ = log10(FREQ);
...
end

Sweep
  AC: FREQ: from 0.01 to 10 step=0.01 LOG10_FREQ MS11 MS21;
end

```

Fig. 9.11 illustrates the logarithmic plot when LOG10\_FREQ is specified as the parametric label and the Y-axis is chosen as "All Output Labels" (the parametric label is automatically excluded, leaving the other two output labels, namely MS11 and MS21).

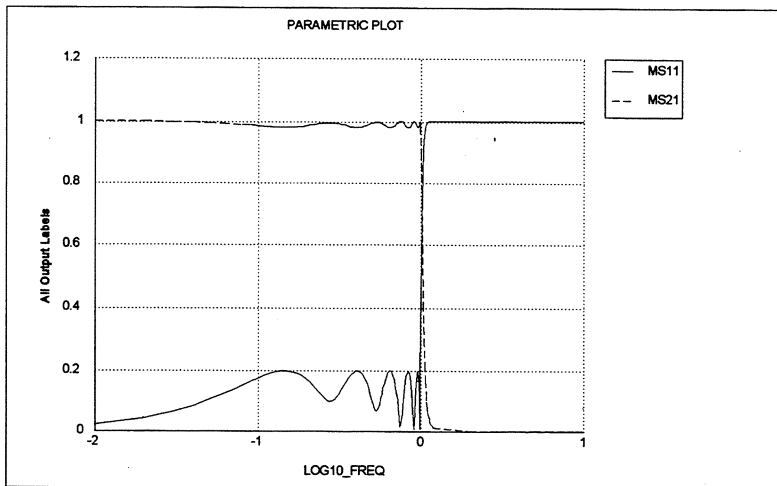


Fig. 9.11 Logarithmic plot using parametric display.

☞ Log-scale plots can be generated directly using Zoom, see Section 9.10.



## 9.13 Array Display

The “Array” option under the “Display” menu plots the elements of an output array. For this option to be available, the sweep set must include array or arrays as outputs.

The X-axis of the display is the index of array elements. For example, if A[10] is the array to be displayed, then the X-axis shows the index from 1 to 10.

The Y-axis is the values of the array elements. By default, the draw type is Bars (i.e., the display is a bar chart). The draw type can be changed to Curves, Points, Dots, Bins or Circles.

Fig. 9.12 illustrates an example of array display, where the array POUTPUT is the output power spectrum of a nonlinear circuit. The array index shown as the X-axis is actually the harmonic index.

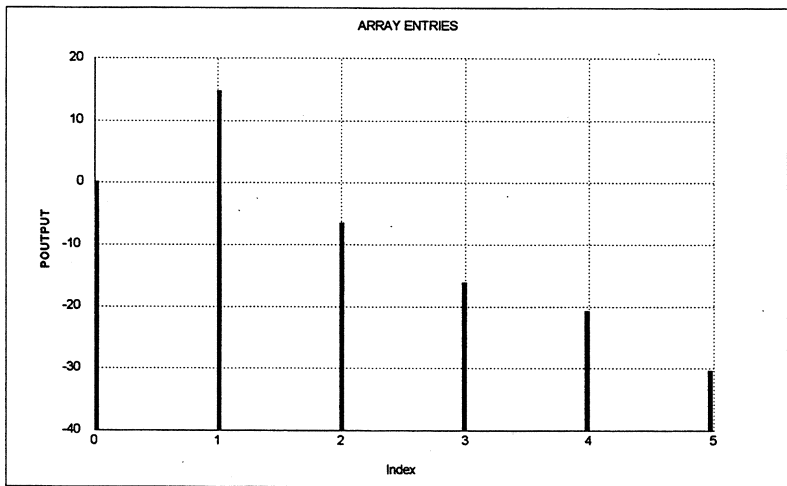


Fig. 9.12 Array display.

## 9.14 Visualization and Contour Plotting

The “Visual” option under the “Display” menu facilitates 3D visualization and contour plotting. Fig. 9.13 is an illustration of the 3D visualization screen. The actual display in full colors is of course much more attractive.

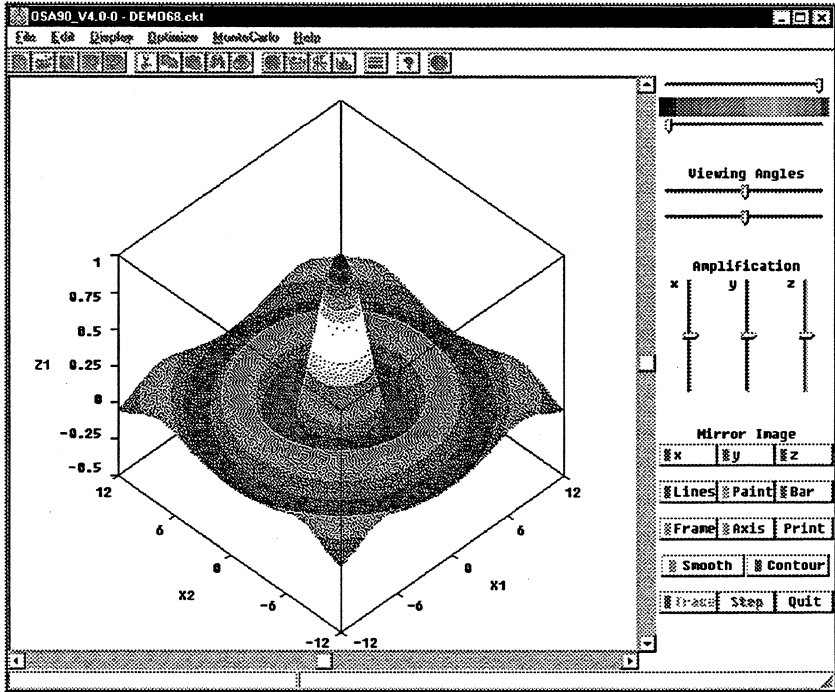


Fig. 9.13 Example of visualization.

The function depicted in Fig. 9.13 is defined in the demo file demo68.ckt as

```
t = sqrt(X1 * X1 + X2 * X2);
Z1 = if (t < 0.01) (1) else (sin(t)/t);
```

## Definition of Output Labels and Sweep Parameters

The functions and variables for 3D visualization must be defined in the Sweep block as output labels and sweep parameters, respectively.

For example, the sweep set corresponding to Fig. 9.13 is defined as

```
Sweep
  X1: from -12 to 12 n=50   X2: from -12 to 12 n=50   Z1;
end
```

If more than one output label is defined, such as

```
X1: from -12 to 12 n=50
X2: from -12 to 12 n=50   Z1, Z2, Z3;
```

only one output label can be plotted at a time.

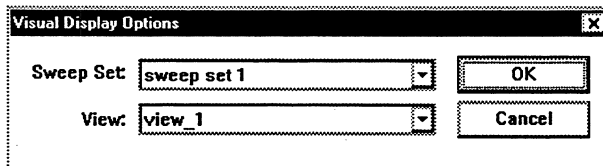
The values of the sweep parameters do not have to be uniformly spaced. For example,

```
X1: 0.5 0.6 1.0 2.0 3.0 4.0 5.0 5.3 5.5
X2: from -7 to 0 step=1   from 0.5 to 7 step=0.5   ...;
```

This is perfectly acceptable.

## Select Sweep Set and Function in the Pop-Up Window

Once you select "Visual" from the "Display" menu, a dialog box will appear.



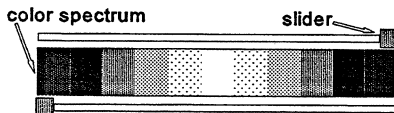
Sweep sets which contain two sweep parameters are suitable for 3D visualization. If the Sweep block contains more than one suitable sweep set, then you can select from the different sweep sets using the "Sweep set" option.

The "View" option allows you to select the output label for visualization, if several output labels are available in the sweep set.

## Coloring Scheme

Although drawings are illustrated in this Manual as wire-frames, the actual drawings appear on the screen in color.

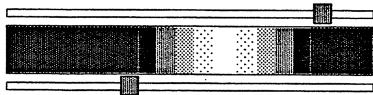
Twenty different colors are used for visualization. The coloring scheme is represented by a color spectrum and controlled by a pair of sliders.



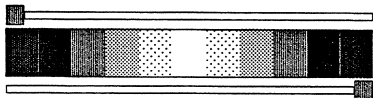
By default, the left-most color in the spectrum (blue) represents the lowest function value and the right-most color (red) represents the highest function value.

You can change the coloring scheme using the sliders. The upper and lower sliders control the right-most and left-most colors, respectively. The total length of the slider represents the total spread of the function values, and the slider thumb positions mark proportionally the division of the function values into three sections. The section below the lower thumb is painted in one color, namely the left-most color (blue). The section above the upper thumb is also painted in one color, namely the right-most color (red). The section between the two thumbs is further divided into 20 subsections, each represented by a different color in the color spectrum.

By moving the sliders, you can zoom-in on a portion of the spread of function values which will be colored in greater detail. Example:



You can reverse the positions of the slider thumbs, as shown in the following illustration. This makes blue represent high function values and red represent low function values.



## Viewing Angles

The drawing produced by visualization is a projection of 3D images onto the 2D computer screen. You can control the viewing angles using two sliders located directly under the color sliders.

The upper slider controls the angle of rotation, and the lower slider controls the angle of elevation. The default value for both angles is  $45^\circ$ .

Using the sliders, you can change the angles between  $0^\circ$  and  $90^\circ$ , in increments of  $1^\circ$ . The current value of the angle is shown in the thumb of the slider.

When rotation is  $0^\circ$ , the X-axis is parallel to the screen and the YZ-plane is perpendicular to the screen. When rotation is  $90^\circ$ , the Y-axis is parallel to the screen and the XZ-plane is perpendicular to the screen.

When elevation is  $0^\circ$ , the Z-axis is parallel to the screen. When elevation is  $90^\circ$ , the Z-axis is perpendicular to the screen.

Examples:

rotation = $0^\circ$ and elevation = $0^\circ$	--	frontal view
rotation = $90^\circ$ and elevation = $0^\circ$	--	side view
elevation = $90^\circ$	--	top view

## Mirror Images

The sliders give you control over the viewing angles between  $0^\circ$  and  $90^\circ$ . The row of buttons under the label "Mirror Image" allow you to reverse the reference directions of the X-, Y- and Z-axes.

x When this button is off, the X-axis reference direction is from left to right. When it is on, the direction is from right to left.

y When this button is off, the Y-axis reference direction is from front to back. When it is on, the direction is from back to front.

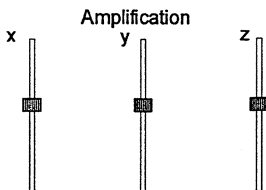
z When this button is off, the Z-axis reference direction is from bottom to top. When it is on, the direction is from top to bottom.

By default, all three buttons are "off".

## Ratios of Amplification

By default, the X, Y and Z values (i.e., values of the sweep parameters and the function) are scaled by the same factor to fit into the drawing window on the screen.

You can control the amplification ratios using the vertical sliders.



The default ratios are indicated by the position of the thumbs, which is the middle of the sliders. If you move the thumb of a slider down, it reduces the amplification ratio. If you move the thumb up, it increases the ratio.

Taking the default ratio as a reference value of 1.0, the total length of the sliders represents a range from 0.0 to 3.0. In other words, as you move the thumb from the bottom of the slider to the top, the values on the corresponding axis are scaled by 0 to 3 times the default amplification ratio.

If a magnified drawing is too large to be shown entirely within the window, you can use the scroll bars to move the center of the drawing so that different parts of the drawing may become visible.

## Buttons

Another type of graphical tools on the visualization screen is buttons.



This is an on/off button. You use it to switch on and off the option the *name* represents.

Each time you point the cursor at the button and click the left-hand mouse button, it switches between on and off. The indicator on the button will turn green when the switch is "on".



This is an activator button. Each time you point the cursor at the button and click the left-hand mouse button, it activates the option the *name* represents.

## Exit from Visualization Screen



The function of this button is rather obvious. It allows you to quit the 3D visualization screen and return to the editor.

## Painting On/Off Switch



This button switches on and off the painting (coloring) of the cells. The default setting is on.

## Grid Lines (Wire Frames)



This button switches on and off the display of grid lines (wire frames).

If you switch on "Lines" and switch off "Paint", then the 3D surface is rendered as a wire frame, as illustrated in Fig. 9.14.

If you switch on "Lines" and the "Paint" button is also on, then the lines are shown in a dim shade and superimposed on the color drawing.

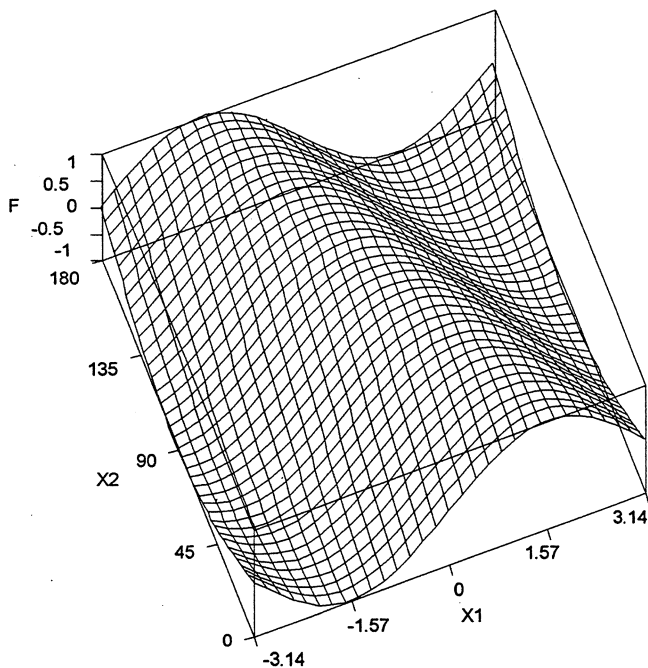


Fig. 9.14 Illustration of 3D wire frame.

### 3D Bar Chart



This button switches on and off the display of 3D bar chart. When it is on, instead of connecting adjacent grid points by line segments to form a surface, each grid point is drawn as a rectangular parallelepiped. The heights of the bars are proportional to the corresponding function values.

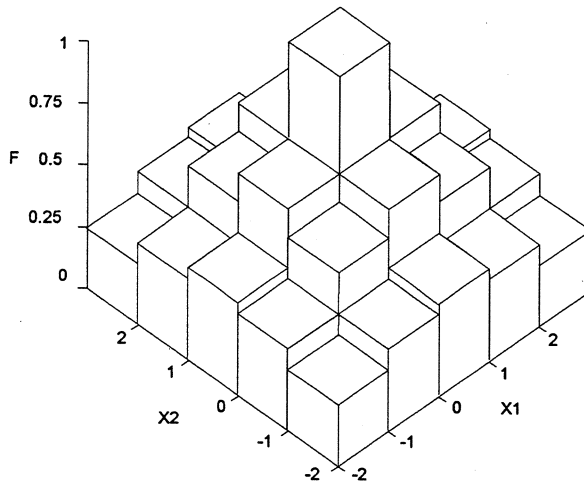


Fig. 9.15 Illustration of 3D bar chart:  $F = \exp(-0.5 \cdot \sqrt{X_1^2 + X_2^2})$ .

### The Frame



This button switches on and off the display of the frame.

### Axes and Ticks



This button switches on and off the display of the X-, Y- and Z-axes, including the ticks and labels.



## Printing

A rectangular button with a thin border and the word "Print" centered inside.

This button generates a print out of the current display. Clicking this button will cause the standard Windows printing dialog box to appear to allow you to select the printer, etc.

## Data Smoothing

A rectangular button with a thin border, a small black square on the left, and the word "Smooth" to its right.

This button switches on and off the data smoothing feature.

When this button is off (default), each cell of the drawing is filled with one color. The discrete nature of the data is clearly visible, since the changes of color always coincide with the grid lines (boundaries of the cells).

When the data smoothing feature is switched on, linear interpolation is applied to detect possible changes of color within a cell if the function values at the corners (grid points) are significantly different.

Data smoothing can dramatically improve the appearance of visualization. The accuracy of interpolation depends, of course, on the nonlinearity of the function as well as the step size of parameter sweeps.

## Contours

Contour This button switches on and off contour plotting.

Fig. 9.16 shows the contours of the minimax function

$$U(x_1, x_2) = \max \{f_1, f_2, f_3\}$$

$$f_1 = x_1^2 + x_2^2 - 1$$

$$f_2 = 3 - x_1^2 - x_2^2$$

$$f_3 = x_1 - x_2 + 3$$

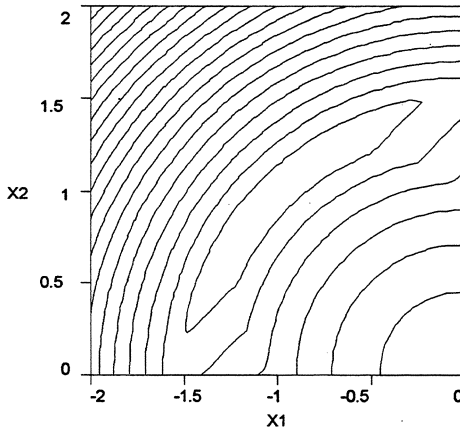
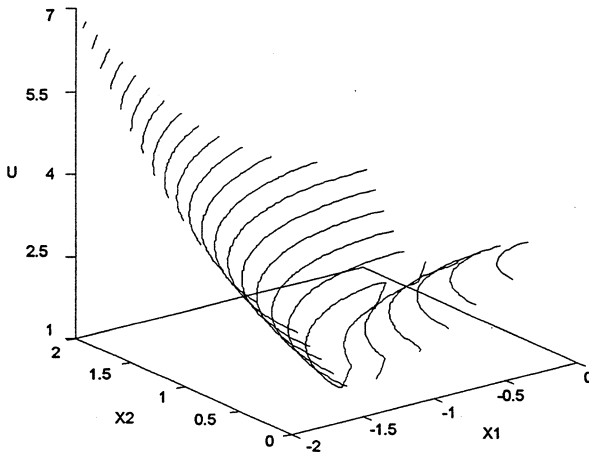


Fig. 9.16 Minimax contours

To obtain the conventional planar (2D) contour plot, as illustrated in Fig. 9.16, you must set the elevation angle to  $90^\circ$  (i.e., top view).

An elevation angle other than  $90^\circ$  will produce a 3D contour plot, as illustrated in Fig. 9.17.



*Fig. 9.17 Minimax contours in 3D.*

### Zoom-In on Part of the Contours for Details

By adjusting the color sliders (on the top-right-hand corner of the visualization screen, also see descriptions under the heading *Coloring Scheme* in this section), you can concentrate on part of the contours by reducing the increments in function value between adjacent contour lines.

### Accuracy of Contours

Contour generation utilizes linear interpolation within cells. Hence, the accuracy of the contours depends on the nonlinearity of the function as well as the step size of parameter sweeps. Broken contours may be observed near sharp kinks or narrow valleys.

Accuracy can be improved by increasing the data resolution, i.e., by using a smaller step size in the parameter sweeps.

### Contours for a Specific Set of Function Values

To display contours for a specific set of function values, you can use the Visual graphical View definition (see Chapter 10).

## Traces of Optimization Variables

If both sweep parameters are optimization variables, it is possible to superimpose on the contours a trace representing the values of the variables from iteration to iteration in a previous optimization.

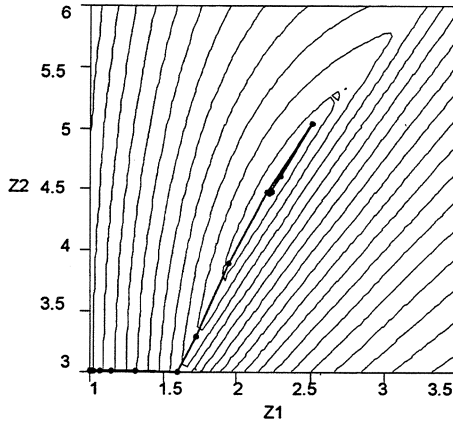


Fig. 9.18. Illustration of a trace superimposed on minimax contours.

To be able to plot traces, a record of the optimization variables must have been created and kept in the Trace block in the input file (see Chapter 11).



These buttons control the display of traces. They are shown in solid color (i.e., like all the other buttons) if the Trace block exists and is not empty. Otherwise the buttons are shown in a dim shade to indicate that trace data is not available.

## Animated Display of Trace



This button switches on the animated display of trace. The points in the Trace block are displayed in sequence, with a one-second pause between points. This allows you to see clearly the progress of optimization from iteration to iteration.

## Step-by-Step Display of Trace



This button allows you to manually control the pace of the display. Each time you press this button, the next point in the trace is displayed.

## 9.15 Host Acceleration Files

Simulation, especially nonlinear harmonic balance simulation of large circuits, can be very time-consuming. Host acceleration file (.haf file) is a feature built into OSA90 to avoid duplicated simulation.

Once the simulation is successfully completed according to the Sweep block of an input file, the results are saved in an encoded "host acceleration file". This information can then be retrieved during subsequent sessions instead of repeating the simulation.

The creation and usage of host acceleration files are transparent to the user. However, an explanation of the basic rules can be helpful.

### The Input File Must Have the Extension .ckt

A host acceleration file will be created only if the input file has the extension .ckt. The host acceleration file is named after the associated input file by replacing the extension .ckt with .haf. This rule is necessary to ensure a unique name for the host acceleration file.

For example, if you have two input files named `example.ckt` and `example.old`, then the host acceleration file for `example.ckt` will be named `example.haf`, but `example.old` will not have a host acceleration file. For this reason, it is strongly recommended that you name all the input files with the extension .ckt.

### Rules to Ensure the Data Is Up to Date

To ensure that the information stored is up to date, a "time stamp" is encoded into the host acceleration file which reflects the date and time of the corresponding input file. OSA90 always checks this time stamp in a host acceleration file before using it. Therefore, any operation which changes the date and time of the input file after a host acceleration file is created will invalidate the contents of the host acceleration file.

Editing or copying the file may modify the date and time of the files, therefore we recommend that you do not edit the files manually.

A new host acceleration file is created in three situations:

- ▶ A new input file is read, and simulated without editing or optimization.
- ▶ An input file is edited or optimized, then saved, and simulated.
- ▶ An input file is edited or optimized, then simulated, and saved.

## 9.16 Report Generation

The report generation feature allows you to embed numerical outputs from OSA90 into a text document.

### The Report Block

First, you create in the input file a Report block which contains

- ▷ the text of the document into which the numerical outputs are to be embedded;
- ▷ references to label names which will be substituted by OSA90 with the appropriate numerical values after simulation;
- ▷ printing formats.

Example:

```
Report
  John F. Smith.   Lab #4.   May 12, 1993.

          Results Calculated by OSA90
-----
Magnitude of the voltage is $MV2$ volts.
Phase of the voltage is $PV2$ degrees.
-----
End
```

where MV2 and PV2 are references to labels defined elsewhere in the input file.

The report generated by OSA90 may look like this:

```
John F. Smith.   Lab #4.   May 12, 1993.

          Results Calculated by OSA90
-----
Magnitude of the voltage is 2.355 volts.
Phase of the voltage is -71.08 degrees.
-----
```

where the label references are replaced by the appropriate numerical values.

➤ Examples of reports are provided in demo44.ckt, demo45.ckt and demo46.ckt.

## Generating Reports

The Report block in the input file defines the text and formats. To generate a report, select “Report” from the “Display” menu.

OSA90 will first perform simulation if necessary and then create a file containing the report and launch a separate window (“Notepad”) to display the file.

The numerical output file is created in the current directory under the name “report.txt”. You can make a copy of the file under a different name so it will not be overwritten by the next report.

## References to Labels

### Syntax:

```
$label$
```

where *label* must be the name of a label defined in the Expression or Model block (the label can be either predefined or user-defined).

### Example:

```
Report
```

```
The values of the variables are: $X1$, $X2$, $X3$
```

```
Magnitude of the gain (S21) is $MS21$
```

```
The second harmonic component of the output voltage:
$RVout[2]$ + j $IVout[2]$
```

```
End
```

where the labels X1, X2 and X3 represent user-defined labels, MS21 is a built-in circuit response label, and RVout[2] and IVout[2] are elements of the arrays RVout and IVout.

The report generated according to this example may look like:

```
The values of the variables are: 1.234, -56.78, 0.09012
```

```
Magnitude of the gain (S21) is 2.086
```

```
The second harmonic component of the output voltage:
7.654 + j 3.21
```

## Literal Dollar Signs

The dollar sign "\$" is used as a delimiter for commands in the Report block. To produce a literal character "\$" in the text, use "\$\$".

Example:

```
The calculated price is $$$Price$
```

Suppose that the label Price evaluates to 123.45, then the corresponding text in the generated report will be

```
The calculated price is $123.45
```

## References to Arrays

The Report block can contain references to labels which represent arrays.

For example, suppose that A is an array defined as A[4] and the Report block contains the following reference

```
The array A = [ $A$ ]
```

then the corresponding output in the generated report may look like this:

```
The array A = [ 1.234 5.678 9.012 3.456 ]
```

In other words, all the array elements are printed in the report at the location of the array reference.



## Formatting the Printing of Arrays

Instead of printing all the elements of an array on the same line, you can instruct OSA90 to print an array on multiple lines by specifying the number of array elements to be printed per line.

### Syntax:

```
$array % n$
```

where *array* must be a label name representing an array and *n* is a positive integer specifying the number of array elements to be printed per line.

For example, if A is an array of length 4, then

```
Array A = [ $A % 1$ ]
```

will lead OSA90 to produce

```
Array A = [ 1.234
            5.678
            9.012
            3.456 ]
```

Another example:

```
Array A = [ $A % 3$ ]
```

will lead to

```
Array A = [ 1.234 5.678 9.012
            3.456 ]
```

## Matrices Are Naturally Formatted

Matrices referenced in the Report block are automatically printed in their natural format when the report is generated.

For example, if Matrix[3,6] is referenced in the Report block as

```
We store the coefficients in a 3 x 6 matrix as
$Matrix$
```

then the following will be generated in the report:

```
We store the coefficients in a 3 x 6 matrix as
0.1 0.2 0.3 0.4 0.5 0.6
0.7 0.8 0.9 1.0 1.1 1.2
1.3 1.4 1.5 1.6 1.7 1.8
```

As in the case of one-dimensional arrays, you can specify explicitly the number of elements to be printed per line.

Example:

```
$Matrix % 3$
```

will produce the following results:

```
0.1 0.2 0.3
0.4 0.5 0.6
0.7 0.8 0.9
1.0 1.1 1.2
1.3 1.4 1.5
1.6 1.7 1.8
```

You can also have all the elements of a matrix printed on the same line by specifying the number of elements per line to be the overall size of the matrix. For example:

```
$Matrix % 18$
```

will lead to

```
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8
```

## Floating-Point Printing Formats

### Syntax:

```
$%format$
```

where *format* can be any valid floating-point format specification defined by the C language, i.e., acceptable to the standard C function `printf()`.

A format specification controls the printing of the labels following it until another format is specified. The default format is "%g".

For example, suppose that a label X1 evaluates to 123.456. The following formats in the Report block

```
$%e$ "e" format = $X1$
$%8.1f$ "f" format = $X1$
$%g$ "g" format = $X1$
```

will produce

```
"e" format = 1.234560e+02
"f" format = 123.5
"g" format = 123.456
```

 Consult a standard C language reference book for more details on available formats.



OSA90 treats the values of all labels as floating-point numbers. Therefore, you can only specify floating-point formats. You must not specify formats for integers ("%d"), hexadecimal numbers ("%x"), new-line controls ("\n"), etc. Otherwise the results are unpredictable.

## Specify the Number of Digits

You can specify the minimum width and precision of floating-point printing within a format command.

For example, "%8.2f" specifies printing using the "f" format with a minimum of 8 characters and with 2 digits after the decimal point.

Another example, "%.5g" specifies printing using the "g" format with a maximum number of 5 significant digits.

## Specify Left-Justified Printing

If you specify the field width in the floating-point printing format and the number of digits is less than the field width, the output is right-justified within the field by default.

You can have the output left-justified using the "-" flag after the "%". For example,

```

%%8.1f$ By default      : $X1$
%%-8.1f$ Left-justified: $X1$

```

The results are

```

By default      :      123.5
Left-justified: 123.5

```

## Specify a Leading Blank for Positive Numbers

If the values of the labels to be printed in the report include both positive and negative numbers, the outputs may not be aligned properly.

For example, suppose that you wish to include a matrix Matrix[3,3] in the report. By default, it may be printed as

```

1.01 -2.02 3.03
-4.04 -5.05 6.06
7.07 8.08 -9.09

```

You can force a leading blank to be prefixed to a printed number if it does not begin with a sign. To do this, you include in the format a blank (" ") immediately after the "%":

```

%% g$ $Matrix$

```

The resulting outputs will be

```

1.01 -2.02 3.03
-4.04 -5.05 6.06
7.07 8.08 -9.09

```

## Labels in the Report and Sweep Blocks

Before generating a report, OSA90 must first evaluate the labels referenced in the Report block by simulation.

Simulation in OSA90 is organized according to the Sweep block. Therefore, labels which represent circuit responses (both built-in and user-defined) must first be defined as output labels in the Sweep block before they can be referenced in the Report block.

An exception is granted for labels which are defined by generic expressions and do not require circuit simulation nor Datapipe connections. Such generic labels can be included in the Report block without being included in the Sweep block.

## Reporting Parameter Sweeps

The Report block may contain labels which are functions of a parameter swept in the Sweep block. In this case, each label has not a single value but a set of values. For instance, you may sweep the frequency and wish to report the values of MS21 versus frequency.

### Syntax:

```
$( text and commands to be repeated for each sweep step )$
```

### Example:

```
$( Frequency = $FREQ$           |S21| = $MS21$ )$
```

Assuming that the Sweep block contains the following sweep set

```
FREQ: from 2 to 10 step=1 .... MS21 ...;
```

the generated report will contain the corresponding results

```
Frequency = 2      |S21| = 0.2364
Frequency = 3      |S21| = 0.2502
Frequency = 4      |S21| = 0.2626
Frequency = 5      |S21| = 0.2764
Frequency = 6      |S21| = 0.2919
Frequency = 7      |S21| = 0.309
Frequency = 8      |S21| = 0.3275
Frequency = 9      |S21| = 0.3469
Frequency = 10     |S21| = 0.367
```

where the values of MS21 are for the purpose of illustration only.

## Formatted Report of Parameter Sweeps

You can incorporate formatting commands within the sweep to improve the appearance of the report.

For example, if the Report block contains the following section

```

Small-Signal Responses
-----
Frequency   |S11|      Gain
-----
${ %2g$ $FREQ$GHz %9.4f$ $MS11$ %6.2f%$ $Gain$dB }$
-----

```

and the Sweep block contains the following sweep set

```
FREQ: from 2 to 10 step=1 MS11 Gain ...;
```

then the corresponding section in the generated report will look like this:

```

Small-Signal Responses
-----
Frequency   |S11|      Gain
-----
2GHz       1.0034   -7.24dB
3GHz       1.0177   0.55dB
4GHz       1.0523   6.25dB
5GHz       1.0809   9.76dB
6GHz       1.0496  10.74dB
7GHz       0.9877  10.32dB
8GHz       0.9336   9.54dB
9GHz       0.8884   8.81dB
10GHz      0.8475   8.23dB
-----

```

## Repeating Multiple Lines When Reporting Parameter Sweep

You can enclose a number of lines within `{ ... }`, then those lines will be repeated as a block for each sweep step when the report is generated.

Example:

```
{ At frequency=$FREQ$
  we have |S11|=$MS11$
    and Gain=$Gain$dB
}$
```

The corresponding report will contain

```
At frequency=2
  we have |S11|=1.0034
    and Gain=-7.24dB

At frequency=3
  we have |S11|=1.0177
    and Gain=0.55dB

... ..

At frequency=10
  we have |S11|=0.8475
    and Gain=8.23dB
```

## Matching Sweeps in the Report and Sweep Blocks

The Sweep block may contain a number of sweep sets which may represent different simulation types (AC, DC and HB). OSA90 automatically matches the labels referenced in the Report block to the appropriate sweep sets in the Sweep block.

For example, consider a sweep set in the Report block as

```
{ Frequency=$FREQ$ |S11|=$MS11$ Gain=$Gain$ }$
```

and four sweep sets in the Sweep block as follows

```
AC: FREQ: from 1 to 10 step=1 MS11 Gain;
AC: FREQ: from 1 to 10 step=1 MS22 PS22;
DC: VD: from 0 to 4 step=0.1 Iout_DC;
AC: FREQ: from 12 to 20 step=2 MS11 PS11 MS22 PS22 Gain;
```

The report will contain the results of the first and the fourth sweep sets.

## Reporting Two-Dimensional Sweeps

If a sweep set in the Report block matches a sweep set in the Sweep block which contains two sweep labels, then the generated report will contain all the steps in the two-dimensional sweep. As described in Section 9.4, the first sweep label is considered as the outer sweep and the second sweep label is considered as the inner sweep.



# 10

## Graphical Views

<b>10.1 Overview</b> .....	10-1
<b>10.2 Xsweep Views</b> .....	10-2
<b>10.3 Specifications Shown In Views</b> .....	10-14
<b>10.4 Select Views for Display</b> .....	10-17
<b>10.5 Parametric Views</b> .....	10-18
<b>10.6 Waveform Views</b> .....	10-21
<b>10.7 Smith Chart Views</b> .....	10-26
<b>10.8 Polar Plot Views</b> .....	10-29
<b>10.9 Visualization Views</b> .....	10-32



## 10

## Graphical Views

## 10.1 Overview

The default display setting, such as the scales of the axes, the colors and draw types, may not always be satisfactory. The dialog boxes for the “Display” menu options, such as “Zoom” and “Style”, give you certain flexibility to modify the display settings, as described in Chapter 9.

However, if it takes many experiments with the various parameters to obtain the right setting, you would not wish to repeat it every time you need to view the same display.

The feature of user-definable Graphical Views allows you to define the setting of one or more displays. The graphical view definitions are recorded in the input file and therefore will not be lost after you exit the program.

Graphical Views also allow you greater freedom in terms of selecting output labels and choosing colors, shapes, scales, etc.

Furthermore, two of the “Display” menu options, namely “Waveform” (time-domain waveforms) and “SmithChart” (Smith charts and polar plots) can only be defined through Graphical Views.



Graphical Views are simply referred to as “Views” where the context permits.

One or more Views can be defined as a part of a sweep set.

**Syntax:**

```
basic_sweep_set
{view_type view_definition}
...
{view_type view_definition};
```

where *basic\_sweep\_set* represents the basic definitions of a sweep set (sweep type, sweep labels, parameter labels and output labels), *view\_type* is a keyword to identify the view type and *view\_definition* defines the view.

The basic definitions of sweep sets are described in Chapter 9.

TABLE 10.1 VIEW TYPES

Keyword	Type of Display
Xsweep	responses versus parameter sweeps
Parametric	parametric plots of responses
Waveform	time-domain waveforms
Smith	Smith charts of complex responses
Polar	Polar plots of complex responses
Visual	3D visualization and contours

The various Views share many common features. These common features are illustrated using the Xsweep View as an example.

## 10.2 Xsweep Views

The Xsweep View has many features. To make it easier to follow the descriptions, the features are divided into a number of subsets and presented in a progressive sequence, starting from most basic syntax.

### Syntax:

```
{ XSWEEP Y=output_label1 & output_label2 ... & output_labeln
  X=sweep_label }
```

where *output\_label1*, ..., *output\_labeln* are a subset of the output labels in the sweep set. You can also specify ALL for Y to include all the output labels. If Y is omitted, the first output label in the sweep set is assumed. *sweep\_label* is one of the sweep labels in the sweep set. If *sweep\_label* is omitted, X defaults to the inner sweep label.

### Example:

```
Sweep
AC:  FREQ: from 1 to 20 step=1  VD: 0 1.5 2 3 5
     MS11, PS11, MS21, PS21, MS12, PS12
     {XSWEEP X=FREQ Y=MS21 & MS11 & MS12};
...
End
```

The Xsweep display according to the View defined in the example will plot simultaneously three output labels, namely MS21, MS11 and MS12, versus the sweep label FREQ.

## X-axis of Xsweep Views

Xsweep View requires that at least one sweep label be defined in the sweep set. If there are two sweep labels, you can specify either one for the X-axis of the View. By default, the inner sweep label is selected.

In the preceding example, FREQ is selected as the X-axis. Otherwise, the inner sweep label VD would be the default choice for the X-axis.

## Y-axis of Xsweep Views

You can specify one or more output labels. Multiple output labels are separated by an ampersand "&". You can specify ALL to include all the output labels. If no output label is explicitly specified, the first output label in the sweep set is assumed.

The output labels can only be a subset of the output labels defined in the sweep set, i.e., labels that are not included in the sweep set cannot be included in the Views associated with that sweep set. For instance, consider the above example: the label MS22 is not included as an output label in the sweep set, and therefore cannot be included in the View.

## Colors and Shapes of View Output Labels

### Syntax:

```
{ XSWEEP  Y=output_label1.color1.shape1 & output_label2.color2.shape2
          ... & output_labeln.colorn.shapen
          X=sweep_label }
```

where *output\_label1*, ..., *output\_labeln* are a subset of the output labels in the sweep set, *color1*, ..., *colorn*, *shape1*, ..., *shapen* are the associated display attributes.

*sweep\_label* is one of the sweep labels in the sweep set.

At least one output label is required; all the other entries are optional.

The colors defined are listed in Table 10.2. Table 10.3 lists the available shapes.

TABLE 10.2 COLORS IN THE DEFAULT COLOR MAP FILE

Black,	White,	Red,	Cream,	DarkGreen,
Yellow,	Green,	Blue,	Pink,	Brown

TABLE 10.3 SHAPES FOR GRAPHICAL VIEWS

Keyword	Shape	Keyword	Shape
CIRCLE	○	SQUARE	□
TRIANGLE	△	DOT	●
POINT	·	DIAMOND	◇
BAR	vertical bar	CROSS	+
HAT	∧	V	∇
LEFT	<	RIGHT	>
X	×		

Example:

```
ANGLE: from 0 to (2 * PI) N=50 SIN_ANGLE COS_ANGLE
{XSWEEP X=ANGLE
  Y=SIN_ANGLE.white.circle & COS_ANGLE.yellow.triangle};
```

where SIN\_ANGLE and COS\_ANGLE are output labels. The display of the View is illustrated in Fig. 10.1, although the colors "white" and "yellow" cannot be shown distinctly.

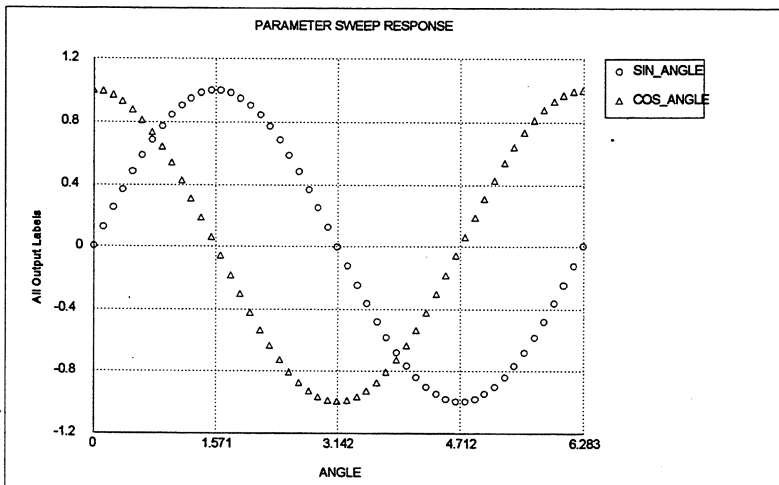


Fig. 10.1 View display with colors and shapes specified for the output labels.

When a shape is specified for an output label, the corresponding data is plotted as discrete (unconnected) shapes. By default, the data is shown as a curve.

The most obvious benefit of using colors and shapes is to make it easier to separate the data belonging to different output labels.

Consider the following example:

```
DC: VG: from -2 to 0 step=0.5
VD: from 0 to 2 step=0.5 from 3 to 12 step=1
ID_MA ID_DATA
{XSWEEP X=VD VG=ALL
Y=ID_MA.white & ID_DATA.white.circle};
```

where ID\_MA and ID\_DATA represent the calculated and measured DC current responses. The View is designed to show the match between ID\_MA and ID\_DATA by plotting them in the same color (white) but in different shapes (curve and circle). The corresponding display is illustrated in Fig. 10.2.

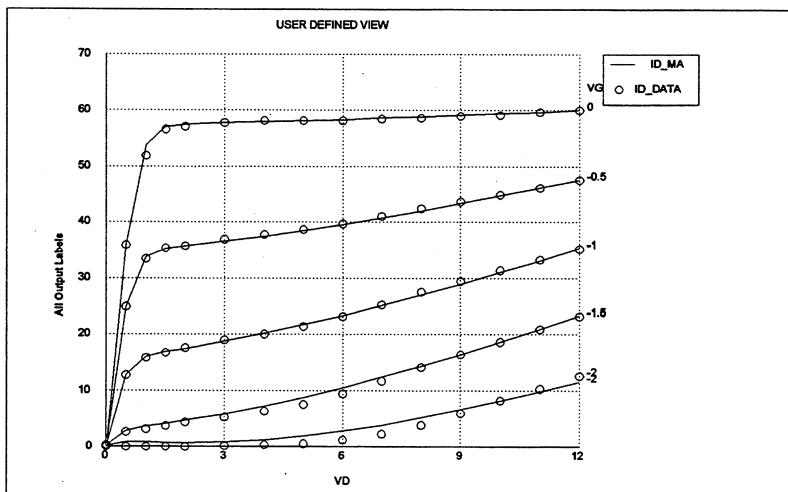


Fig. 10.2 View display of data matching.

## Select Values for the Other Sweep Label

In cases where the sweep set contains two sweep labels, one is selected for the X-axis, and for the other sweep label you can select one or all of the values available:

### Syntax:

```
{ XSWEEP Y=output_label1 & output_label2 ... & output_labeln
  X=sweep_label other_sweep_label=value }
```

where *output\_label1*, ..., *output\_labeln* are a subset of the output labels in the sweep set. At least one output label must be specified. *sweep\_label* is one of the two sweep labels in the sweep set. If *sweep\_label* is omitted, X defaults to the inner sweep label. *other\_sweep\_label* represents the sweep label not chosen for the X-axis, and it can be assigned either one of the available values or the keyword "ALL".

### Example:

```
AC: FREQ: from 1 to 20 step=1 VD: 0 1.5 2 3 5
    MS11, PS11, MS21, PS21, MS12, PS12
    {XSWEEP Y=MS21 & MS11 & MS22 X=VD FREQ=10};
    {XSWEEP Y=PS11 & PS22 X=FREQ VD=ALL}
```

where two Views are defined.

The first View selects VD for the X-axis, and specifies a value of 10 for the other sweep label FREQ. In other words, the View is designed to display the output labels versus VD while holding the other sweep label to a constant, namely FREQ = 10.

The second View is designed to display a different subset of output labels versus FREQ for all the values of VD. The display will show a family of curves, each curve representing one output label versus FREQ at a given value of VD.

If the other sweep label is omitted from the View definition (i.e., it is not assigned a value), then by default the first value in its sweep range is used.



The selection of a sweep label for the X-axis and a value or values for the other sweep label is identical in concept to the selection through dialogboxes, as described in Chapter 9. Using Views, you have greater freedom to group a subset of output labels for display.



## Titles of Views

### Syntax:

```
{ XSWEEP Y=... X=...
  TITLE=view_title X_TITLE=x_title Y_TITLE=y_title }
```

where *view\_title*, *x\_title* and *y\_title* are character strings.

### Example:

```
DC: VG: from -2 to 0 step=0.5 VD: from 0 to 12 step=1
ID MA ID_DATA
{XSWEEP X=VD VG=ALL Y=ID_MA & ID_DATA
  TITLE="Match of Simulated and Measured DC Currents"
  X_TITLE="Drain Bias Voltage (V)"
  Y_TITLE="Drain Current (mA)"};
```

## Display Scale of Views

### Syntax:

```
{ XSWEEP Y=... X=...
  XMIN=xmin XMAX=xmax YMIN=ymin YMAX=ymax
  NXTICKS=nxticks NYTICKS=nyticks }
```

where *xmin*, *xmax*, *ymin* and *ymax* are numerical constants defining the display scale, *nxticks* and *nyticks* are integers specifying the number of ticks (grids) on the axes.

The meaning of XMIN, XMAX, YMIN, YMAX, NXTICKS and NYTICKS is identical to the meaning of the corresponding parameters in the zoom dialog box (Chapter 9).

### Example:

```
DC: VG: from -2 to 0 step=0.5 VD: from 0 to 12 step=1
ID MA ID_DATA
{XSWEEP X=VD VG=ALL
  Y=ID_MA.white & ID_DATA.white.circle
  YMIN=0.0 YMAX=70 NYTICKS=7};
```

You can define logarithmic scale for Views by specifying "LOG" after the View keywords "NXTICKS" and "NYTICKS". Example:

```
{ Xsweep ... YMIN=0.01 YMAX=10 NYTICKS=LOG ... }
```

This defines a Xsweep View in which the Y-axis is displayed in logarithmic scale.

## Position of Legend

When two or more responses are displayed in the same figure, a legend is shown to indicate the different responses. For example, the legend in Fig. 10.2 indicates that the two responses ID\_MA and ID\_DATA are displayed as solid lines and circles, respectively.

By default, the legend is positioned to the right of the plot. Using View definition, you can change the position of the legend to the top of the plot.

### Syntax:

```
{ XSWEEP Y=... X=... ...
  LEGEND=TOP }
```

Example:

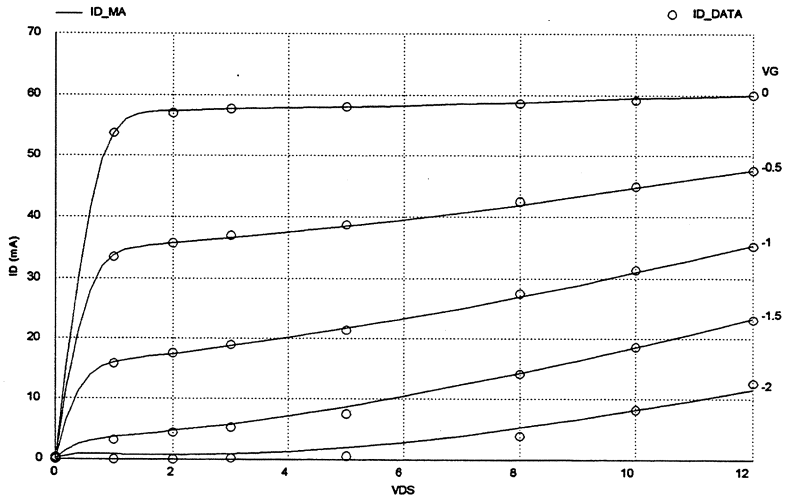


Fig. 10.3 Legend positioned on the top of the plot.

You can also position the legend inside the box of the plot.

**Syntax:**

```
{ XSWEEP Y=... X=... ...
  LEGEND=Inside }
```

Example:

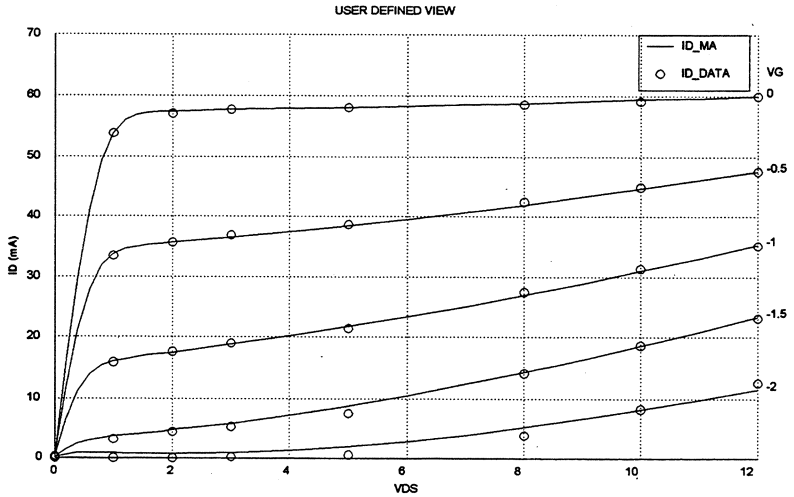


Fig. 10.4 Legend positioned inside the box of the plot.

You can even suppress the legend completely.

**Syntax:**

```
{ XSWEEP Y=... X=... ...
  LEGEND=OFF }
```

Example:

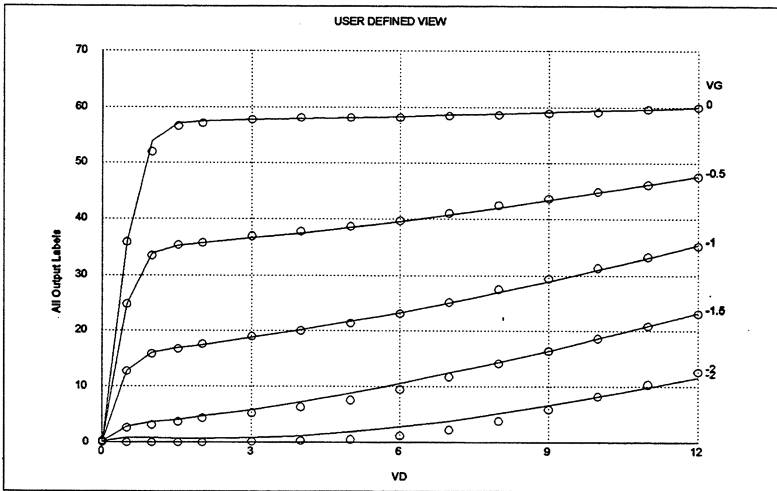


Fig. 10.5 View display without a legend.

## Display on Two Y-Axes

In a display of multiple responses, by default all the responses are scaled to the same Y-axis which is labelled on the left-hand side of the plot. This can be inconvenient if the actual scale (range of values) of the responses are very different. In such situations, you can define a View to show the responses on two separate scales, represented by the left-hand Y-axis and the right-hand Y-axis, respectively.

### Syntax:

```
{ XSWEEP Y=output_label11 & ... & output_label1n &&
  output_label21 & ... & output_label2n
  Y2MIN=y2min Y2MAX=y2max Y2_TITLE=y2_title
  ... }
```

where *output\_label11*, ..., *output\_label1n* will be plotted with one scale, and *output\_label21*, ..., *output\_label2n* will be plotted with a different scale, represented by the left-hand and right-hand Y-axes, respectively.

The two groups of labels are separated by "&&".

### Example:

```
HB: FREQ=7GHZ Pin: from -3dBm to 15dBm step=0.2dBm
  Conversion_Gain Spectral_purity
  {Xsweep Y=Conversion_gain && Spectral_purity
    Ymin=-40 Ymax=10 NYticks=5 NXticks=6 Y2min=0 Y2max=25};
```

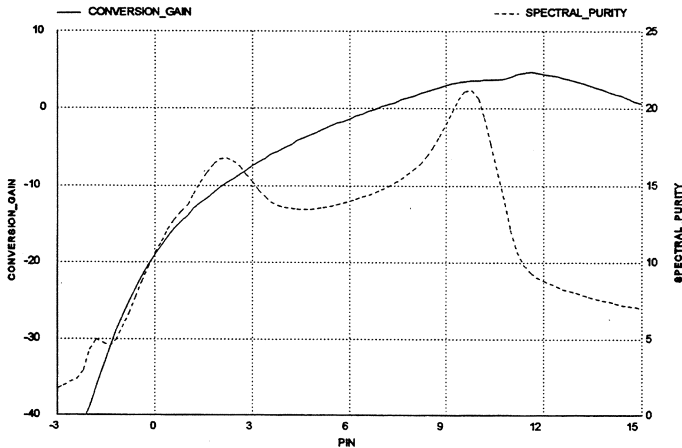


Fig. 10.6 View display with two separate Y-axes.

## User Comments Defined in Views

## Syntax:

```
{ XSWEEP Y=... X=... ...
  COMMENTS=comment1 & comment2 & ... commentm }
```

where *comment1*, ..., *commentm* are character strings.

## Example:

```
AC: FREQ: from 2GHZ to 18GHZ step=1GHZ RS21 ...
  { Xsweep X=FREQ Y=RS21 ...
    Comments= "gate bias -1.74V" &
              "drain bias 4.00V" &
              "model FETM" & "(c) OSA Inc." };
```

Comments should be enclosed in quotation marks. Multiple comments must be separated (delimited) by an ampersand "&".

Comments are displayed in the top-right corner of the screen together with the legends for the graphical display, as illustrated in Fig. 10.7. Long comments may be truncated due to the size of the display.

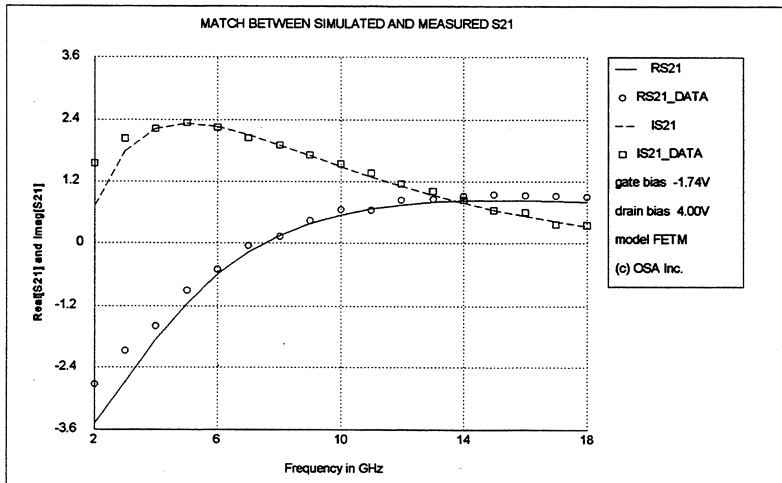


Fig. 10.7 Xsweep View.

## Summary of Xsweep Views

## Syntax

```
{ XSWEEP  Y=output_label1.color1.shape1 & output_label2.color2.shape2
          ... & output_labeln.colorn.shapen
  X=sweep_label      other_sweep_label=value
  TITLE=view_title  X_TITLE=x_title      Y_TITLE=y_title
  XMIN=xmin         XMAX=xmax           YMIN=ymin
  YMAX=ymax        NXTICKS=nxticks     NYTICKS=nyticks
  COMMENTS=comment1 & comment2 & ... commentm
  LEGEND=legend_pos }
```

*output\_label1*, ..., *output\_labeln* are a subset of the output labels in the sweep set, and *color1*, ..., *colorn*, *shape1*, ..., *shapen* specify the associated display attributes.

*sweep\_label* is one of the sweep labels in the sweep set, and if the sweep set contains two sweep labels, *other\_sweep\_label* is assigned *value* which is one of the available values defined in the sweep set or the keyword "ALL" for all the available values.

*view\_title*, *x\_title* and *y\_title* are character strings.

*xmin*, *xmax*, *ymin* and *ymax* are numerical constants defining the display scale.

*nxticks* and *nyticks* are integers specifying the number of ticks (grids) on the axes.

*comment1*, ..., *commentm* are character strings.

*legend\_pos* can be ON, OFF, TOP or INSIDE.

All the entries are optional.

## Example:

```
AC:  .FREQ: from 2GHZ to 18GHZ step=1GHZ
      RS21, IS21, RS21_data, IS21_data
      { Xsweep X=FREQ
        Y=RS21.white & RS21_data.white.circle &
          IS21.yellow & IS21_data.yellow.square
        title="MATCH BETWEEN SIMULATED AND MEASURED S21"
        x_title="Frequency in GHz"
        y_title="Real[S21] and Imag[S21]"
        NXTicks=4 Ymin=-3.6 Ymax=3.6 NYTicks=6
        comments= "gate bias -1.74V" &
                  "drain bias 4.00V" &
                  "model FETM" & "(c) OSA Inc."
      };
```

The View is illustrated in Fig. 10.7.

## 10.3 Specifications Shown in Views

You can conveniently superimpose design specifications on graphical Views.

Syntax:

```
{ view_type ... SPEC=(from x1 to x2, < x3) ... }
```

where  $x1$  and  $x2$  define the range and  $x3$  is an upper specification.

Use ">" instead of "<" for a lower specification.

Use "=" for a single specification (i.e., a goal to be matched exactly).

Multiple specifications can be defined using "&".

Example:

```
{ XSWEEP Y=MS21_DB Nxticks=8  
  SPEC=(from 6 to 18, > 7) & (from 6 to 18, < 9) }
```

This View includes an upper specification and a lower specification. The corresponding display is illustrated in Fig. 10.8.

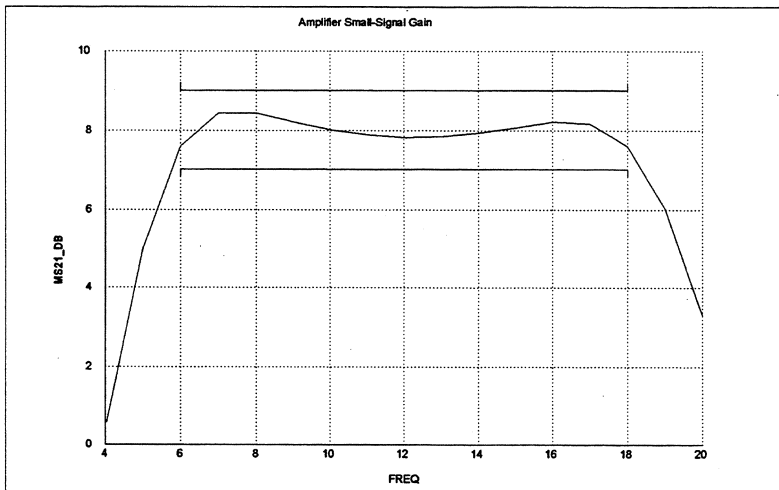


Fig. 10.8 Illustration of specifications shown in Views.



## Variations of Range Definitions

The definition

$$\text{SPEC}=(\text{from } x, < y)$$

is equivalent to

$$\text{SPEC}=(\text{from } x \text{ to } x_{\text{max}}, < y)$$

where  $x_{\text{max}}$  represents the maximum of the X-axis of the display. In other words, the specification extends to the right edge of the plot.

Similarly,

$$\text{SPEC}=(\text{to } x, < y)$$

is equivalent to

$$\text{SPEC}=(\text{from } x_{\text{min}} \text{ to } x, < y)$$

where  $x_{\text{min}}$  represents the minimum of the X-axis of the display.

The range definition can be entirely omitted, such as

$$\text{SPEC}=( < y)$$

then it is equivalent to

$$\text{SPEC}=(\text{from } x_{\text{min}} \text{ to } x_{\text{max}}, < y)$$

In other words, the specification extends across the whole X-axis.

## Specification at a Single Point

To define a specification at a single point, use

$$\text{SPEC}=(\text{at } x, < y)$$

## Multiple Specifications in One View

Multiple specifications can be included in one View, separated by "&". For example:

$$\text{SPEC}=(\text{from } 6 \text{ to } 18, < 9) \ \& \ (\text{from } 6 \text{ to } 18, > 7) \ \& \\ (\text{to } 4, < 1) \ \& \ (\text{from } 20, < 1)$$

## Colors for Specifications

You can name the colors for displaying the specifications:

```
SPEC=(from x1 to x2, < x3).color
```

Example:

```
SPEC=(from 6 to 18, < 9).yellow
```

You can choose from the default set of colors as well as any additional colors defined in the current color map (see Chapters 10 and 15).

## Vertical Specifications

By default, specifications are assumed to be defined with respect to the Y-axis. You can define "vertical" specifications with respect to the X-axis.

Example:

```
SPEC=(from -17 to 17, > -3).vertical &
      (to -25, < -30).vertical & (from 25, < -30).vertical
```

The display is illustrated in Fig. 10.9.

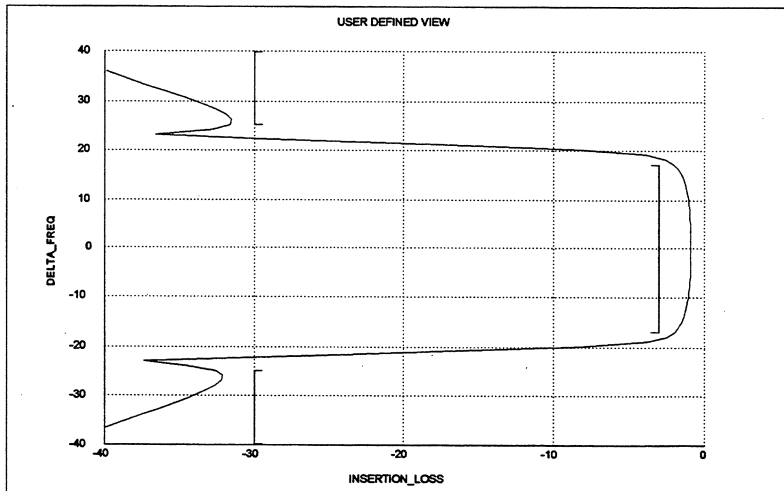
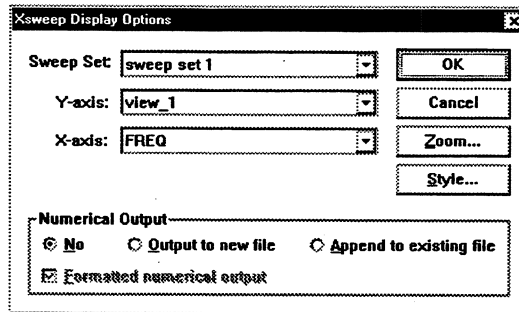


Fig. 10.9 Illustration of vertical specifications in Views.

## 10.4 Select Views for Display

Views are displayed under the corresponding "Display" menu option. In particular, "Xsweep" Views defined in the input file are available for display when you click on the "Xsweep" option under the "Display" menu.

When you invoke a display option, a dialog box allows you to select a sweep set from those defined in the Sweep block. If the selected sweep set of the input file (by default, the first sweep set) contains Views, then the Views will be listed in the "Y-axis" drop-down list box as "view\_1", "view\_2", etc. For example,



Views are *additional* choices for the "Y-axis" option, i.e., the individual output labels can be selected for display as well, as described in Chapter 9.

## 10.5 Parametric Views

### Syntax:

```
{ PARAMETRIC P=parametric_label  other_sweep_label=value
  X=x_axis_label
  Y=output_label1.color1.shape1 &
  output_label2.color2.shape2 & ...
  output_labeln.colorn.shapen
  TITLE=view_title  X_TITLE=x_title  Y_TITLE=y_title
  XMIN=xmin          XMAX=xmax      YMIN=ymin
  YMAX=ymax          NXTICKS=nxticks  NYTICKS=nyticks
  COMMENTS=comment1 & comment2 & ... commentm
  LEGEND=legend_pos }
```

*parametric\_label* must be specified as one of the sweep labels in the sweep set, and if the sweep set contains two sweep labels, *other\_sweep\_label* is assigned *value* which can be one of the available values defined in the sweep set or the keyword "ALL".

*x\_axis\_label* must be one of the output labels in the sweep set.

*output\_label1*, ..., *output\_labeln* are a subset of the output labels in the sweep set, and *color1*, ..., *colorn*, *shape1*, ..., *shapen* specify the associated display attributes.

*view\_title*, *x\_title* and *y\_title* are character strings.

*xmin*, *xmax*, *ymin* and *ymax* are numerical constants defining the display scale.

*nxticks* and *nyticks* are integers specifying the number of ticks (grids) on the axes.

*comment1*, ..., *commentm* are character strings.

*legend\_pos* can be ON, OFF, TOP or INSIDE.

All the entries are optional.

### Example:

```
Sweep
  Angle: from 0 to (2 * PI) N=100
  Sin_Angle, Cos_Angle
  {Parametric P=Angle X=Cos_Angle Y=Sin_Angle};
  ...
End
```

The parametric display according to the View defined in the example will show a complete circle (Sin\_Angle and Cos\_Angle represent the Cartesian coordinates of the points on a circle as Angle is swept).

A parametric View is very similar to a Xsweep View except that the sweep label does not appear explicitly in the display, instead, the sweep label serves as the parametric parameter and both the X-axis and the Y-axis are output labels.

➤ Parametric display is described in detail in Chapter 9.

## Parametric Labels

Parametric View requires that at least one sweep label be defined in the sweep set. If there are two sweep labels, you can specify either one as the parametric label of the View. By default, the inner sweep label is selected.

## X-axis of Parametric Views

The X-axis of a parametric View can be specified by any one of the output labels in the sweep set. By default, the first output label is selected.

## Y-axis of Parametric Views

You must specify at least one output label per View (there is no default). Multiple output labels are separated by an ampersand "&".

The output labels can only be a subset of the output labels defined in the sweep set, i.e., labels that are not included in the sweep set cannot be included in the Views associated with that sweep set.

You can specify a color and/or a shape for each Y-axis output label, as described in Section 10.2 (available colors and shapes are listed in Tables 10.2 and 10.3, respectively).

## Select Values for the Other Sweep Label

In cases where the sweep set contains two sweep labels, one is selected as the parametric label, and for the other sweep label you can select one of the values available or choose to simultaneously display all the available values using the keyword "ALL". If omitted from the View definition, the other sweep label is assigned the first value in its sweep range.

Example:

```

Expression
  Angle: 0;
  Center: 0;
  Cos_Angle = cos(Angle) + Center;
  Sin_Angle = sin(Angle) - Center;
End

Sweep
  Center: from 0 to 5 step=1
  Angle: from 0 to (2 * PI) N=100 Sin_Angle Cos_Angle
  { Parametric P=Angle Center=ALL X=Cos_Angle Y=Sin_Angle
    Title="PARAMETRIC VIEW OF CIRCLES"
    Xmin=-4 Xmax=8 NXticks=12 X_title="cos(Angle) + Center"
    Ymin=-7 Ymax=2 NYticks=9 Y_title="sin(Angle) - Center" };
End

```

where two sweep labels are defined: Angle is the parametric label for displaying the circles and Center shifts the center of each circle to a different location.

The View is designed to display all the circles, as illustrated in Fig. 10.10.

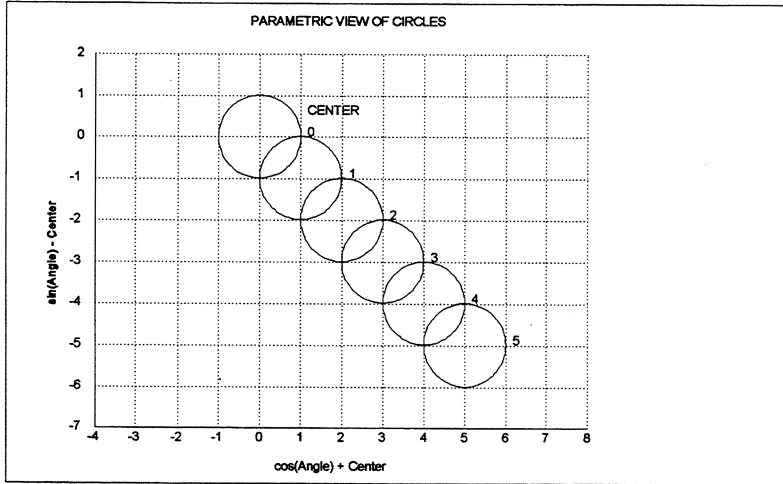


Fig. 10.10 Parametric display of circles.

### Parametric View Scales, Titles, Comments and Legend

You can define display scales ("zoom"), titles and comments for parametric Views, as described in Section 10.2. The example shown in Fig. 10.7 also illustrates these features.

### Select Views for Display

Parametric Views are displayed through the "Parametric" menu option under the "Display" menu.

If the current sweep set (which you select the dialog box) contains Views, then the Views will be listed in the "Y-axis" drop-down list box as "view\_1", "view\_2", etc.

☞ See also Section 10.4.

## 10.6 Waveform Views

The built-in responses calculated by the harmonic balance simulation in OSA90 are complex frequency-domain spectra of the voltages and currents defined in the circuit model.

To obtain time-domain waveforms from the frequency-domain spectra, you can utilize the built-in DFT transformation (DFT\_FT) as described in Chapters 4 and 6.

For example, suppose that MVout[0:N\_Spectra] and PVout[0:N\_Spectra] are response arrays which contain the magnitudes and phases of a voltage spectrum, where N\_Spectra is the predefined label referring to the number of spectral components (see Chapter 6).

You can obtain the time-domain waveform using the built-in transformations:

```
MP2RI (MVout, PVout, RVout[0:N_Spectra], IVout[0:N_Spectra]);
Vout_T[nt] = DFT_FT (RVout, IVout);
```

where RVout and IVout are the spectrum in rectangular form, Vout\_T contains the time-domain waveform and  $nt$  is the desired number of time points.

To display the waveform, you may define a sweep set as:

```
HB: ... K: from 1 to nt step=1 Vout_T[K] ...;
```

and then use Xsweep to plot Vout\_T[K] versus K.

### Display Waveforms Using Views

Displaying waveforms from spectra by postprocessing, as shown in the preceding discussion, involves quite a few steps. The necessity of an explicit sweep of the time-domain array index (namely K in the above example) can be especially inconvenient, considering the limit of two sweep labels for each sweep set.

The waveform View can display time-domain waveforms directly from frequency-domain spectra in a convenient way. The necessary DFT transformations are automatically included and carried out in a way that is transparent to the user.

**Syntax:**

```
{ WAVEFORM SPECTRUM=(mag_array1, ang_array1).name1.color1.shape1 &
                    (mag_array2, ang_array2).name2.color2.shape2 &
                    (mag_arrayn, ang_arrayn).namen.colorn.shapen
  TMIN=tmin TMAX=tmax NT=nt
  sweep_label1=value1 sweep_label2=value2 }
```

*mag\_arrayi* and *ang\_arrayi*,  $i = 1, 2, \dots, n$ , represent  $n$  pairs of output arrays specified in the sweep set, each pair represents the magnitudes and phases of a frequency-domain spectrum response, and all the arrays must have the same dimension of [0:N\_SPECTRA].

*name1*, *name2*, ..., *namen* are character strings identifying the time-domain responses, *color1*, ..., *colorn*, *shape1*, ..., *shapen* specify the associated display attributes.

*tmin* and *tmax* are constants defining the time interval (default: *tmin* = 0, *tmax* = 1),

*nt* is a positive integer specifying the number of time points (default: *nt* = 100).

*sweep\_label1* and *sweep\_label2* represent the sweep labels in the sweep set (if defined),

*value1* and *value2* are selected values for *sweep\_label1* and *sweep\_label2*, respectively, *value1* or *value2* (but not both) can be the keyword "ALL".

At least one pair of output arrays is required; all the other entries are optional.

**Example:**

```
HB: FREQ=5GHZ Input Power: from -12dBm to 6dBm step=3dBm
    MVout PVout MVinput PVinput
    { Wavform Spectrum=(MVout, PVout)."Vout_T"
      Tmin=0 Tmax=0.4ns NT=80
      Input_Power=all };
```

where MVout and PVout are assumed to be the magnitudes and phases of the output voltage spectrum. The View is designed to display the time-domain output voltage waveforms at different input power levels, as illustrated in Fig. 10.11.



Waveform Views are available only for HB (harmonic balance) sweep sets, since the waveforms are derived from spectrum responses.

**Specify Spectrum Arrays for Waveform Views**

You must specify each of the frequency-domain spectrum responses to be transformed into time-domain waveforms by a pair of arrays representing the magnitudes and phases of the spectrum. The arrays must have the dimension of [0:N\_Spectra] and the array elements represent the spectrum components at different spectral frequencies.

Typically, you can select from the built-in circuit response arrays available from harmonic balance simulation (Chapter 6). The arrays can also be user-defined from postprocessing the built-in responses, as long as they can be meaningfully interpreted as the magnitudes and phases of frequency-domain spectra.

You must specify at least one pair of spectrum arrays per View (there is no default). Multiple pairs of arrays must be separated by an ampersand "&".



The spectrum arrays must be defined as output labels in the sweep set, i.e., arrays that are not included as outputs in the sweep set cannot be included in the Views either.

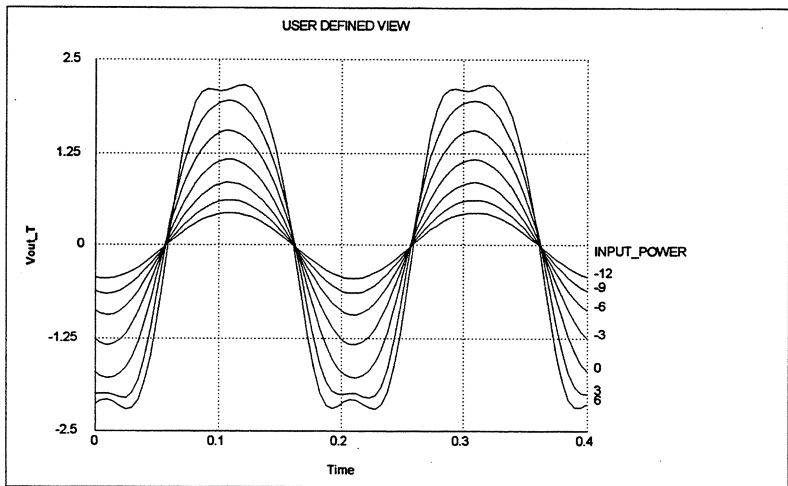


Fig. 10.11 Output voltage waveforms at different input power levels.

## Waveform Names, Colors and Shapes

For each pair of spectrum arrays, you can specify a name to identify the corresponding waveform in the graphical display. In the above example, the waveform corresponding to the spectrum arrays (MVout, PVout) is named as "Vout\_T".

You can also specify the colors and/or shapes for the waveform display. The available colors and shapes are listed in Tables 10.2 and 10.3, respectively, in Section 10.2.

## Time Interval and Number of Time Points

The time interval in which the waveforms are to be calculated and displayed can be specified using the keywords TMIN and TMAX. The default interval is [0 1].

In the foregoing example, the time interval is chosen as [0 0.4ns], which is two periods since the fundamental frequency is 5GHZ.

The keyword NT can be used to specify the number of time points to be sampled within the time interval. The default number of time points is 100.

## Select Values for the Sweep Labels

If the sweep set contains one or two sweep labels, you can select a value for each sweep label from those available. For one of the sweep labels you can choose to display all the available values using the keyword "ALL". In the preceding example, all the available values for the sweep label `Input_Power` are included in the display.



If there are two sweep labels, you cannot choose "ALL" for both sweep labels. By default, sweep labels are assigned the first values in their respective sweep ranges.

## Waveform Display Scales, Titles, Comments and Legend

### Syntax:

```
{ WAVEFORM SPECTRUM=(mag_array1, ang_array1).name1.color1.shape1 &
                    (mag_array2, ang_array2).name2.color2.shape2 &
                    (mag_arrayn, ang_arrayn).namen.colorn.shapen
  TMIN=tmin  TMAX=tmax  NT=nt
  sweep_label1=value1  sweep_label2=value2
  YMIN=ymin  YMAX=ymax  NXTICKS=nxticks  NYTICKS=nyticks
  TITLE=view_title  Y_TITLE=y_title
  COMMENTS=comment1 & comment2 & ... commentm
  LEGEND=legend_pos }
```

*mag\_arrayi* and *ang\_arrayi*,  $i = 1, 2, \dots, n$ , represent  $n$  pairs of output arrays specified in the sweep set, each pair represents the magnitudes and phases of a frequency-domain spectrum response, and all the arrays must have the same dimension of [0:N\_SPECTRA].

*name1*, *name2*, ..., *namen* are character strings identifying the time-domain responses, *color1*, ..., *colorn*, *shape1*, ..., *shapen* specify the associated display attributes.

*tmin* and *tmax* are constants defining the time interval (default: *tmin* = 0, *tmax* = 1), *nt* is a positive integer specifying the number of time points (default *nt* = 100).

*sweep\_label1* and *sweep\_label2* represent the sweep labels in the sweep set (if defined), *value1* and *value2* are selected values for *sweep\_label1* and *sweep\_label2*, respectively, *value1* or *value2* (but not both) can be the keyword "ALL".

*ymin* and *ymax* are constants defining the Y-axis scale,

*nxticks* and *nyticks* are integers specifying the number of ticks (grids) on the axes.

*view\_title*, *y\_title*, *comment1*, ..., *commentm* are character strings.

*legend\_pos* can be ON, OFF, TOP or INSIDE to control the display of legends.

At least one pair of output arrays is required; all the other entries are optional.

You can define display scales ("zoom"), titles and comments for waveform Views in the same way as described in Section 10.2.

The exception is that the keywords XMIN, XMAX and X\_TITLE do not apply to waveform Views, because the X-axis scale is already defined by TMIN and TMAX, and the title for the X-axis is predefined and fixed as "Time".

Example:

```

HB:  FREQ=5GHZ  Input_Power: from -12dBm to 6dBm step=3dBm
      MVinout PVinput MVout PVout
      { Waveform Spectrum=(MVinput, PVinput)."Vinout_T".diamond &
        (MVout, PVout)."Vout_T".X
        Tmin=0 Tmax=0.4ns NT=80 Input_power=6dBm
        Title="INPUT AND OUTPUT VOLTAGE WAVEFORMS"
        Y Title="Voltage Waveforms"
        Ymin=-2.5 Ymax=2.5 NYTicks=4 NXTicks=4 };

```

This View displays both the input and output voltage waveforms at a single input power level, and illustrates the definitions of shapes, titles and scales, as shown in Fig. 10.12.

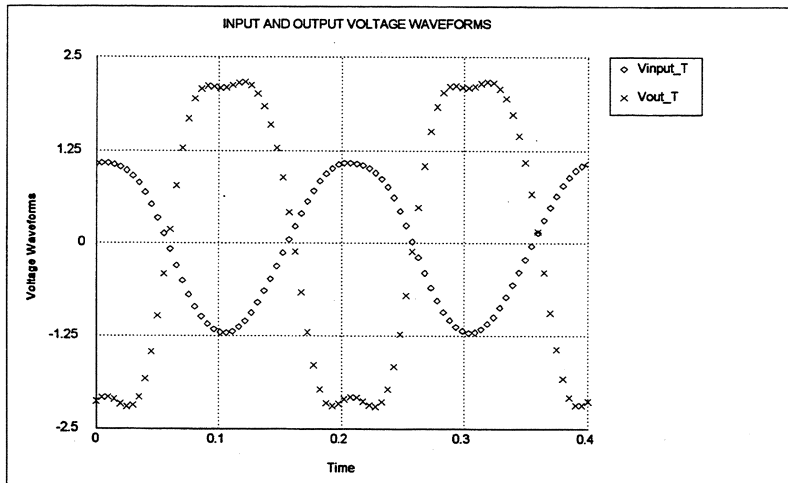


Fig. 10.12 Waveform View of input and output voltages.

## Select Views for Display

Waveform Views are displayed through the “Waveform” menu option under the “Display” menu.

## 10.7 Smith Chart Views

Smith Chart Views are designed to display complex responses, such as  $S$  parameters, on a Smith Chart.

### Syntax:

```
{ SMITH MP=(mag_label1, ang_label1).name1.color1.shape1 &
      (mag_label2, ang_label2).name2.color2.shape2 &
      (mag_labeln, ang_labeln).namen.colorn.shapen
  X=sweep_label other_sweep_label=value
  TITLE=view_title
  COMMENTS=comment1 & comment2 & ... commentm }
```

*mag\_labeli* and *ang\_labeli*,  $i = 1, 2, \dots, n$ , represent  $n$  pairs of output labels specified in the sweep set, each pair represents the magnitude and phase of a complex response.

*name1*, *name2*, ..., *namen* are character strings to identify the complex responses,

*color1*, ..., *colorn*, *shape1*, ..., *shapen* specify the associated display attributes.

*sweep\_label* must be one of the sweep labels in the sweep set.

If the sweep set contains two sweep labels, *other\_sweep\_label* is assigned *value* which can be either one of the available values or the keyword "ALL".

*view\_title*, *comment1*, *comment2*, ..., *commentm* are strings.

At least one pair of output labels is required; all the other entries are optional.

### Example:

```
AC: FREQ: from 2 to 18 step=1
MS11, PS11, MS11_data, PS11_data
MS22, PS22, MS22_data, PS22_data
{ SMITH X=FREQ
  MP=(MS11,PS11)."simulated S11" &
     (MS11_data,PS11_data)."S11 data".circle &
     (MS22,PS22)."simulated S22" &
     (MS22_data,PS22_data)."S22 data".triangle
  title="MATCH BETWEEN SIMULATED AND MEASURED S11 and S22"
  comments = "model FETM" & "(c) OSA Inc." }
```

where MS11, PS11, MS22 and PS22 are built-in responses representing the magnitudes and phases of  $S$  parameters, and MS11\_data, PS11\_data, MS22\_data and PS22\_data represent measured data. This View is designed to display the match between the simulated and measured  $S$  parameters on a Smith Chart, as illustrated in Fig. 10.13.

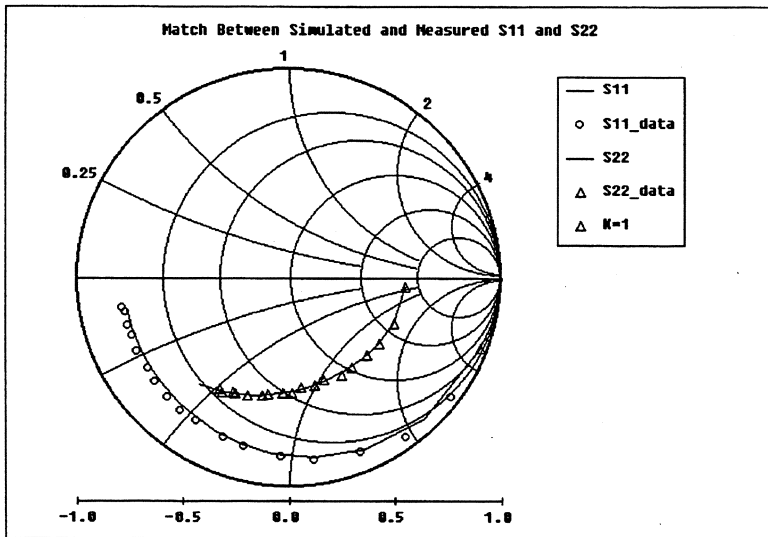


Fig. 10.13 Smith Chart View of  $S$  parameters.

### Specify Complex Responses for Smith Chart Views

The complex responses to be displayed on the Smith Chart can be specified by pairs of output labels representing the magnitudes and phases (i.e., in the polar form).

The complex responses can also be given in the rectangular form:

#### Syntax

```
{ SMITH RI=(real_label1, imag_label1).name1.color1.shape1 &
  (real_label2, imag_label2).name2.color2.shape2 &
  (real_labeln, imag_labeln).namen.colorn.shapen
  X=sweep_label other_sweep_label=value
  TITLE=view_title
  COMMENTS=comment1 & comment2 & ... commentm }
```

*real\_labeli* and *imag\_labeli*,  $i = 1, 2, \dots, n$ , represent  $n$  pairs of output labels in the sweep set, each pair represents the real and imaginary parts of a complex response.

All other entries are identical to those defined for the polar form.



All the responses in one View must be given in the same form, i.e., you cannot mix the polar and rectangular forms in the same View.

You must specify at least one complex response (i.e., at least one pair of output labels) per View. Multiple responses must be separated by an ampersand "&".

The output labels must be defined in the sweep set (outside the Views) before they can be included in any Views associated with that sweep set.

In addition to the built-in circuit responses, you can also use Smith Chart Views to display postprocessed responses and any other complex functions, as long as they are properly represented in the polar or rectangular form by output labels.

### Response Names, Colors and Shapes

For each pair of output labels, you can specify a name to identify the corresponding complex response in the Smith Chart display. For the example, in Fig. 10.13, the response corresponding to the pair of output labels (MS11, PS11) is named as "simulated S11".

You can also specify the colors and/or shapes for the display, as also illustrated in Fig. 10.13. The available colors and shapes are listed in Tables 10.2 and 10.3, respectively.

### Specify the Sweep Labels for Smith Chart

The complex responses displayed on the Smith Chart must be functions of a sweep label (e.g., the frequency label FREQ). Therefore, the sweep set must contain at least one sweep label.

If the sweep set contains two sweep labels, either one can be selected as the underlying sweep parameter for the Smith Chart. By default, the inner sweep label is chosen. For the other sweep label, you can specify one of the available values or use the keyword "ALL" to include all the available values. By default, the first available value is used.

### Smith Chart Titles and Comments

You can specify a title for the Smith Chart as well as comments, as illustrated in Fig. 10.13.

The Smith Chart is displayed in a fixed scale. No user-definable scaling is available.

### Display Smith Chart Views

Smith Chart Views are displayed through the "SmithChart" menu option under the "Display" menu.

## 10.8 Polar Plot Views

Polar plot Views are designed to display complex responses, such as  $S$  parameters.

### Syntax

```
{ POLAR MP=(mag_label1, ang_label1).name1.color1.shape1 &
      (mag_label2, ang_label2).name2.color2.shape2 &
      (mag_labeln, ang_labeln).namen.colorn.shapen
  X=sweep_label other_sweep_label=value
  YMAX=ymax NYTICKS=nyticks TITLE=view_title
  COMMENTS=comment1 & comment2 & ... commentm }
```

*mag\_labeli* and *ang\_labeli*,  $i = 1, 2, \dots, n$ , represent  $n$  pairs of output labels specified in the sweep set, each pair represents the magnitude and phase of a complex response.

*name1, name2, \dots, namen* are character strings identifying the complex responses, *color1, \dots, colorn, shape1, \dots, shapen* specify the associated display attributes.

*sweep\_label* must be one of the sweep labels in the sweep set.

If the sweep set contains two sweep labels, *other\_sweep\_label* is assigned *value* which can be either one of the available values or the keyword "ALL".

*yymax* is a positive constant defining the radius of the polar plot.

*nyticks* is a positive integer specifying the number of concentric circles to be plotted.

*view\_title, comment1, comment2, \dots, commentm* are strings.

At least one pair of output labels is required; all the other entries are optional.

### Example:

```
AC: FREQ: from 2 to 18 step=1
MS21, PS21, MS21_data, PS21_data
{ POLAR X=FREQ
  MP=(MS21,PS21)."simulated S21" &
    (MS21_data,PS21_data)."S21 data".circle
  Ymax=4.5 NYticks=3
  title="MATCH BETWEEN SIMULATED AND MEASURED S21"
  comments = "model FETM" & "(c) OSA Inc." }
```

where MS21 and PS21 are built-in responses representing the magnitudes and phases of  $S_{21}$ , and MS21\_data and PS21\_data represent measured data. This View is designed to display the match between the simulated and measured  $S_{21}$  on a polar plot, as shown in Fig. 10.14.

### Display Polar Plots

Polar plot Views are displayed through the "SmithChart" menu option under the "Display" menu.

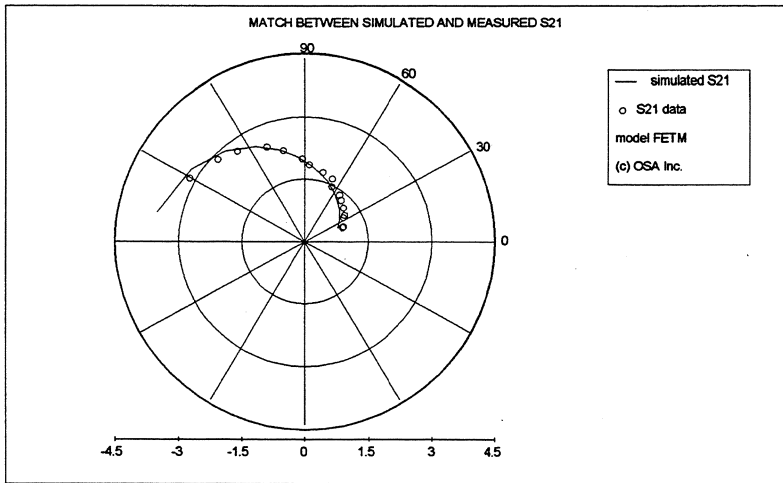


Fig. 10.14 Polar plot View of  $S_{21}$ .

## Specify Complex Responses for Polar Plot Views

The complex responses to be displayed on the polar plot can be specified by pairs of output labels representing the magnitudes and phases (i.e., in the polar form).

The complex responses can also be given in the rectangular form:

### Syntax:

```
{ POLAR RI=(real_label1, imag_label1).name1.color1.shape1 &
      (real_label2, imag_label2).name2.color2.shape2 &
      (real_labeln, imag_labeln).namen.colorn.shapen
  X=sweep_label other_sweep_label=value
  YMAX=yymax NYTICKS=nyticks TITLE=view_title
  COMMENTS=comment1 & comment2 & ... commentm }
```

*real\_label<sub>i</sub>* and *imag\_label<sub>i</sub>*,  $i = 1, 2, \dots, n$ , represent  $n$  pairs of output labels in the sweep set, each pair represents the real and imaginary parts of a complex response.

All other entries are identical to those defined for the polar form.



All the responses in one View must be given in the same form, i.e., you cannot mix the polar and rectangular forms in the same View.



You must specify at least one complex response (i.e., at least one pair of output labels) per View. Multiple responses must be separated by an ampersand "&".

The output labels must be defined in the sweep set (outside the Views) before they can be included in any Views associated with that sweep set.

In addition to the built-in circuit responses, you can also use polar plot Views to display postprocessed responses and any other complex functions, as long as they are properly represented in the polar or rectangular form by output labels.

## Response Names, Colors and Shapes

For each pair of output labels, you can specify a name to identify the corresponding complex response in the polar plot display. For example, in Fig. 10.14, the response corresponding to the pair of output labels (MS21, PS21) is named as "simulated S21".

You can also specify the colors and/or shapes for the display, as also illustrated in Fig. 10.14. The available colors and shapes are listed in Tables 10.2 and 10.3, respectively.

## Specify the Sweep Labels for Polar Plot

The complex responses displayed on the polar plot must be functions of a sweep label (e.g., the frequency label FREQ). Therefore, the sweep set must contain at least one sweep label.

If the sweep set contains two sweep labels, either one can be selected as the underlying sweep parameter for the polar plot. By default, the inner sweep label is chosen. For the other sweep label, you can either specify one of the available values or use the keyword "ALL" to include all the available values. By default, the first available value is used.

## Polar Plot Radius and Number of Circles

You can define the scale of the polar plot by specifying the radius using the keyword YMAX. This defines the radius of the largest (outmost) circle of the polar plot.

You can also specify the total number of concentric circles in the plot (including the outmost one) using the keyword NYTICKS, as illustrated in Fig. 10.14.

## Polar Plot Titles and Comments

You can specify a title for the polar plot as well as comments, as illustrated in Fig. 10.14.

## 10.9 Visualization Views

Visual Views are designed for 3D visualization displays (see also Chapter 9).

### Syntax:

```
{ VISUAL  Z=output_label.color
  ROTATION=rotation  ELEVATION=elevation
  ZMIN=zmin          ZMAX=zmax
  XRATIO=xratio      YRATIO=yratio        ZRATIO=zratio
  X_TITLE=x_title    Y_TITLE=y_title      Z_TITLE=z_title
  NXTICKS=nxticks    NYTICKS=nyticks      NZTICKS=nzticks
  LINES=on_or_off    FRAME=on_or_off      PAINT=on_or_off
  SMOOTH=on_or_off   CONTOUR=on_or_off    AXIS=on_or_off }
```

*output\_label* must be one of the output labels in the sweep set.

*color* specifies the color for displaying wire frames.

*rotation* and *elevation* are numerical constants defining the viewing angles in degrees; their values must be between 0 and 90.

*zmin* and *zmax* are numerical constants defining the limits for the Z-axis.

*xratio*, *yratio* and *zratio* are numerical constants defining the amplification ratios; their values must be between 0 and 3.

*x\_title*, *y\_title* and *z\_title* are character strings.

*nxticks*, *nyticks* and *nzticks* are integers specifying the number of ticks on axes.

*on\_or\_off* must be either ON or OFF, specifying the setting of a switch.

*output\_label* is required; all the other entries are optional.

### Example:

```
Sweep
  X1: from 0.5 to 5.5 step=0.1
  X2: from -7 to 7 step=1  F1, F2, F3
  {Visual Z=F2 Rotation=30 Zmax=20};
  ...
End
```

### Visual View Requires Two Sweep Parameters

Visual Views can be defined only in sweep sets which include two sweep parameters. On the visualization display, the first (outer) sweep parameter is represented by the X-axis, and the second (inner) sweep parameter is the Y-axis.

### Output Label

Each Visual View must include one and only one output label. This label is a function of the two sweep parameters and represented by the Z-axis on the 3D display.

## Display Color

You can specify a color associated with the output label, for example:

```
{Visual Z=F2.yellow ... }
```

You can specify any color defined in the OSA90 color map. The specified color will be used for drawing grid lines in the 3D display.

## Viewing Angles

Visual Views can include rotation and elevation angles for 3D display.

Example:

```
{Visual ... Rotation=30 Elevation=60 ... }
```

The functions of these angles are explained in Chapter 9. The default angles are 45°.

## View Limits on Function Values

You can define the minimum and maximum limits for the Z-axis. Function values outside these limits will be truncated.

Example:

```
{Visual ... Zmin=0 Zmax=15 ... }
```

By default, all the function values are displayed.

## Amplification Ratios

You can specify the amplification ratios for the X-, Y- and Z-axes.

Example:

```
{Visual ... Xratio=1.2 Yratio=1.5 Zratio=0.8 ... }
```

The default amplification ratio is 1 for all axes.

## X-, Y- and Z-Axis Titles

By default, the X- and Y-axes are labelled by the respective sweep parameter names. The Z-axis is labelled by the output label name.

Example:

```
Sweep
  FREQ: from 2GHz to 10GHz step=0.5GHz
  PIN: from -9dBm to 12dBm step=1dBm   POUT, POWER_GAIN
  {Visual Z=POWER_GAIN ... };
...
End
```

By default, the X-, Y- and Z-axes are labelled FREQ, PIN and POWER\_GAIN, respectively.

You can specify different titles for the axes in the Visual View definition.

Example:

```
{Visual ... X_title="Frequency in GHz" ... }
```

## Number of Ticks on Axis

You can specify the number of ticks to be displayed for the each of the axes.

Example:

```
{Visual ... NXticks=10 NYticks=5 NZticks=6 ... }
```

The default number of ticks is 4.

## Settings of Visualization Options

The keywords LINES, FRAME, PAINT, SMOOTH and AXIS allow you to specify the settings of the corresponding options to either ON or OFF.

The meaning of these options and their default settings are defined in Chapter 9.

Example:

```
{Visual ... LINES=ON PAINT=OFF ... }
```

This Visual View produces a single-color wire-frame drawing.

## Contours for a Specific Set of Function Values

You can specify a set of values for contour plotting following the keyword CONTOUR.

Example:

```
{Visual ... CONTOUR=(0.5 1 2 5 10 20) ...}
```

This Visual View produces contours for the function (output label) values 0.5, 1, 2, 5, 10 and 20.

You can specify a maximum of 19 contour values for each Visual View.

Alternatively, you can simply switch on the contour option:

```
{Visual ... CONTOUR=ON ... }
```

By default, the contours correspond to 19 values uniformly spaced between the minimum and maximum function values (or between Zmin and Zmax if they are specified).



# 11

## Optimization

<b>11.1</b>	<b>Overview</b>	11-1
<b>11.2</b>	<b>Specification Block</b>	11-2
<b>11.3</b>	<b>Simulation Types</b>	11-5
<b>11.4</b>	<b>Parameter Sweeps</b>	11-9
<b>11.5</b>	<b>Parameter Labels</b>	11-15
<b>11.6</b>	<b>Output Labels and Goals</b>	11-16
<b>11.7</b>	<b>Error Functions and Weights</b>	11-19
<b>11.8</b>	<b>FUN Datapipe</b>	11-21
<b>11.9</b>	<b>FDF Datapipe</b>	11-24
<b>11.10</b>	<b>Objective Functions</b>	11-27
<b>11.11</b>	<b>OSA90 Optimize Menu</b>	11-32
<b>11.12</b>	<b>Trace of Optimization Variables</b>	11-45
<b>11.13</b>	<b>Sensitivity Analysis</b>	11-46
<b>11.14</b>	<b>Space Mapping Optimization</b>	11-51
<b>11.15</b>	<b>Technical References</b>	11-58





# 11

## Optimization

### 11.1 Overview

One of the outstanding advantages of OSA90 is its optimization capabilities. OSA90 offers a large variety of powerful, state-of-the-art optimizers with proven track records, including

- ▷ gradient-based minimax optimizer
- ▷ gradient-based  $\ell_1$  optimizer
- ▷ gradient-based  $\ell_2$  optimizer
- ▷ gradient-based quasi-Newton optimizer
- ▷ conjugate-gradient optimizer
- ▷ gradient-based Huber optimizer
- ▷ gradient-based one-sided  $\ell_1$  optimizer
- ▷ non-gradient simplex optimizer
- ▷ random optimizer
- ▷ simulated annealing optimizer
- ▷ robust gradient-based yield optimizer

Using these powerful tools, you can optimize circuit responses (DC, small-signal AC and large-signal), postprocessed responses, mathematical expressions, and functions calculated by external simulators connected via Datapipe.

For gradient-based optimization, the circuit sensitivities are computed accurately and efficiently using the FAST technique. Using Datapipe, OSA90 can even take advantage of gradients calculated by external programs.

### What Is Covered in This Chapter

For optimization, you need to define a set of optimizable variables (see Chapter 4), formulate a set of specifications in the Specification block, and choose an appropriate optimizer from the "Optimize" menu.

This chapter begins with a description of the Specification block, followed by a discussion on objective functions which also provides some hints on how to choose between the different optimizers from the "Optimize" menu.

Another useful feature related to optimization is the parameter sensitivity display through the "Sensitivity Analysis" option under the "Optimize" menu. It compares graphically the influence of different variables on the objective function, and thereby helps you select the most influential variables for optimization.

This chapter addresses nominal optimization. Another area of optimization, namely statistical (yield) optimization is covered in Chapter 13.

## 11.2 Specification Block

Specifications express the objectives you wish to achieve through optimization. You select a set of outputs of interest and define the desired goals for these outputs. Here, "output" is a generic term representing circuit responses, Datapipe outputs and other functions.

### Syntax

#### Specification

```

specification_set;
specification_set;
...

specification_set;
End

```

where each *specification\_set* is a statement delimited by a semicolon ";".

### Syntax of specification sets:

```

type: sweep_label_1: sweep_range
      sweep_label_2: sweep_range
      label=expression label=expression ... label=expression
      output_label ...
      output_label = goal ...
      output_label < goal ...
      output_label > goal ...;

```

where *sweep\_label\_1* and *sweep\_label\_2* will sweep through multiple values defined by the corresponding *sweep\_range*, each *label* will be assigned a value given by the corresponding *expression*, and *output\_label* and *goal* represent specifications.

### Example:

#### Specification

```

AC: FREQ: from 2GHZ to 18GHZ step=1GHZ
    MS21 = MS21_data, PS21 = PS21_data;

HB: Signal Power: -10dBm -5dBm 0dBm
    FREQ: from 5GHZ to 12GHZ step=1GHZ
    Omega=(2.0 * PI * FREQ)
    Bias_Voltage=0.75V
    Pout_dBm[2] < -30, Efficiency > 0.75;

End

```

## Simulation Type

The simulation type is defined by a keyword representing the type of circuit analysis (DC, small-signal AC or harmonic balance) involved in the calculation of the output labels. The simulation type can also indicate that the functions are calculated by an external program connected to OSA90 via Datapipe. You can omit the simulation type if the specification set does not involve circuit analysis or Datapipe of the FUN or FDF protocol.

## Sweep Labels

Sweep labels represent parameter sweeps (i.e., simulation loops). A sweep label loops through a set of (multiple) values defined in the sweep range. One or two independent sweep labels can be defined for each specification set.

## Parameter Labels

A parameter label is assigned a single value or expression. For instance, in the example shown at the beginning of this section, there are two parameter labels, namely `Omega` and `Bias_Voltage`. `Bias_Voltage` is assigned a single value and `Omega` is to be evaluated from an expression.

## Output Labels

Output labels represent the functions and responses to be optimized. In the example, the output labels are `MS21`, `PS21`, `Pout_dBm[2]` and `Efficiency`.

## Single, Upper and Lower Specifications

Specifications can be classified as single, upper and lower specifications.

A single specification is of the form

$$\text{output\_label} = \text{goal}$$

In other words, you wish the output label to match the goal as closely as possible. For example, in data-fitting and modeling problems, you wish to optimize the calculated responses to match a given set of measured data.

An upper specification is defined as

$$\text{output\_label} < \text{goal}$$

or

$$\text{output\_label} \leq \text{goal}$$

A lower specification is defined as

$$\text{output\_label} > \text{goal}$$

or

$$\text{output\_label} \geq \text{goal}$$

Upper and lower specifications are illustrated in Fig. 11.1.

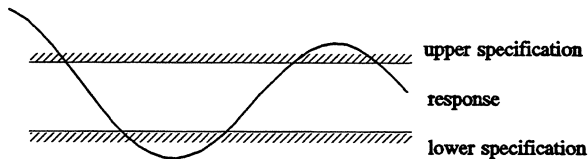


Fig. 11.1 Upper and lower specifications.

## Output Labels with Goals Omitted

If *goal* is omitted from a specification, then *output\_label* represents a function which you wish to minimize directly. This is discussed more precisely in the context of error functions in Section 11.7.



There are many similarities between the Specification and Sweep blocks. Therefore, you may find many of the details in this chapter already described in Chapter 9.

## 11.3 Simulation Types

The simulation types of specification sets are indicated by the keywords listed in Table 11.1.

TABLE 11.1 SIMULATION TYPES

Keyword	Type of Simulation
DC	DC circuit analysis
AC	small-signal AC circuit analysis
HB	large-signal harmonic balance circuit analysis
Datapipe	external simulation via FUN or FDF Datapipe

The simulation type keyword, followed by a colon, must be the first entry in a specification set:

```
DC: ..... ;
AC: ..... ;
HB: ..... ;
Datapipe: ..... ;
```

A simulation type keyword is not needed if the specification set does not involve circuit analysis or Datapipe of the FUN or FDF protocol.

### Built-In Circuit Responses

The simulation type determines which set of built-in circuit responses will be calculated. For instance, if the keyword DC is specified for a specification set, then DC responses will be calculated for that specification set.

☞ DC, small-signal and large-signal circuit responses are described in Chapter 6.

The output labels of a specification set can involve only one type of circuit response. For example, if the simulation type of a specification set is DC, then the output labels of that set may include DC responses and functions of DC responses, but not small-signal AC or harmonic balance responses.

## FUN and FDF Datapipes

The simulation type **Datapipe** is used for specification sets which contain error functions calculated by an external program and directly supplied for optimization. The external program is connected to OSA90 via Datapipe using the FUN protocol (function only) or the FDF protocol (functions and derivatives).

☞ The details are described in Sections 11.8 and 11.9.

## Generic Numerical Simulation

If a specification set does not require circuit simulation or FUN/FDF Datapipe, then you should omit the simulation type keyword. In this case, all the output labels must be defined by generic expressions. In other words, the output labels cannot be circuit responses or functions of circuit responses.

## Frequency Labels

For AC and HB simulation types, you must specify the frequency values by means of the label **FREQ**.

For AC, **FREQ** specifies the frequency for the small-signal circuit analysis. For HB, **FREQ** specifies the fundamental frequency of the AC sources. During DC analysis, the value of **FREQ** is automatically set to zero.

You can define a sweep range for the label **FREQ** (☞ Section 11.4):

```
FREQ: from 2GHZ to 16GHZ step=1GHZ
```

or, you can assign it a single value, such as

```
FREQ = 7GHZ
```

## Two-Tone Frequencies

In the case of two-tone excitations, **FREQ** specifies the first tone frequency, and the second tone frequency is specified by a keyword **FREQ2**.

Example:

```
HB: FREQ=900MHZ FREQ2=907MHZ ...;
```

where the tone frequencies are specified as 900MHZ and 907MHZ.

The keyword `FREQ2` cannot be directly defined as a sweep label. However, you can still sweep the second tone frequency, by tying `FREQ2` to another label:

```
Second_Tone_Freq: from 1GHZ to 5GHZ step=1GHZ
FREQ2 = Second_Tone_Freq
```

The following example ties the two tone frequencies together:

```
FREQ: from 1GHZ to 15GHZ step=2GHZ
FREQ2 = (FREQ + IM_Freq)
```

where `IM_Freq` is a user-defined label representing the difference between the two tone frequencies.

☞ See Chapter 6 for a discussion on the generation of intermodulation frequencies in two-tone simulation.

## The HARM Keyword

In the `CIRCUIT` statement in the Model block, you can specify the highest harmonic to be included in harmonic balance simulation (☞ see Chapter 6 for further details).

### Syntax

```
CIRCUIT HARM=m ...;
```

*m* is the highest harmonic to be included in harmonic balance simulation,  $m_0 \leq m \leq 16$ , where *m*<sub>0</sub> is the highest harmonic among all the sources.

Example:

```
Model
...
CIRCUIT ... HARM=5;
...
End
```

This becomes the "global" default which applies to all specification sets of the HB type.

You can also specify the highest harmonic to be included in harmonic balance simulation for individual specification sets of the HB type using the keyword HARM.

**Syntax**

```
HB: HARM=m1 ...;
```

*m1* specifies the highest harmonic for the specification set,  $m0 \leq m1 \leq m$ , where *m0* is the highest harmonic among all the sources and *m* is the global default.

Example:

```
Specification
  HB: HARM=4 ...;
...
End
```

Specification sets for which the HARM keyword is not explicitly specified will assume the global default.

## The RREF Keyword

For small-signal AC specification sets (i.e., the simulation type is AC), the *S* parameters are, by default, calculated with respect to the actual port terminations (impedances) as specified in the PORT definitions in the Model block of the input file (see Chapter 6). You can override this default by specifying the reference impedance using the keyword RREF.

**Syntax**

```
AC: RREF=x ...;
```

where *x* specifies the *S*-parameter reference impedance for the specification set, and *x* must be a positive constant value or constant label.

Example:

```
Sweep
  AC: RREF=60 ...;
...
End
```

This will work much the same way as though the impedance of 60 ohm (purely resistive) were specified for each port as its terminating impedance. The difference is that this definition takes effect only for the specification set in which it is specified.



## 11.4 Parameter Sweeps

Parameter sweeps are defined by means of sweep labels and sweep ranges.

### Syntax:

```
sweep_label: x1, x2, ... xn
```

where *sweep\_label* is an existing label and *x1*, *x2*, ..., *xn* are numerical values.

### Example:

```
Width: 1.1, 1.2, 1.7, 3.5, 5
```

where the label *Width* must have already been defined.



The sweep label must be followed by a colon ":". You cannot use an equal sign "=" instead of the colon. Colon is chosen to designate sweep labels, whereas the equal sign is reserved for assignment of a single value. For example,

```
Width = 1.1, 1.2, 1.7, 3.5, 5
```

would be mistaken as an assignment of a single value, namely 1.1, to *Width*.

### Sweep an Interval with Uniform Step Size

### Syntax:

```
sweep_label: from x1 to x2 step=x3
```

where *sweep\_label* is an existing label and *x1*, *x2* and *x3* are numerical values. This translates into a set of values as: *x1*, *x1+x3*, *x1+2×x3*, ..., *x2*.

### Example:

```
Width: from 1.5 to 5 step=0.5
```

This translates into a set of values for *Width* as 1.5, 2, 2.5, ..., 5.



The upper limit of the interval, namely *x2*, is always included in the translated values, even if the step size does not divide the interval evenly. For example,

```
Width: from 1.5 to 2.75 step=0.5
```

The translated values are 1.5, 2, 2.5, 2.75, which include the upper limit 2.75.

You can specify a negative step size, such as

Width: from 5 to 1.5 step=-0.5

The translated values are 5, 4.5, 4, ..., 1.5. In this case, the lower limit must be greater than the upper limit, i.e.,  $x1 > x2$ .

The number of subintervals is  $n = (x2 - x1) / x3$ , if  $x3$  divides  $(x2 - x1)$  evenly, or else  $n = (x2 - x1) / x3 + 1$ , (the number of values is  $n + 1$ ). The limit is  $n \leq 1023$ .

## Specify the Number of Steps

### Syntax:

*sweep\_label*: from *x1* to *x2* N=*n*

where *sweep\_label* is an existing label, *x1* and *x2* are numerical values, *n* is an integer,  $1 \leq n \leq 1023$ . The step size is  $(x2 - x1) / n$ . The number of values is  $n + 1$ .

Example:

Width: from 1 to 5 N=8

This translates into a step size of  $(5 - 1) / 8 = 0.5$ , and a set of values as 1, 1.5, 2, ..., 5. Note that the total number of values is 9.

## Sweep an Interval with Exponential Step Size

### Syntax:

*sweep\_label*: from *x1* to *x2* ratio=*x3*

where *sweep\_label* is an existing label and *x1*, *x2* and *x3* are numerical values. This translates into a set of values as:  $x1, x1 \cdot x3, x1 \cdot x3^2, \dots, x2$ .

Example:

Width: from 3 to 96 ratio=2

This translates into a set of values for Width as 3, 6, 12, 24, 48, 96.



The upper limit of the interval, namely  $x_2$ , is always included in the translated values, even if it does not coincide with the prescribed ratio. For example,

Width: from 3 to 30 ratio=2

The translated values are 3, 6, 12, 24, 30.

The values  $x_1$ ,  $x_2$  and  $x_3$  must be consistent, i.e.,

$x_3 > 1$  if  $x_2 > x_1$

$x_3 < 1$  if  $x_2 < x_1$

### Specify the Number of Exponential Steps

#### Syntax

```
sweep_label: from x1 to x2 NEXP=n
```

where *sweep\_label* is an existing label,  $x_1$  and  $x_2$  are numerical values,  $n$  is an integer,  $1 \leq n \leq 1023$ . The step ratio is  $(x_2 / x_1)^{1/n}$ . The number of values is  $n + 1$ .

Example:

Width: from 3 to 96 NEXP=5

This is equivalent to specifying a step ratio of  $(96 / 3)^{1/5} = 2$ . The values for sweeping Width are 3, 6, 12, 24, 48, 96. Note that the total number of values is 6.

### Combine Sweep Intervals and Points

For a sweep label, you can specify an arbitrary combination of uniformly spaced sweep intervals, exponentially spaced sweep intervals and discrete points. For example,

```
Width: 0.75 0.8 from 1 to 5 step=1
       from 7 to 15 step=2 19.99
```

which translates into the following set of values:

```
0.75 0.8 1 2 3 4 5 7 9 11 13 15 19.99
```



The set of values assigned to a sweep label will not be sorted by the file parser. During simulation, the values are assigned in the exact order as they appear in the input file. If the values need to be processed in a particular order (ascending, descending, etc.), they must be entered in the desired order.

## What Can Be Sweep Labels

You can use the predefined label `FREQ` as a sweep label:

```
FREQ: from 4GHZ to 20GHZ step=2GHZ
```

You can also sweep any labels which are defined in the Expression and Model blocks to represent numerical constants.

For example, suppose the following labels are defined in the Expression (or Model) block:

```
Width: 3;
Area = Width * Length;
```

You can use `Width` as a sweep label, because it represents a constant. But `Area` cannot be a sweep label because it represents a formula.



Labels representing optimization variables can be used as sweep labels in the Sweep block but not in the Specification block. This is because during optimization the optimizer must have exclusive control over the values of the variables.

Once a parameter sweep is finished during simulation, the value of the sweep label is reset to the original value as defined in the Expression or Model block.

## Macro Constants and Expressions

Macro constants, labels and expressions can be used in the definition of sweep ranges (values). For example,

```
#define MIN_X 1.0
#define X_RANGE 8.0
```

```
X_Step = 2;
...
```

```
X: from MIN_X to (MIN_X + X_RANGE) step=X_Step
```

where `MIN_X` and `X_RANGE` are macro constants, `X_Step` is a label, and the upper limit of the sweep interval is defined by an expression.

Only constant labels can be used. Expressions, if used, must be enclosed in a pair of parentheses to avoid ambiguity.

☞ See Chapter 3 for further details on text macros.

## Two-Dimensional Sweep

OSA90 permits up to two independent sweep labels per specification set. If a specification set contains two sweep labels, the first sweep label is called the **outer sweep label**, and the second one is called the **inner sweep label**.

Example:

```
Width: from 2 to 8 step=2
Length: 3, 4, 5
```

Width is the outer sweep label and Length is the inner sweep label.

The two-dimensional sweep defined in the above example translates into pairs of values for the pair of labels (Width, Length) in the following (row-wise) order:

```
(2,3), (2,4), (2,5)
(4,3), (4,4), (4,5)
(6,3), (6,4), (6,5)
(8,3), (8,4), (8,5)
```

It is equivalent to a double loop in the "C" programming language as

```
for (Width = 2; Width <= 8; Width += 2) {
    for (Length = 3; Length <= 5; Length += 1) {
        ...
    }
}
```

## The Order of Sweeps in Circuit Simulation

The order of the two sweep labels can be significant in circuit simulation. For instance, if the following two-dimensional sweep for small-signal AC simulation is involved in optimization

```
Bias_Voltage: from 1V to 3V step=0.5V
FREQ: from 6GHZ to 18GHZ step=2GHZ
```

it is desirable to have the bias voltage as the outer sweep label and the frequency as the inner sweep label (as shown). Because, the nonlinear devices, if used, need to be linearized for small-signal analysis by solving the DC nonlinear equations. A new solution of the DC nonlinear equations is needed only when the bias voltage changes, but not when the frequency changes. Therefore, having the bias voltage as the outer sweep label minimizes the number of nonlinear DC solutions required for the specification set.

If a large-signal harmonic balance simulation involves a power sweep (i.e., a sweep of the input power of an AC source), it is preferable to have the power sweep as the inner sweep, because once the nonlinear harmonic balance equations are solved at one point, it is relatively easy to obtain a new solution after an incremental change in the power level. It is usually more time-consuming to obtain a new solution after changing other parameters.

## Sweeping Array Indices

One very useful application of parameter sweeps is to sweep indices of arrays. For example,

```

Model
  A[10] = [a1 a2 ... a10];
  B[10] = [b1 b2 ... b10];

  K: 1;

  C = A[K] + B[K] ...;
  ...
end

Specification
  K: from 1 to 10 step=1 ...;
  ...
end

```

where the sweep label *K* is used as an index of reference to the arrays *A* and *B* in the expression by which *C* is defined.

☞ See Chapter 4 for further details on arrays.

An example of sweeping the indices of matrices:

```

Model
  MS_Data[3,3] = [ ... ];
  ....
  CIRCUIT ...;

  I: 1;
  J: 1;
  ...
end

Specification
  AC:  I: from 1 to 3 step=1
       J: from 1 to 3 step=1
       MS[I,J] = MS_data[I,J] ...;
end

```

where the matrix *MS* contains the calculated small-signal *S* parameter magnitudes (built-in response) and *MS\_Data* may represent measured data.

## 11.5 Parameter Labels

In addition to sweep labels, you can also define parameter labels for a specification set. Parameter labels are not to be swept, rather, they are assigned a single value or to be evaluated from an expression.

### Syntax

$$\text{parameter\_label} = x$$

where *parameter\_label* is a label and *x* can be a value, a label or an expression.

Examples:

$$\text{Drain\_Bias} = 4V$$

$$\text{Omega} = (2 * \text{PI} * \text{FREQ})$$

where *Drain\_Bias* is a parameter label which is assigned a constant value, and *Omega* is a parameter label to be evaluated from an expression.

Up to 64 parameter labels can be defined in each specification set.

Similar to sweep labels, parameter labels must be defined in the Model or Expression block and represent numerical constants.

### Parameter Labels as Functions of Sweep Labels

Parameter labels can be functions of the sweep labels defined in the same specification set.

Example:

$$\text{FREQ: from 1GHZ to 10GHZ step=1GHZ}$$

$$\text{Omega} = (2 * \text{PI} * \text{FREQ})$$

The formula (expression) must be enclosed within parentheses to avoid ambiguity.

Using this concept, you can create synchronous sweeps, such as

$$\text{K: from 0 to 20 step=1}$$

$$\text{XK} = (\text{XO} + \text{K} * \text{DX})$$

$$\text{YK} = (\text{YO} + \text{K} * \text{DY})$$

$$\text{ZK} = (\text{ZO} + \text{K} * \text{DZ})$$

where a number of parameter labels are tied together through a sweep label. In other words, this creates a one-dimensional sweep of several parameters.

## 11.6 Output Labels and Goals

Output labels are required components of a specification set. They represent the responses and functions that you wish to optimize.

### Single, Upper and Lower Specifications

Single, upper and lower specifications can be defined by

$$\text{output\_label} = \text{goal}$$

$$\text{output\_label} < \text{goal}$$

$$\text{output\_label} > \text{goal}$$

respectively (see also Section 11.2).

The operator " $\leq$ " can be used as an equivalent of "<" for upper specifications, and " $\geq$ " as an equivalent of ">" for lower specifications.

Output labels may include any of the labels defined in the Model and Expression blocks: constants, variables, circuit responses, postprocessed functions, outputs from external programs via Datapipe, etc.

Example:

```
Specification
DC: ... Idrain_DC = Idrain_data;

AC: ... MS11 < 0.1  MS21_dB > 7  MS21_dB < 9;

HB: ... MVoutput[2] < 0.05,  Conversion_Gain > 2.5;

... AA = 1,  BB < 2,  CC > 3;
end
```

where the first specification set includes `Idrain_DC` (DC circuit responses) as an output label with a single specification, the output labels of the second specification set include `MS11` (small-signal  $S$  parameter) and `MS21_dB` (postprocessed response), and the output labels of the third specification set include `MVoutput[2]` (large-signal voltage spectrum at the second harmonic) and `Conversion_Gain` (postprocessed response). The last specification set does not have an explicitly specified simulation type, which indicates that the output labels `AA`, `BB` and `CC` are defined by generic expressions which do not require circuit simulation.



If any circuit responses are included in a specification set, they must be consistent with the simulation type (DC, AC, HB). See Chapter 6 for the built-in response labels corresponding to each simulation type.



## Goals Can Be Constants or Labels

You can specify the goals by constants or labels.

Example:

```

Model
  Gain_slope = 2.0 + 0.1 * FREQ;
  ...
End

Specification
  AC: FREQ: from 2 to 20 step=1 MS21 - Gain_slope;
End

```

## Arrays as Output Labels and Goals

Arrays can be used as output labels and goals.

For example, suppose that A1[10] is an array and a specification is defined as

```
A1 > Goal
```

If Goal is a scalar, it is equivalent to creating a specification for each array element with the same goal:

```

A1[1] > Goal
A1[2] > Goal
...
A1[10] > Goal

```

If the Goal is also an array, it must have the same dimension as A1. In this case, OSA90 interprets the specification as

```

A1[1] > Goal[1]
A1[2] > Goal[2]
...
A1[10] > Goal[10]

```

## Functions and Goals Defined by Expressions

The functions and goals can be defined by expressions. The expressions must be enclosed within a pair of parentheses.

Example:

```
AC:  FREQ: from 6 to 14 step=0.5 (20 * log10(MS21)) > 8;
```

Another example:

```
DC:  ...  Igate_DC = (Ig_data_mA / 1000);
```

## Specifications with Goals Omitted

If a specification is defined without an explicit goal, then it is assumed that you wish to have the output label minimized directly.

Example:

```
HB:  ...  MVoutput[2];
```

The response MVoutput[2] is to be minimized as much as possible.

In terms of the error function, this is equivalent to an upper specification with a goal of zero:

```
HB:  ...  MVoutput[2] < 0;
```

## Maximum Number of Output Labels

The maximum number of output labels per specification set is limited to 512. An output array is equivalent to as many output labels as the number of its elements. If more is needed, you can divide and spread the output labels among a number of specification sets of the same simulation type.

## 11.7 Error Functions and Weights

Error functions are created to measure the deviation of the responses (output labels) from the goals prescribed according to the specifications, as shown in Table 11.2.

**TABLE 11.2 ERROR FUNCTIONS**

Specification	Syntax	Error Function
single specification	$output\_label = goal$	$F =  output\_label - goal $
upper specification	$output\_label < goal$	$F = output\_label - goal$
lower specification	$output\_label > goal$	$F = goal - output\_label$
goal omitted	$output\_label$	$F = output\_label$

### Weighting Factors

An optimization problem often requires a number of specifications to be considered simultaneously. In such cases, weights (weighting factors) can be useful.

Following each specification, you can specify an optional weight, as shown in Table 11.3.

**TABLE 11.3 WEIGHTS AND WEIGHTED ERROR FUNCTIONS**

Syntax	Error Function
$output\_label = goal \quad W=weight$	$F = weight \times  output\_label - goal $
$output\_label < goal \quad W=weight$	$F = weight \times (output\_label - goal)$
$output\_label > goal \quad W=weight$	$F = weight \times (goal - output\_label)$
$output\_label \quad W=weight$	$F = weight \times output\_label$

*weight* must be a positive constant.

Example:

```
Specification
DC: ... Idrain_DC = Idrain_data W=50;
AC: ... MS21_dB > 7 MS21_dB < 9 MS11 < 0.1 W=10;
end
```

where the specification on Idrain\_DC is assigned a weight of 50 and the specification on MS11 is assigned a weight of 10.

Weights can be used to balance the scales (orders of magnitude) of the error functions. Different responses, goals and consequently error functions may have significantly different scales (orders of magnitude). This can cause the optimization to be dominated by the error functions which have relatively larger magnitudes. Properly introduced weights can set the error functions onto a more uniform, comparable scale.

On the other hand, weights can be used to purposely change the scale of an individual error function to emphasize or de-emphasize its importance relative to the other error functions.

## The Sign of Error Functions

Error functions are formulated in such a way that a positive value of  $F$  would indicate that the specification is violated, and a zero or negative value of  $F$  would indicate that the specification is satisfied. In other words, error functions are expected to be minimized.

## The Number of Error Functions

The number of error functions produced by each specification set is equal to the number of specifications in that set times the number of sweep points in that set. The total number of error functions for an input file is the sum of the numbers of error functions from all the specification sets defined in the Specification block.

For example,

```
Specification
HB: Input_Power: from -15dBm to 10dBm step=5dBm
    MVoutput[2] < 0.05 Conversion_Gain > 2.5;
AC: FREQ: from 2GHZ to 15GHZ step=1GHZ
    MS11 < 0.1 W=10 MS21_DB > 7 MS21_DB < 9;
End
```

The first set has 2 specifications, 6 sweep points and hence 12 error functions. The second set has 3 specifications, 14 sweep points and hence 42 error functions. The total number of error functions is 54.

## 11.8 FUN Datapipe

In addition to the error functions derived from built-in responses, error functions can also be calculated by external programs, sent back to OSA90 via Datapipe, and directly used in optimization.

The FUN Datapipe protocol is specifically designed for this purpose. Unlike the SIM Datapipe protocol, the FUN protocol does not label the individual outputs from the child program. The outputs from the child program are taken as error functions which need not and cannot be further processed by expressions.



Chapter 5 contains general descriptions on Datapipe. Please read Chapter 5 first, if you have not already done so.

### Define FUN Datapipe in the Model or Expression Block

#### Syntax

```
Datapipe: FUN FILE="filename"
          N_INPUT=n   INPUT=(x1, ..., xn)
          N_OUTPUT=m  NAME=errf_name
          TIMEOUT=k;
```

where *filename* is the name of the child program, *n* is the number of inputs to the child, *m* is the number of outputs from the child, *errf\_name* is a collective identifier for all the outputs and *k* is the number of seconds for time-out.

#### Example:

Expression

...

```
Datapipe: FUN FILE="sim_errf"
          N_INPUT=4   INPUT=(X1, X2, 2.0, FREQ)
          N_OUTPUT=6  NAME=error_set1;
```

...

end

This indicates that the child program "sim\_errf" accepts 4 inputs from OSA90 and returns 6 outputs. The outputs are not represented by individual labels, instead they are collectively identified by the name error\_set1.

## Reference to FUN Datapipe in the Specification Block

The outputs from the FUN Datapipe can be included in the Specification block by creating a specification set with the simulation type of Datapipe.

### Syntax:

```
Datapipe: sweep_label_1: sweep_range
          sweep_label_2: sweep_range
          label=expression label=expression ... label=expression
          errf_name W=weight;
```

where *sweep\_label\_1* and *sweep\_label\_2* are sweep labels and each *label* represents a parameter label defined in the same way as for all other specification sets. *errf\_name* must identify a FUN Datapipe and *weight* is an optional weighting factor which, if given, will be applied to all the error functions from the FUN Datapipe.

Example:

```
Specification
  Datapipe: FREQ: from 5 to 20 step=2
            error_set1 W=10;
  ...
end
```

where *error\_set1* identifies the set of 6 error functions to be calculated by the child program "sim\_errf", as shown in the FUN Datapipe definition. The weighting factor defined as 10 is applied to all 6 error functions.

## FUN Datapipe with Parameter Sweeps

If one or two sweep labels are defined for a Datapipe specification set, then the child program is called at each sweep point. Thus, the total number of error functions produced by that specification set is  $m$  times the number of sweep points, where  $m$  is the number of outputs from the FUN Datapipe.



If you have more than one FUN Datapipe, each must be contained in a separate specification set.

## Comparison with SIM Datapipe

The FUN Datapipe protocol differs from the SIM protocol only in the way OSA90 treats the outputs from the child program. From the child's point of view there is no difference, i.e., the child programs written for the FUN and the SIM protocols are identical.

Consider this example:

```

Expression
  Datapipe: FUN FILE="sim_errf"
            N_INPUT=4  INPUT=(X[4])
            N_OUTPUT=6 NAME=error_set1;

  Datapipe: SIM FILE="sim_errf" ...
            N_INPUT=4  INPUT=(X[4])
            N_OUTPUT=6 OUTPUT=(F[6]);
end

Specification
  Datapipe: error_set1;

  F;
end

```

where the same child program is referenced in two Datapipe definitions, one using the FUN protocol and the other using the SIM protocol. The outputs from the two Datapipes are used in two separate specification sets, and the error functions produced by the two sets are identical (which is redundant and shown here just to illustrate the concept).

Using the FUN protocol, the Datapipe outputs are directly used as error functions. Because the outputs do not need to be individually labelled, the FUN protocol requires less computer time and memory for processing. This can be a significant benefit if the number of outputs is large (the maximum is 4096).

On the other hand, the SIM protocol has the advantage that you can selectively reference and postprocess individual Datapipe outputs. In the example, F[1], F[2], ... F[6] can be processed individually if you wish. Furthermore, you can display the outputs from the SIM Datapipe by including them in the Sweep block, which is not possible for the FUN Datapipe.

## 11.9 FDF Datapipe

The FDF protocol, like the FUN protocol, is specifically designed for optimization: the outputs from the child program are not labelled but directly taken as error functions.

The difference is that the FDF protocol allows the child program to supply not only the error functions but also their first-order derivatives with respect to the inputs.



Chapter 5 contains general descriptions on Datapipe. Please read Chapter 5 first, if you have not already done so.

### Define FDF Datapipe in the Model or Expression Block

#### Syntax

```
Datapipe:  FDF  FILE="filename"
           N_INPUT=n   INPUT=(x1, ..., xn)
           N_OUTPUT=m  NAME=errf_name
           TIMEOUT=k;
```

where *filename* is the name of the child program, *n* is the number of inputs to the child, *m* is the number of outputs from the child, *errf\_name* is a collective identifier for all the outputs and *k* is the number of seconds for time-out.

#### Example:

```
Expression
...
```

```
Datapipe:  FDF  FILE="sim_fdf"
           N_INPUT=4   INPUT=(X1, X2, 2.0, FREQ)
           N_OUTPUT=6  NAME=error_fdf;
```

```
...
end
```

This indicates that the child program "sim\_fdf" will calculate and return to OSA90 6 error functions (outputs) and their first-order derivatives with respect to the 4 inputs, namely X1, X2, 2.0 and FREQ. The outputs are not represented by individual labels, instead they are collectively identified by the name error\_fdf.



## Gradients Calculated by the Child Program

Via the FDF Datapipe, the child program provides OSA90 with first-order derivatives of the outputs with respect to the inputs. This can be a significant advantage if the child program is capable of calculating the derivatives efficiently and accurately.

Not all the Datapipe inputs are necessarily optimization variables. The inputs can be labels representing functions of variables, outputs from other Datapipes, or even constants. Any necessary conversion from the derivatives supplied by the child to the appropriate gradients required by the optimizer is done automatically by OSA90.



The child program must return the derivatives with respect to *all* the inputs. If you are sure that a particular input will *never* be defined as an optimization variable nor be dependent on any optimization variable, then you can set the corresponding derivatives to zero in the child program.

The total number of derivatives is  $n \times m$ , where  $n$  is the number of inputs and  $m$  is the number of outputs. The derivatives must be sent in the following (row-wise) order:

$$\begin{aligned} &\partial f_1/\partial x_1, \partial f_2/\partial x_1, \dots, \partial f_m/\partial x_1, \\ &\partial f_1/\partial x_2, \partial f_2/\partial x_2, \dots, \partial f_m/\partial x_2, \\ &\dots \\ &\partial f_1/\partial x_n, \partial f_2/\partial x_n, \dots, \partial f_m/\partial x_n \end{aligned}$$

where  $f_j$  represents the  $j$ th output and  $x_i$  represents the  $i$ th input.

## Reference to FDF Datapipe in the Specification Block

The outputs from the FDF Datapipe can be included in the Specification block by creating a specification set with the simulation type of Datapipe.

### Syntax

```
Datapipe: sweep_label_1: sweep_range
          sweep_label_2: sweep_range
          label=expression label=expression ... label=expression
          errf_name W=weight;
```

where *sweep\_label\_1* and *sweep\_label\_2* are sweep labels and each *label* represents a parameter label defined in the same way as for all other specification sets. *errf\_name* must identify an FDF Datapipe and *weight* is an optional weighting factor which, if given, will be applied to all the error functions from the FDF Datapipe.

Example:

```
Specification
  Datapipe: FREQ: from 5 to 20 step=2
            error_fdf W=10;
  . . .
end
```

where `error_fdf` identifies the set of 6 error functions to be calculated by the child program "sim\_fdf", as shown in the FDF Datapipe definition. The weighting factor defined as 10 is applied to all 6 error functions.

### FDF Datapipe with Parameter Sweeps

If one or two sweep labels are defined for a Datapipe specification set, then the child program is called at each sweep point. Thus the total number of error functions produced by that specification set is  $m$  times the number of sweep points, where  $m$  is the number of outputs from the FDF Datapipe.



If you have more than one FDF Datapipe, each must be contained in a separate specification set.

## 11.10 Objective Functions

Objective function is a scalar criterion by which an optimization algorithm compares one set of variables with another to decide which one is more desirable.

Given a set of error functions, different objective functions can be constructed. Available in OSA90 are the minimax,  $\ell_1$ ,  $\ell_2$ , generalized  $\ell_2$ , Huber, one-sided Huber, one-sided  $\ell_1$  and "sum" objective functions.

In this section, a set of error functions is denoted by

$$F_1, F_2, \dots, F_m$$

where  $F_j$  represents an error function derived from a specification or a FUN/FDF Datapipe output at one of the sweep points, and  $m$  represents the total number of error functions.

### Sum of Functions

The simplest form of objective function is the sum of functions:

$$U = \sum_{j=1}^m F_j$$

An important special case is when there is only one function ( $m = 1$ ), then the objective function is  $U = F_1$ . In other words, the single function is taken directly as the objective function. This can be useful when you wish to formulate your own objective function and directly provide it for optimization.

### Minimax Objective Function

The minimax objective is formulated as

$$U = \max_j \{ F_j \}$$

The minimax objective function represents the worst case among the error functions. Its minimization typically leads to an equal-ripple solution, i.e., at the solution there are several worst-case error functions which are equal in value to the objective function.

The minimax objective function is most suitable for design optimization where all the performance specifications are equally important. A good example is the design of filters where the loss must be uniformly minimized across the passband.

The minimax objective function is the least suitable for optimizing model responses to match measured data, especially when significant residual errors and/or measurement errors are expected. If used in such cases, the optimization can be dominated by a few error functions and easily trapped in local minima (false solutions).

## L1 Objective Function

The  $\ell_1$  objective function is formulated as

$$U = \sum_{j=1}^m |F_j|$$

The  $\ell_1$  objective function is most suitable for data-fitting and modeling, especially in the presence of measurement errors and uncertainties. It has been shown to have the unique ability to bypass isolated large errors and concentrate on error functions which are closer to zero (see, for example, references [1] and [3] at the end of this chapter).

Since the  $\ell_1$  objective function is a sum of absolute values, its minimization means that the individual error functions are driven towards a zero value. This is perfect for single specifications. But, if  $F_j$  is derived from an upper or lower specification, then a negative value of  $F_j$  would indicate that the specification is satisfied and therefore it should not be included in the  $\ell_1$  objective function.

If you have upper and lower specifications and choose the  $\ell_1$  objective function, then OSA90 will automatically truncate any negative errors (from upper and lower specifications) to a zero value.

## L2 Objective Function

The  $\ell_2$  objective function is the sum of squares:

$$U = \sum_{j=1}^m F_j^2$$

Another conventional definition uses the square root of the sum of squares, which is equivalent to the above formulation in the sense that both have the same minimizers (solutions).

The  $\ell_2$  objective function is widely used for its smooth convergence properties. One of the classical applications is to obtain unbiased estimates in approximation problems. However, in circuit modeling, the  $\ell_2$  objective function can be more susceptible to measurement errors than  $\ell_1$ .

Since the  $\ell_2$  objective function is a sum of squares, its minimization means that the individual error functions are driven towards a zero value. This is perfect for single specifications. But, if  $F_j$  is derived from an upper or lower specification, then a negative value of  $F_j$  would indicate that the specification is satisfied and therefore it should not be included in the  $\ell_2$  objective function.

If you have upper and lower specifications, the generalized  $\ell_2$  objective function is more suitable. If you choose the  $\ell_2$  objective function, OSA90 will automatically truncate any negative error function values (from upper and lower specifications) to a zero value.

## Generalized L2 Objective Function

First, consider an index set of positive error functions defined as

$$J = \{ j \mid F_j \geq 0 \}$$

The generalized  $\ell_2$  objective function is defined as

$$U = \begin{cases} \sum_{j \in J} F_j^2 & \text{if } J \neq \emptyset \\ -1 & \text{otherwise} \\ \sum_{j=1}^m \frac{1}{F_j^2} & \end{cases}$$

In situations where you have single specifications only, the generalized  $\ell_2$  objective function is identical to the  $\ell_2$  objective function, because the error functions will never be negative, i.e., the condition  $J \neq \emptyset$  will always be true.

When upper and lower specifications are involved, the generalized  $\ell_2$  objective function has a significant advantage over the  $\ell_2$  objective function, because the error function values are not truncated when they become negative. The generalized  $\ell_2$  objective function allows the optimization to continue in a meaningful way even after all the specifications are satisfied.

The generalized  $\ell_2$  objective function is suitable for design optimization problems where an equal-ripple solution is not necessary. For example, if you wish to design an amplifier with smooth in-band gain characteristics and you are willing to tolerate a higher attenuation towards the band edges, the generalized  $\ell_2$  objective function can be a good choice. However, if the gain characteristics must be uniform throughout the entire band, then you should choose the minimax objective function.

### Huber Objective Function

The Huber objective function is defined as [7,8]

$$U = \sum_{j=1}^m \rho_k(F_j)$$

where

$$\rho_k(F) = \begin{cases} F^2/2 & \text{if } |F| \leq k \\ k|F| - k^2/2 & \text{if } |F| > k \end{cases}$$

where  $k$  is a positive constant threshold value.

The Huber function  $\rho_k$  is a hybrid of the least-squares ( $\ell_2$ ) (when  $|F| \leq k$ ) and the  $\ell_1$  (when  $|F| > k$ ) functions.

The Huber function is robust against errors above the threshold (i.e.,  $|F| > k$ ), since those errors are treated in the  $\ell_1$  sense. The choice of  $k$  defines the threshold between "large" and "small" errors. Varying  $k$  alters the proportion of error functions to be treated in the  $\ell_1$  or  $\ell_2$  sense.

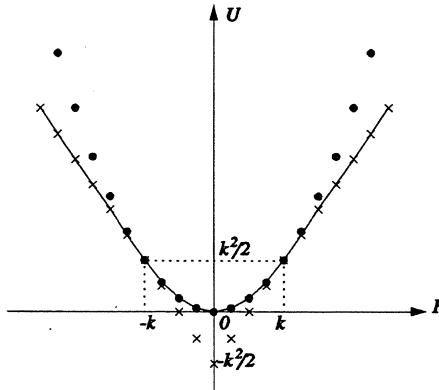


Fig. 11.2 Comparison of the Huber,  $\ell_1$  and  $\ell_2$  objective functions in the one-dimensional case. The continuous curve, the strikes and the dots illustrate the Huber,  $\ell_1$  and  $\ell_2$  objective functions, respectively.

### Huber+ (One-Sided Huber) Objective Function

The Huber+ objective function in OSA90 represents the one-sided Huber objective function defined as [7,8]

$$U = \sum_{j=1}^m \rho_k^+(F_j)$$

where

$$\rho_k^+(F) = \begin{cases} 0 & \text{if } F \leq 0 \\ F^2/2 & \text{if } 0 < F \leq k \\ kF - k^2/2 & \text{if } F > k \end{cases}$$

where  $k$  is a positive constant threshold value.

This one-sided Huber function is tailored for design optimization with upper and/or lower specifications (Section 11.2).  $F$  is truncated when negative because the corresponding design specification is satisfied.

### One-Sided L1 Objective Function

The one-sided  $\ell_1$  objective function is defined as

$$U = \sum_{j=1}^m F_j^+$$

where  $F_j^+ = \max \{ 0, F_j \}$ .

The one-sided  $\ell_1$  objective function is tailored for design optimization with upper and/or lower specifications.  $F_j$  is truncated when negative because the corresponding design specification is satisfied.

## 11.11 OSA90 Optimize Menu

The optimize menu contains a list of menu options as summarized in Table 11.4.

TABLE 11.4 OPTIMIZE MENU OPTIONS

Menu Option	Function
L1	invokes an L1 optimizer
L2	invokes an L2 optimizer
Minimax	invokes a minimax optimizer
Quasi-Newton	invokes a Quasi-Newton optimizer
Conjugate Gradient	invokes a conjugate gradient optimizer
Huber	invokes a Huber optimizer
One-Sided L1	invokes a one-sided L1 optimizer
Random	invokes a random optimizer
Simplex	invokes a simplex optimizer
Simulated Annealing	invokes a simulating annealing optimizer
Yield	invokes the yield optimization feature (covered in Chapter 13)
Sensitivity Analysis	invokes the sensitivity analysis feature (covered in Section 11.13)



## Optimizer and Objective Function

There is no "universal" optimizer which can satisfactorily minimize all the different objective functions. For instance, a specialized algorithm is needed for the minimax objective function. A few key references are listed at the end of this chapter.

The optimizers listed in Table 11.5 include state-of-the-art algorithms specialized in minimizing the minimax,  $\ell_1$ ,  $\ell_2$ , one-sided  $\ell_1$  and Huber objective functions as well as optimizers of a more generic nature.

TABLE 11.5 OPTIMIZERS AND OBJECTIVE FUNCTIONS

Optimizer	Gradient-Based	Objective Function
L1	Yes	L1
L2	Yes	L2
Minimax	Yes	Minimax
Quasi-Newton	Yes	Generalized L2, L2, Sum of Errors
Conjugate Gradient	Yes	Generalized L2, L2, Sum of Errors
Huber	Yes	Huber, One-Sided Huber
One-Sided L1	Yes	One-Sided L1
Random	No	L1, L2, Generalized L2, Minimax, Sum of Errors
Simplex	No	L1, L2, Generalized L2, Minimax, Sum of Errors
Simulated Annealing	No	L1, L2, Generalized L2, Minimax, Sum of Errors

↩ Objective functions are defined in Section 11.10

## Default Optimizer

You can designate a default optimizer in the input file Control block (see also Chapter 3), so that it will be invoked by the “Optimize” toolbar button (see Chapter 2).

```

Syntax:

Control
  Optimizer = optimizer_name;
End

optimizer_name can be:  L1,           L2,           Minimax, Quasi_Newton,
                        Random, Simplex,   Yield,       Conjugate_Gradient,
                        Huber,  One_Sided_L1, Simulated_Annealing
    
```



If you choose a different optimizer manually (by selecting one from the “Optimize” menu) it will become the new default optimizer for the “Optimize” toolbar button until the input file is re-compiled.

## Optimization Options

When any optimizer listed in Table 11.5 is invoked, a dialog box will appear containing the various options for that optimizer. All of the optimization dialog boxes contain a default set of options. Certain optimization dialog boxes contain additional options that are applicable to the particular optimizer chosen. For a list of options and the optimizers to which they are applicable see Table 11.6.

TABLE 11.6 OPTIMIZATION OPTIONS AND RELATED OPTIMIZERS

Option	Applicable Optimizers
Number of iterations	All
Accuracy of solution	All
Show downhill iterations only	All
Objective function	Quasi-Newton, Conjugate Gradient, Huber, Random, Simplex
Threshold	Huber
Temperature	Simulated Annealing
Cooling ratio	Simulated Annealing
Iterations per T	Simulated Annealing

## Number of iterations

This option allows you to limit the maximum number of iterations. The optimizer will stop once this limit is reached even if a solution within the desired accuracy has not been found. If this happens, the best set of values of the variables (in terms of minimizing the objective function) is presented as the solution.

You can also designate your choice in the Control block.

### Syntax:

```
Control
  N_Iterations = n;
End
```

where  $n$  is one of the choices: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 30, 50, 100, 999, 9999.

Other values of  $n$  will be rounded to the closest choice which is greater than  $n$ , e.g., 45 will be rounded to 50. If  $n > 9999$ , it will be set to 9999.

## Accuracy of solution

This option allows you to specify the desired accuracy of the solution. The optimization will stop when the step size separating the current step and the next step is smaller than the specified accuracy. The step size is relative to the norm of the variable vector.

You can also designate your choice in the Control block.

### Syntax:

```
Control
  Accuracy = v;
End
```

where  $v$  is one of the choices: 0.01, 0.001, 1.0e-4, 1.0e-5, 1.0e-6.

Other values of  $v$  will be rounded to the closest choice which is smaller than  $v$ , e.g., 0.003 will be rounded to 0.001. If  $v < 1.0e-6$ , it will be set to 1.0e-6.

## Show downhill iterations only

During optimization, OSA90 displays the iteration count and current value of the objective function on the screen. By default, this information is displayed at every iteration.

For example:

```

Minimax Optimization
Iteration 1/30 Max Error=0.388132
Iteration 2/30 Max Error=0.380194
Iteration 3/30 Max Error=0.363949
Iteration 4/30 Max Error=0.329979
Iteration 5/30 Max Error=0.27877
Iteration 6/30 Max Error=0.243729
Iteration 7/30 Max Error=0.251028
Iteration 8/30 Max Error=0.213164
Iteration 9/30 Max Error=0.20241
Iteration 10/30 Max Error=0.218615
Iteration 11/30 Max Error=0.198647
Iteration 12/30 Max Error=0.198021
Iteration 13/30 Max Error=0.198333
Iteration 14/30 Max Error=0.197433
Iteration 15/30 Max Error=0.197355
Iteration 16/30 Max Error=0.19733
Iteration 17/30 Max Error=0.197292
Iteration 18/30 Max Error=0.197291
Solution Max Error=0.197291

```

Of the iteration count, the first number is the current iteration and the second one is the maximum number of iterations specified. For example, 12/30 means that the current iteration is the 12th out of a maximum of 30.

It is sometimes possible to observe the value of the objective function getting larger than that of a previous iteration. In the example, the 7th iteration produces a higher objective function value than that of the 6th iteration. This is not unusual. It is a part of the optimization algorithm.

You can instruct the program to display only those iterations that lead to a better (smaller) objective function value than the previous ones. You can do so by checking the "Show downhill iterations only" checkbox. You may find this especially desirable for the random optimizer, since it usually requires a large number of random explorations which result in fluctuating objective function values.

Printing the best iterations only can also be set up in the Control block.

### Syntax:

```

Control
  Print_Best_Iterations;
End

```

## Objective Function

Some of the optimizers offered by OSA90 can optimize more than one type of objective function (see Table 11.5). For example, the Huber optimizer can optimize the one- or two-sided Huber objective function. You can indicate the preferred objective function in the Control block.

### Syntax:

```
Control
  Objective_Function = name;
End
```

*name* can be:    L1,        L2,        Minimax,    Generalized\_L2,    Sum\_of\_Errors,  
                  Yield,    GL1,    Huber,        One\_Sided\_Huber,    One\_Sided\_L1

Not all the objective functions are appropriate for every optimizer. In the event that there is a conflict between the objective function and the optimizer chosen, the objective function specified in the Control block will be ignored.

## Threshold

Separates L1 and L2 regions of the Huber function (see Fig. 11.2). You can also specify a value for this option in the Control block.

### Syntax:

```
Control
  Huber_Threshold = v;
End
```

where  $v$  must be a positive value.

## Temperature

The initial temperature (the "Temperature" option) controls the step size of exploratory moves. An initial temperature that is too small may retard the process by not allowing enough exploratory moves over a sufficiently large area of the parameter space. In general, a proper choice of the initial temperature is problem dependent. You can also specify a value for this option in the Control block.

### Syntax:

```
Control
  Initial_Temperature = v;
End
```

where  $v$  must be a constant and  $v \geq 0.01$ . The default value is 5.0.

## Cooling Ratio

This is a parameter for the simulated annealing optimizer in OSA90. It controls the ratio for reducing the annealing temperature between iterations. As the temperature declines, exploratory moves are less likely to be accepted and the optimization is more likely to focus on the vicinity of the most promising local optimum.

You can specify a value for the cooling ratio either in the Control block or interactively in the optimization dialog box.

**Syntax:**

```
Control
  Cooling_Ratio = v;
End
```

where  $v$  must be a constant and  $0.01 \leq v \leq 0.999999$ . The default value is 0.85.

## Iterations per T

This is a parameter for the simulated annealing optimizer in OSA90. It controls the number of iterations before temperature reduction.

You can specify a value for this parameter either in the Control block or interactively in the optimization dialog box.

**Syntax:**

```
Control
  Iterations_of_T = n;
End
```

where  $n$  is one of the choices: 5, 10, 20, 40, 70, 100.

Other values of  $n$  will be rounded to the closest choice which is greater than  $n$ , e.g., 30 will be rounded to 40. If  $n > 100$ , it will be set to 100.

## Additional Optimization Options

In addition to the options listed above which can be set either in the applicable dialog box or through Control block statements, there are a number of options which can be set only in the Control block (see Table 11.7).

TABLE 11.7 ADDITIONAL OPTIMIZATION OPTIONS

Option	Applicable Optimizers
Two-Sided Perturbation	All gradient-based optimizers
Perturbation Scale	All gradient-based optimizers
No Default Bounds	All
Disable Adjoint	All gradient-based optimizers

## Two-Sided Perturbation

OSA90 employs a number of advanced techniques to calculate in the most efficient way the derivatives required for gradient-based optimization. However, in certain situations some of the derivatives may have to be estimated by perturbations (finite differences).

By default, one-sided perturbations (forward difference) are used, i.e., the derivative of a function  $f(x)$  w.r.t. a variable  $x$  is estimated by

$$[f(x + \Delta x) - f(x)] / \Delta x$$

In the Control block, you can request two-sided perturbations (full differences):

### Syntax:

```
Control
    Two_Sided_Perturbation;
End
```

Using two-sided perturbations, the derivative of  $f(x)$  w.r.t.  $x$  is estimated by

$$[f(x + \Delta x) - f(x - \Delta x)] / (2 \Delta x)$$

This usually improves the accuracy at the expense of increased computational effort.

## Perturbation Scale

OSA90 employs a number of advanced techniques to calculate in the most efficient way the derivatives required for gradient-based optimization. However, in certain situations some of the derivatives may be estimated by perturbations (finite differences).

For example, the derivative of a function  $f(x)$  with respect to  $x$  can be estimated by

$$[f(x + \Delta x) - f(x)] / \Delta x$$

You can specify the relative step size for  $\Delta x$  in the Control block.

### Syntax:

```
Control
  Perturbation_Scale = v
End
```

where  $v$  must be a constant and  $1.0e-6 \leq v \leq 0.5$ . The default value is  $5.0e-6$ .

A related control option is `Two_Sided_Perturbation`.

## No Default Bound

In OSA90, you can assign explicit bounds on optimization variables (see Chapter 4). On the other hand, if an optimization variable is defined without explicit bounds, then it is implicitly constrained within a set of default bounds.

The default bounds depend on the starting value (initial value) of the optimization variable. If the starting value is positive or zero, the default bounds are  $(0, \infty)$ , otherwise the default bounds are  $(-\infty, 0)$ .

You can disable the assignment of default bounds in the Control block. This will allow variables defined without explicit bounds to vary freely from  $-\infty$  to  $\infty$ .

### Syntax:

```
Control
  No_Default_Bounds;
End
```



## Disable Adjoint

OSA90 incorporates circuit adjoint sensitivity analysis to compute gradients efficiently for optimization. This feature can be disabled in the Control block.

### Syntax

```
Control
  Disable_Adjoint;
End
```

This can be useful in studying the impact of gradient accuracy on optimization. Also, it may alter the sequence of simulation during optimization. For example, the adjoint analysis in OSA90 may require parameter perturbations at each frequency of circuit simulation, i.e., the parameter perturbation loop is within the frequency sweep loop (or power sweep, etc.). When the adjoint analysis is disabled, the parameter perturbation loop, if required, is outside of the frequency (power, etc.) sweep loop.

## Simulation During Optimization

Simulation is involved at each iteration of optimization in order to calculate the error functions. The calculations are organized in the following sequence:

- 1 Start from the first specification set in the Specification block. After the first specification set is done, go on to the second set, and so on. The simulation is complete when all the specification sets are processed in the natural order.
- 2 The outer sweep loop. Set the outer sweep label to its next available value. After all the values have been assigned, go to step 1 (i.e., go to the next specification set).
- 3 The inner sweep loop. Set the inner sweep label to its next available value. After all the values have been assigned, go to step 2.
- 4 Set the parameter labels. Each parameter label is either specified with a single value or evaluated from an expression (which may depend on the sweep labels).
- 5 Perform circuit analysis if required. Calculate the output labels.
- 6 Calculate the error functions from the output labels and goals according to the specifications. Go to step 3.

If the optimizer is gradient-based, then sensitivity calculations are incorporated into the simulation as appropriate.

## Interrupt the Optimization Process

During optimization, a progress bar is displayed at the bottom of the screen (on top of the status bar).

While the optimization is in progress, you can interrupt it at any time by clicking on the “Stop” button, located on the toolbar. After clicking on the button optimization is stopped immediately.

If you interrupt and terminate the optimization in progress, then the best set of values of the optimization variables achieved up to that point will be presented as the solution.

## Termination of Optimization

Optimization is terminated under one of the following conditions:

- ▶ Interrupted and terminated by the user.
- ▶ A local minimum of the objective function is found, i.e., the value of the objective function cannot be further reduced in the vicinity of the solution.
- ▶ The specified accuracy is reached, i.e., the step size separating the current step and the next step suggested by the algorithm is smaller than the specified accuracy. The step size is relative to the norm of the vector of optimization variables.
- ▶ The maximum number of optimization iterations is reached.
- ▶ A severe error condition is encountered such as a floating-point divide by zero error.

A message indicating the cause of termination will be displayed. Regardless of the cause the solution is always the best set of values of the variables obtained up to the point of termination.

The input file is automatically updated with the solution, i.e., the variables are updated with their optimized values.



The input file that is updated with the solution is actually the copy kept by OSA90 in the computer's memory. The original disk file is not changed. You will be prompted to save or discard the updated file if you try to exit OSA90 or read-in a new file.

## Recording the Optimization Setting in the Input File

Upon completion of optimization, OSA90 automatically inserts a comment line into the input file which reports the date, optimizer, number of iterations and CPU time:

```
! Mon Jul 7 12:05:10 1997. Minimax Optimizer. 15 Iterations. 00:02:10 CPU
```

Since the same input file is also updated with the optimized variables, when you save the file to the disk, the record will be kept together with the corresponding solution.

The inserted line is placed immediately after the first group of comment lines, if any, in the input file.

For example, if the opening lines of the original input file are

```
! test07.ckt
! Project AAA, Team BBB, Designer John Smith Jr.
! FET power amplifier. X-band. Bias and matching circuits to be optimized.

Model
...
```

The updated file after optimization will appear as

```
! test07.ckt
! Project AAA, Team BBB, Designer John Smith Jr.
! FET power amplifier. X-band. Bias and matching circuits to be optimized.
! Mon Jul 7 12:05:10 1997. Minimax Optimizer. 15 Iterations. 00:02:10 CPU

Model
...
```

## Record for Multiple Runs of Optimization

Sometimes a solution is obtained from more than one optimization, i.e., from a starting point you perform optimization, then at the solution you restart optimization, perhaps even with a different optimizer, to obtain a new and presumably better solution.

A separate record is inserted into the input file for each optimization. For example:

```
! Mon Jul 7 14:35:10 1997. L1 Optimizer. 25 Iterations. 00:04:21 CPU
! Mon Jul 7 14:41:26 1997. L2 Optimizer. 11 Iterations. 00:02:03 CPU
! Tue Jul 8 10:01:47 1997. Huber Optimizer. 3 Iterations. 00:00:31 CPU
...
```

## Find a Starting Point for Unfamiliar Problems

Optimization methods are sensitive, to a varying degree, to starting points (i.e., the initial values of the variables).

Generally speaking, the  $\ell_2$  and generalized  $\ell_2$  objective functions are "smoother" and can be easier for the optimization to converge to a reasonable solution from an arbitrary starting point. Therefore, when solving new and unfamiliar problems for which a good starting point is not readily available, you may choose the  $\ell_2$  or generalized  $\ell_2$  objective function in conjunction with the  $\ell_2$ , quasi-Newton or random optimizer to obtain a suitable starting point for subsequent minimax or  $\ell_1$  optimization.

## 11.12 Trace of Optimization Variables

Upon the completion of optimization, the input file is automatically updated with the values of the optimization variables at the solution. In addition, you can use the Trace block to obtain a record of the values of the variables at all iterations.

To do this, you create an empty Trace block in the input file *before* invoking optimization:

```
Trace
end
```

This instructs OSA90 to keep a complete record of the values of the variables and fill in the Trace block with the recorded values after the optimization is completed.

Example:

```
Trace
  10 10 10 10
  9.9005 9.9005 10.1005 10.1005
  9.70446 9.70446 10.3045 10.3045
  9.32394 9.32394 10.7251 9.9005
  8.60708 10.1005 11.6183 9.13931
  7.33447 11.853 12.261 7.78801
  7.09323 16.3232 16.7393 6.08051
  7.76172 24.3495 21.0447 10.1095
  8.00952 25.7783 22.2732 10.5891
  8.42892 28.5187 24.167 11.8682
  8.56213 29.3131 24.7367 12.2294
  8.56288 29.3124 24.7375 12.2285
end
```

The first line in the Trace block is the starting point. In this example, each line corresponds to one iteration. If the number of variables is large, however, the values for each iteration may occupy several lines.



The values on the last line are not necessarily identical to the solution contained in the updated file, because the last trial point may not be the best point.

The data in the Trace block can be used for visualization (see Chapter 9).

## 11.13 Sensitivity Analysis

The "Sensitivity Analysis" option under the "Optimize" menu provides sensitivity displays to assist you in selecting optimization variables. Large-change sensitivities of the  $\ell_1$ ,  $\ell_2$ , minimax, generalized  $\ell_2$  and "sum of errors" objective functions are calculated and displayed.

☞ Objective functions are defined in Section 11.10.

### Large-Change Sensitivities

The term "large-change" refers to changing (perturbing) the values of the optimization variables by a large (significant) amount.

More specifically, all the variables defined in the input file are perturbed by -10%, -5%, 5% and 10% of their nominal values. One variable is perturbed at a time, while the other variables are kept constant at their nominal values. The total number of perturbed points is 4 times the number of variables.

After each perturbation is made, the  $\ell_1$ ,  $\ell_2$ , minimax, generalized  $\ell_2$  and "sum of errors" objective functions are evaluated and the changes with respect to the corresponding objective function values at the nominal point are recorded.

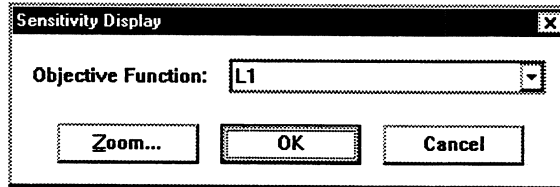
### Simulation for Computing the Sensitivities

When you invoke "Sensitivity Analysis", the program will perform the necessary simulation to compute the sensitivities.

During simulation, you can interrupt it at any time by clicking on the "Stop" button, located on the toolbar. After clicking on the button simulation is stopped immediately.

## Sensitivity Display Options

When the sensitivity analysis is completed, a dialog box appears.



The “Objective Function” option allows you to select one of the five objective functions available: L1, L2, Minimax, Generalized L2 and Sum of Errors.

The “Zoom” option allows you to change the display scale by specifying the parameters “Y-min”, “Y-max”, “X-min” and “X-max” (see Chapter 9 on further details on graphics zoom).

The sensitivities with respect to different variables are displayed as separate curves, as illustrated in Fig. 11.3, where three variables named X1, X2 and X3 are assumed.

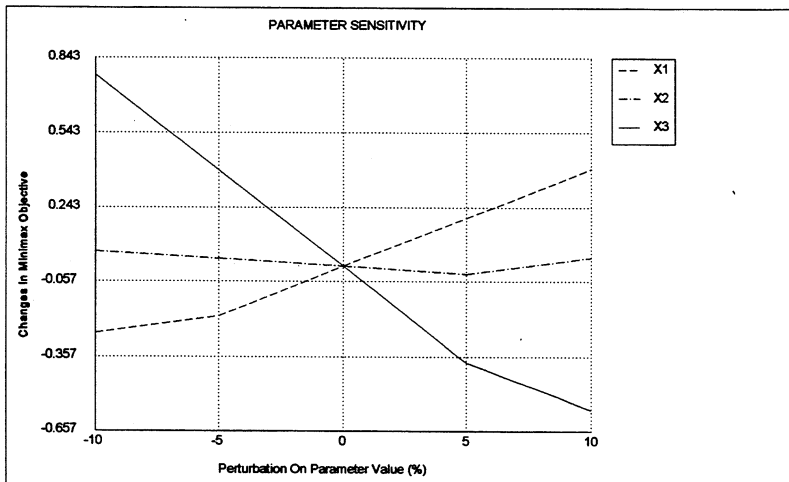


Fig. 11.3 Sensitivity display.

The X-axis of the display is marked by the numbers -10, -5, 0, 5 and 10, representing the percentage perturbations on the variables.

The Y-axis of the display represents the changes in the selected objective function with respect to the nominal (unperturbed) point. A negative value means that the objective function value decreases from the nominal point to the perturbed point. A positive value means an increase.

For example, a point at  $x = 10$  and  $y = -0.2$  would indicate that the value of the selected objective function decreases by 0.2 when the value of the variable increases by 10% from its nominal value.

Fig. 11.3 shows that the value of the minimax objective function would increase when the value of the variable  $X_1$  increases (while  $X_2$  and  $X_3$  remain at their nominal values), and the minimax objective function value would decrease when the value of  $X_3$  increases (while  $X_1$  and  $X_2$  remain at their nominal values).

The number of different variables that can be accommodated in a single display is limited to 6 to keep the display legible. If necessary, the sensitivities will be organized into multiple displays, each showing a subset of the variables.

### Use Sensitivity in Selecting Variables

The sensitivity display reveals not only which direction (increase or decrease) the objective function would change when a variable is perturbed, but also the magnitude of the change. By comparing the sensitivities with respect to different variables, you can estimate the relative influence that each variable has on the objective function value.

This information can be utilized to help determine which parameters should be selected as optimization variables.

First, you define all the parameters which are *potentially optimizable* as variables (by enclosing them within question marks). Then, by examining the sensitivity display you can select only the most influential parameters as variables for optimization.

For example, Fig. 11.3 shows that among the three variables considered,  $X_1$  and  $X_3$  have significantly greater influence on the minimax objective function than  $X_2$ .

If the optimized solution is substantially different from the starting point, you may need to repeat the sensitivity analysis and augment the set of selected variables as necessary.

A proper selection of variables can be important. Having too many variables increases the computation time and may even cause the optimization problem to become ill-conditioned. On the other hand, failing to include some crucial parameters in optimization would lead to solutions that are not the best possible.



## Verification of an Optimized Solution

The sensitivity display can also provide visual verification of an optimized solution. Theoretically, the sensitivity of the objective function should be zero at an optimized solution (a local minimum). Such a sensitivity display resembles a V, with the solution point at the bottom of the valley, as illustrated in Fig. 11.4.

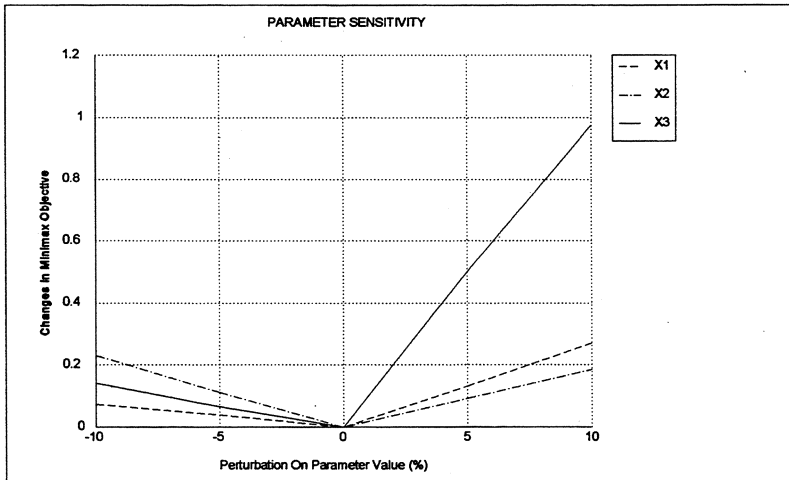


Fig. 11.4 Sensitivity display at an optimized solution.

## The .sen Files

Sensitivity analyses can be time-consuming. To avoid duplicated analyses, a feature is built into OSA90 to save the results of a sensitivity analysis in an encoded file, which can be retrieved and displayed in a subsequent session.

The encoded file is named after the input file by replacing the extension `.ckt` with `.sen`. To ensure that the `.sen` file is uniquely associated with the input file, a `.sen` file will be created only if the input file has the extension `.ckt`.

For example, if you have two input files named `example.ckt` and `example.old`, then the sensitivity analysis results for `example.ckt` will be saved in `example.sen`, but `example.old` will not have a `.sen` file.

Furthermore, to ensure that the information stored is up to date, a "time stamp" is encoded into the `.sen` file which reflects the date and time of the corresponding input file. OSA90 always checks this

time stamp in a `.sen` file before using it. Therefore, any operation which changes the date and time of the input file after a `.sen` file is created will invalidate the stored sensitivity data.

A new `.sen` file is created in three situations:

- ▷ A new input file is read, and analyzed without editing or optimization.
- ▷ An input file is edited or optimized, then saved, and analyzed.
- ▷ An input file is edited or optimized, then analyzed, and saved.

Here "analyzed" refers to sensitivity analysis.

## 11.14 Space Mapping Optimization

The Space Mapping technique establishes a functional relationship between two models: (1) a very accurate but CPU intensive model, and (2) a computationally very efficient but less accurate model. These two models will be referred to as the "fine" and "coarse" models (simulators), respectively.

For example, the fine model may be provided by an electromagnetic simulator and the coarse model by a circuit simulator based on empirical formulas. A circuit simulator model often has a limited validity range for its parameters, beyond which the simulation results become very coarse.

### Theoretical Background

The goal of Space Mapping [10,11] is to direct the bulk of CPU intensive optimization to the coarse model. As a first step, conventional design optimization is carried out using the coarse model. The resulting solution is denoted by  $X_{C_{opt}}$ . Then, a mapping

$$X_c = P(X_f)$$

from the fine model parameter space to the coarse model parameter space needs to be created and, possibly, iteratively refined in order to align the two models.

Once the mapping is established the inverse mapping  $P^{-1}$  (projection) is used to find the fine model solution as the image of  $X_{C_{opt}}$ :

$$P^{-1}(X_{C_{opt}})$$

The current theoretical developments assume that both models have the same number of parameters and the parameters are arranged in the same order. The idea of Space Mapping is to use the optimal responses from the optimized coarse model as a target, and to find the appropriate values for the fine model so that it achieves the target optimal responses as closely as possible.

$P$  is established through an iterative process. The initial mapping  $P^{(0)}$  is found using a preselected set  $X_f\_base$  of  $M$  base points in the fine model parameter space and the set  $X_c\_base$  of corresponding points in the coarse model parameter space. The number  $M$  of these base points for the implementation available in this version of OSA90 needs to be at least  $N + 1$ , where  $N$  is the total number of parameters. The selection of  $X_f\_base$  is fairly arbitrary, though typically one would choose the starting point and  $N$  additional points obtained by perturbing one parameter at a time.

The points in  $X_c\_base$  are determined by  $M$  auxiliary optimizations. For each point (a set of parameter values) in  $X_f\_base$  one needs to find a corresponding point in the coarse model parameter space such that the responses of both models match as closely as possible. This is carried out by a parameter extraction procedure (optimization) using the coarse model to fit the responses of the fine model at the given point in  $X_f\_base$ , and this is repeated for all points in  $X_f\_base$ .



## Mapping Coarse Model Parameters into the Fine Model Space

Given two sets of corresponding base points in both spaces, the function `SMproject` will first establish a mapping between the two spaces and then project a point in the coarse model parameter space (typically an optimized solution) into the fine model parameter space.

### Syntax:

```
Xf[M] = SMproject(Xf_base[M,M], Xc_base[M,M], Xc[M]);
```

where `Xf_base` and `Xc_base` are the model parameters at  $M$  base points.

If `Xc` represents the optimized coarse model parameters, then `Xf` is the projection of `Xc` in the fine model space.

The function `SMproject` is normally used to map the point  $X_{c,opt}$  to obtain the corresponding point for the fine model. When a mapping is fully established the result is the Space Mapping "optimal" solution in terms of the fine model parameters. However, before the mapping is fully established this function is used in the iterative refinement of the mapping by providing the next base point in the fine model space. This new base point will simply augment the existing set of base points.

## Guidelines for Space Mapping Procedure

- 1 Obtain an optimized solution using the coarse model:  $X_{c,opt}$ .
- 2 Let  $Xf = X_{c,opt}$ . Run the fine simulator. Stop if the results are satisfactory already.
- 3 Select  $M$  base points around  $Xf$ :  $Xf\_base[M,N]$  ( $M \geq N + 1$ ).
- 4 For  $i = 1, 2, \dots, M$ , run the fine simulator with  $Xf\_base[i, \cdot]$ . Perform parameter extraction to obtain  $Xc\_base[i, \cdot]$ . The effectiveness of the Space Mapping depends to a large degree on the accuracy of parameter extraction at this step. You must make every effort to achieve the best possible match between the fine and coarse models in terms of the responses of interest (e.g., by alternating the  $\ell_1$ ,  $\ell_2$ , and Huber optimizers and attempting different starting points).
- 5 Project the next trial point  $Xf = SMproject(Xf\_base, Xc\_base, X_{c,opt})$ .
- 6 Run the fine simulator with  $Xf$ . Stop if the results are satisfactory.

## Iterative Refinement of the Mapping

- 7 Perform parameter extraction to obtain  $Xc$  such that the coarse model responses  $Rc(Xc)$  match the fine model responses  $Rf(Xf)$  (using  $X_{c,opt}$  as the starting point for  $Xc$ ).
- 8 Add  $Xc$  and  $Xf$  to the sets of base points (increment  $M$  by 1). Repeat from Step 5.

Example:

```

Model
Xcopt[6] = [ .. .. . ]; ! separately obtained optimal solution

Xf_base[7,6] = [ .. .. . ! first base point  $x_e^{(1)}$ 
                .. .. . ! second base point  $x_e^{(2)}$ 
                .
                .
                .. .. . ]; ! seventh base point  $x_e^{(7)}$ 

Xc_base[7,6] = [ .. .. . ! first base point  $x_c^{(1)}$  corresponding to  $x_e^{(1)}$ 
                .. .. . ! second base point  $x_c^{(2)}$  corresponding to  $x_e^{(2)}$ 
                .
                .
                .. .. . ]; ! seventh base point  $x_c^{(7)}$  corresponding to  $x_e^{(7)}$ 

Xf[6] = SMproject(Xf_base, Xc_base, Xcopt);
.
end

```

The seven base points in *Xf\_base* have been initially and arbitrarily selected. The corresponding base points in *Xc\_base* have been obtained after seven parameter extractions. The point  $X_e^i$  if necessary, will augment *Xf\_base* for the next iteration of refining the mapping.

After five iterations of refining the mapping the corresponding arrays are

```

Model
Xcopt[6] = [ .. .. . ]; ! previously obtained optimal solution

Xf_base[12,6] = [ .. .. . ! first base point  $x_e^{(1)}$ 
                  .. .. . ! second base point  $x_e^{(2)}$ 
                  .
                  .
                  .. .. . ! seventh base point  $x_e^{(7)}$ 
                  .. .. . ! first augmented base point  $x_e^{(8)}$ 
                  .
                  .
                  .. .. . ]; ! fifth augmented base point  $x_e^{(12)}$ 

Xc_base[12,6] = [ .. .. . ! first base point  $x_c^{(1)}$  corresponding to  $x_e^{(1)}$ 
                  .. .. . ! second base point  $x_c^{(2)}$  corresponding to  $x_e^{(2)}$ 
                  .
                  .
                  .. .. . ! seventh base point  $x_c^{(7)}$  corresponding to  $x_e^{(7)}$ 
                  .. .. . ! first augmented base point  $x_c^{(8)}$ 
                  .
                  .
                  .. .. . ]; ! fifth augmented base point  $x_c^{(12)}$ 

Xf[6] = SMproject(Xf_base, Xc_base, Xcopt);
.
end

```

Here, the subsequent base points in *Xf\_base* are obtained, one at a time, by projecting  $X_{c_{opt}}$  into the fine model space using SMproject, and the corresponding base points in *Xc\_base* are obtained by parameter extractions.

## Input Files for Parameter Extractions

The following example illustrates a typical setup for parameter extractions.

Example:

```

Model
Xc[6] = [?...? ...? ...? ...? ...? ...?];
Xf[6] = [ ... .. .. .. .. ];

model1      ! coarse model - a 2-port with optimizable parameter values given in Xc
.
.
PORT 1 0;
PORT 4 0;

model2      ! fine model - a 2-port with parameter values given in Xf
.
.
PORT 5 0;
PORT 6 0;

CIRCUIT;

MS11_DATA = MS33;
PS11_DATA = PS33;
MS21_DATA = MS43;
PS21_DATA = PS43;

MP2RI(MS11_data, PS11_data, RS11_data, IS11_data);
MP2RI(MS21_data, PS21_data, RS21_data, IS21_data);
end

Sweep
AC: FREQ: ....
MS21 MS21_Data PS21 PS21_data
RS11 IS11 RS21 IS21
RS11_DATA IS11_DATA RS21_data IS21_data
{XSweep Y=MS21.yellow & MS21_DATA.yellow.circle}
{XSweep Y=PS21.yellow & PS21_DATA.yellow.circle}
{XSweep Y=RS11.yellow & RS11_DATA.yellow.circle &
RS21.red & RS21_DATA.red.circle &
IS11.green & IS11_DATA.green.circle &
IS21.white & IS21_DATA.white.circle};
end

Spec
AC: FREQ: ....
! MS21 = MS21_data;
! RS11=RS11_DATA IS11=IS11_DATA
RS21=RS21_data IS21=IS21_data;
end

```

The Sweep block is included in the input file for the purpose of visually assessing how good a match has been achieved through the parameter extraction optimization.

## Displaying Space Mapped Responses

The following example illustrates a typical setup for displaying responses of both the fine and the coarse models, aligned through Space Mapping.

Example:

```

Model
  Xcopt[6]      = [ .. .. . ];
  Xf_base[12,6] = [ .. .. .
                  .
                  .. .. . ];
  Xc_base[12,6] = [ .. .. .
                  .
                  .. .. . ];
  Xf[6] = SMproject(Xf_base, Xc_base, Xcopt);

  model1      ! coarse model - a 2-port with parameter values given in Xcopt
  .
  PORT 1 0;
  PORT 4 0;

  model2      ! fine model - a 2-port with parameter values evaluated by SMproject
  .
  PORT 5 0;
  PORT 6 0;

  CIRCUIT;

  MS21_fine = 20 * log10(MS43);
  MS11_fine = 20 * log10(MS33);

  MS21_coarse = 20 * log10(MS21);
  MS11_coarse = 20 * log10(MS11);
end

Sweep
AC: FREQ: from 3.956 to 4.11 step=0.0022
MS21_coarse MS21_fine MS11_coarse MS11_fine
{Xsweep Y=MS21_coarse.red & MS21_fine.red.circle &
MS11_coarse.darkgreen & MS11_fine.darkgreen.circle
Spec=(to 3.967, < -26).green & (from 4.099, < -26).green &
(from 4.008 to 4.058, > -0.5).green
Ymin=-40 Ymax=0 NYticks=4 Y_Title="MS11 and MS21 (dB)"
Title="SM Design of HTS Filter"};
end

```

The original specifications used to obtain  $X_{C_{opt}}$  are included in the Xsweep View for the purpose of visually assessing how good the "optimal" Space Mapping solution is.



## Space Mapping to Produce Parameters of the Coarse Model

Provided with suitable sets of base points, the function `SMmodel` establishes a Space Mapping which can be used to map a point in the fine model parameter space to a point in the coarse model parameter space. (`SMproject` is the inverse of `SMmodel`.)

### Syntax:

```
Xc[M] = SMmodel(Xf_base[M,N], Xc_base[M,N], Xf[M]);
```

where `Xf_base` and `Xc_base` are the model parameters at  $M$  base points. `Xc` represents the coarse model parameters obtained by Space Mapping from the fine model parameters `Xf`.

## Potential Benefits of the `SMmodel` Function

When the Space Mapping is successfully and satisfactorily established it can be used to replace the fine model (which is presumably computationally intensive) by the coarse model (which should be relatively efficient to compute):

$$Rc(Xc = \text{SMmodel}(Xf\_base, Xc\_base, Xf)) \approx Rf(Xf)$$

where  $Rc$  and  $Rf$  represent the coarse and fine model responses, respectively.

Within the region covered by the base points, particularly in the vicinity of the solution,  $Rc$ , evaluated at the parameter values mapped using the `SMmodel` function, should be a good model for  $Rf$ .

In other words, the Space Mapping technique can be used to reduce the computational cost in fine-model-based Monte Carlo simulation and yield optimization.

Compared with polynomial numerical models (such as quadratic models), the Space Mapping technique has an accurate basis and hence should be valid in a much larger region.

## 11.15 Technical References

- [1] J.W. Bandler and S.H. Chen, "Circuit optimization: the state of the art," *IEEE Trans. Microwave Theory Tech.*, vol. 36, 1988, pp. 424-443.
- [2] J.W. Bandler, W. Kellermann and K. Madsen, "A superlinearly convergent minimax algorithm for microwave circuit design," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-33, 1985, pp. 1519-1530.
- [3] J.W. Bandler, W. Kellermann and K. Madsen, "A nonlinear  $\ell_1$  optimization algorithm for design, modeling and diagnosis of networks," *IEEE Trans. Circuits and Systems*, vol. CAS-34, 1987, pp. 174-181.
- [4] R. Fletcher, "A new approach to variable metric algorithms," *Computer Journal*, vol. 13, 1970, pp. 317-322.
- [5] G.R. Gucker, "Stochastic gradient algorithm for searching multidimensional multimodal surfaces," Stanford University Center for Systems Research, Technical Report No. 6778-7, 1969.
- [6] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes*, Cambridge University Press: Cambridge, 1986, pp. 289-293, 301-306.
- [7] H. Ekblom and K. Madsen, "Algorithms for nonlinear Huber estimation," *BIT* 29, 1989, pp. 60-76.
- [8] J.W. Bandler, S.H. Chen, R.M. Biernacki, L. Gao, K. Madsen and H. Yu, "Huber optimization of circuits: a robust approach," *IEEE Trans. Microwave Theory Tech.*, vol. 41, 1993, pp. 2279-2287.
- [9] Goffe, Ferrier and Rogers, "Global optimization of statistical functions with simulated annealing," *Journal of Econometrics*, vol. 60, 1994, pp. 65-100.
- [10] J.W. Bandler, R.M. Biernacki, S.H. Chen, P.A. Grobelny and R.H. Hemmers, "Space mapping technique for electromagnetic optimization," *IEEE Trans. Microwave Theory Tech.*, vol. 42, 1994, pp. 2536-2544.
- [11] J.W. Bandler, R.M. Biernacki, S.H. Chen and P.A. Grobelny, "Optimization technology for nonlinear microwave circuits integrating electromagnetic simulations," *Int. J. Microwave and Millimeter-Wave Computer-Aided Engineering*, vol. 7, January 1997, pp. 6-28.

# 12

## Statistical Analysis

<b>12.1</b>	<b>Overview</b>	12-1
<b>12.2</b>	<b>Statistical Parameters</b>	12-2
<b>12.3</b>	<b>Uniform Distribution</b>	12-3
<b>12.4</b>	<b>Exponential Distribution</b>	12-4
<b>12.5</b>	<b>Lognormal Distribution</b>	12-5
<b>12.6</b>	<b>Normal Distribution</b>	12-6
<b>12.7</b>	<b>Sample Distribution</b>	12-8
<b>12.8</b>	<b>Statistics Block and Correlations</b>	12-11
<b>12.9</b>	<b>User-Created Hybrid Distributions</b>	12-14
<b>12.10</b>	<b>MonteCarlo Block</b>	12-15
<b>12.11</b>	<b>Simulation Types</b>	12-18
<b>12.12</b>	<b>Parameter Sweeps</b>	12-22
<b>12.13</b>	<b>Parameter Labels</b>	12-28
<b>12.14</b>	<b>Output Labels and Goals</b>	12-29
<b>12.15</b>	<b>OSA90 MonteCarlo Menu</b>	12-32
<b>12.16</b>	<b>Monte Carlo Display Options</b>	12-35
<b>12.17</b>	<b>Monte Carlo Xsweep Display</b>	12-37
<b>12.18</b>	<b>Monte Carlo Parametric Display</b>	12-39
<b>12.19</b>	<b>Histograms</b>	12-41
<b>12.20</b>	<b>Run Charts</b>	12-43
<b>12.21</b>	<b>Yield</b>	12-44
<b>12.22</b>	<b>Yield Sensitivity</b>	12-45
<b>12.23</b>	<b>Display of Maximum Errors</b>	12-49
<b>12.24</b>	<b>Scatter Diagrams</b>	12-51
<b>12.25</b>	<b>Views for Monte Carlo Displays</b>	12-53
<b>12.26</b>	<b>Histogram Views</b>	12-56
<b>12.27</b>	<b>Run Chart Views</b>	12-58
<b>12.28</b>	<b>Scatter Views</b>	12-60



## 12

# Statistical Analysis

## 12.1 Overview

The statistical (Monte Carlo) analysis features in OSA90 are very useful for studying the performance of a circuit in the presence of parameter tolerances, input fluctuations and other statistical uncertainties.

The statistical models that can be used include uniform, normal (Gaussian), lognormal, exponential and sample distributions. Parameter correlations can be accommodated in multidimensional normal distributions.

The Monte Carlo analysis feature is invoked through the MonteCarlo menu. Monte Carlo analysis simulates the statistical behaviour of a circuit by computing the responses at a number of random outcomes generated from a specified statistical model. The statistical displays include

- ▷ responses versus parameter sweep for all statistical outcomes
- ▷ parametric plots of responses for all statistical outcomes
- ▷ histograms of responses
- ▷ run charts of responses
- ▷ yield estimated w.r.t. design specifications
- ▷ yield sensitivity (marginal yield versus parameter sweep)
- ▷ maximum errors (violation of specifications) of individual outcomes
- ▷ scatter diagrams between two responses

These statistical features are not limited to the built-in circuit analysis, but can also be applied to user-defined functions and external simulators connected through Datapipe.

This chapter describes the syntax for defining statistical parameters in the Expression and Model blocks, defining correlation matrices in the Statistics block, supplying samples of statistical outcomes in the Sample block, and defining simulation types, ranges and outputs in the MonteCarlo block. A description of the MonteCarlo menu options and the different types of statistical displays follows.

Statistical (yield) optimization is covered in Chapter 13.

## 12.2 Statistical Parameters

Statistical parameters can be defined in the Model and Expression blocks of the input file.

### Syntax

```
label_name: nominal { distribution keyword=value ... keyword=value };
```

**TABLE 12.1 STATISTICAL PARAMETER KEYWORDS**

Distribution	Keywords
Uniform	TOL, HIGH, LOW
Exponent	TOL, HIGH, LOW
LogNormal	TOL, HIGH, LOW
Normal	SIGMA, HIGH, LOW, CORRELATION, DDF
Sample	NAME

### Examples:

```
V1: 50 {Uniform TOL=10};
```

```
X1: ?2.5pF? {Normal SIGMA=5%};
```

The label V1 represents a statistical parameter with a constant nominal value of 50 and a uniform distribution with a tolerance of 10. The label X1 represents a statistical parameter with normal distribution. The mean value of X1 is defined as an optimization variable with an initial value of 2.5pF. The standard deviation of X1 is defined as 5 percent of its mean value.

The meaning of the keywords are more precisely defined when the individual distributions are described in Sections 12.3 to 12.7.

### Statistical Parameters in Element Models

In addition to labels, statistical parameters can also be defined in circuit element models. For example:

```
SRC 1 2 R=50 {Uniform TOL=10} C=?2.5pF? {Normal SIGMA=5%};
```

## 12.3 Uniform Distribution

### Syntax

*nominal* {Uniform TOL=*tolerance* LOW=*low* HIGH=*high*}

where *nominal* represents the nominal value, *tolerance* represents the tolerance, and *low* and *high* specify the distribution interval (the default: *low* = -1 and *high* = 1).

The tolerance can be specified by an absolute value, such as

V1: 5 {Uniform TOL=0.5};

or as a percentage of the nominal value, such as

V1: ?5? {Uniform TOL=10%};

### Generation of Outcomes

First, a normalized (dimensionless) *deviate* is generated from a uniform distribution in the interval [*low high*] (the default interval is [-1 1]).

For an absolute tolerance, the outcome is given by

$$\text{outcome} = \text{nominal} + \text{tolerance} \times \text{deviate}$$

For a relative (percentage) tolerance, the outcome is given by

$$\text{outcome} = \text{nominal} \times (1 + \text{tolerance} \times \text{deviate})$$

Example:

V1: 5 {Uniform TOL=2 LOW=-0.5 HIGH=0.7};

The interval of the normalized uniform distribution deviates is [-0.5 0.7], and the interval of the outcomes is [4 6.4].

## 12.4 Exponential Distribution

### Syntax

*nominal* {Exponent TOL=*tolerance* LOW=*low* HIGH=*high*}

where *nominal* represents the nominal value, *tolerance* represents the tolerance, and *low* and *high* specify the optional truncation limits (the default: *low* = -1 and *high* = 4).

The tolerance can be specified by an absolute value, such as

V1: 5 {Exponent TOL=0.5};

or as a percentage of the nominal value, such as

V1: ?5? {Exponent TOL=10%};

### Generation of Outcomes

First, a normalized (dimensionless) *deviate* is generated from an exponential distribution with a mean value of zero.



The exponential distribution in text books has a mean value of 1 and an interval of  $[0 \infty)$ . In OSA90 the exponential deviates are shifted to the interval  $[-1 \infty)$ , so that the mean value of the deviates is zero.

Then, *deviate* is truncated to be within the limits [*low high*]. In other words, deviates generated outside the truncation limits are discarded. Truncation limits can significantly alter the distribution of the statistical outcomes.

For an absolute tolerance, the outcome is given by

$$\text{outcome} = \text{nominal} + \text{tolerance} \times \text{deviate}$$

For a relative (percentage) tolerance, the outcome is given by

$$\text{outcome} = \text{nominal} \times (1 + \text{tolerance} \times \text{deviate})$$



## 12.5 Lognormal Distribution

### Syntax

*nominal* {Lognormal TOL=*tolerance* LOW=*low* HIGH=*high*}

where *nominal* represents the nominal value, *tolerance* represents the tolerance, and *low* and *high* specify the optional truncation limits (the default: *low* = -1.6 and *high* = 20).

The tolerance can be specified by an absolute value, such as

V1: 5 {Lognormal TOL=0.5};

or as a percentage of the nominal value, such as

V1: ?5? {Lognormal TOL=10%};

### Generation of Outcomes

First, a normalized (dimensionless) *deviate* is generated from a lognormal distribution with a mean value of zero.



The lognormal distribution in text books has a mean value of 1.6487213 ( $e^{0.5}$ ) and an interval of  $(0 \infty)$ . In OSA90 the lognormal deviates are shifted to the interval  $(-1.6487213 \infty)$ , so that the mean value of the deviates is zero.

Then, *deviate* is truncated to be within the limits [*low high*]. In other words, deviates generated outside the truncation limits are discarded. Truncation limits can significantly alter the distribution of the statistical outcomes.

For an absolute tolerance, the outcome is given by

$$\text{outcome} = \text{nominal} + \text{tolerance} \times \text{deviate}$$

For a relative (percentage) tolerance, the outcome is given by

$$\text{outcome} = \text{nominal} \times (1 + \text{tolerance} \times \text{deviate})$$

## 12.6 Normal Distribution

This section focuses on normal (Gaussian) distribution without parameter correlations. Correlations are described in Section 12.8.

### Syntax:

```
nominal {Normal SIGMA= $\sigma$  LOW=low HIGH=high DDF=k1 k2 ... kn}
```

where *nominal* represents the nominal (mean) value,  $\sigma$  specifies the standard deviation, *low* and *high* are optional truncation limits (the default: *low* = -3 and *high* = 3), and *k1*, *k2*, ..., *kn* are a set of integers representing a discrete density function.

The standard deviation can be specified by an absolute value, such as

```
V1: 5 {Normal SIGMA=0.25};
```

or as a percentage of the mean value, such as

```
V1: 5 {Normal SIGMA=5%};
```

### Generation of Outcomes

First, a normalized (dimensionless) *deviate* is generated from a normal distribution with a zero mean value and unit standard deviation.

Then, *deviate* is truncated to be within the limits [*low high*]. In other words, deviates generated outside the truncation limits are discarded. Truncation limits can significantly alter the distribution of the statistical outcomes.

If the standard deviation  $\sigma$  is specified by an absolute value, the outcome is given by

$$\text{outcome} = \text{nominal} + \sigma \times \text{deviate}$$

If  $\sigma$  represents a relative (percentage) value, the outcome is given by

$$\text{outcome} = \text{nominal} \times (1 + \sigma \times \text{deviate})$$

## Discrete Density Function

The random deviates generated from a normal distribution can be mapped into a discrete distribution through a histogram specified by the keyword DDF. The histogram serves as a discrete density function which approximates a non-Gaussian distribution.

Using this approach, the density function derived from a non-Gaussian sample distribution can be reproduced reasonably well in the Monte Carlo analysis.

To obtain the discrete density function (DDF) from the histogram of a statistical sample, the sample outcomes are divided into  $N$  bins (subintervals), where  $N$  is typically between 10 and 20. The number of outcomes which fall into each bin is recorded, and the numeric record is used as the DDF.

Example:

```
V1: 5 (Normal SIGMA=10% DDF=2 8 15 16 18 18 12 4 2 4);
```

where the discrete density function corresponds to a 10-bin histogram as shown in Fig. 12.1.

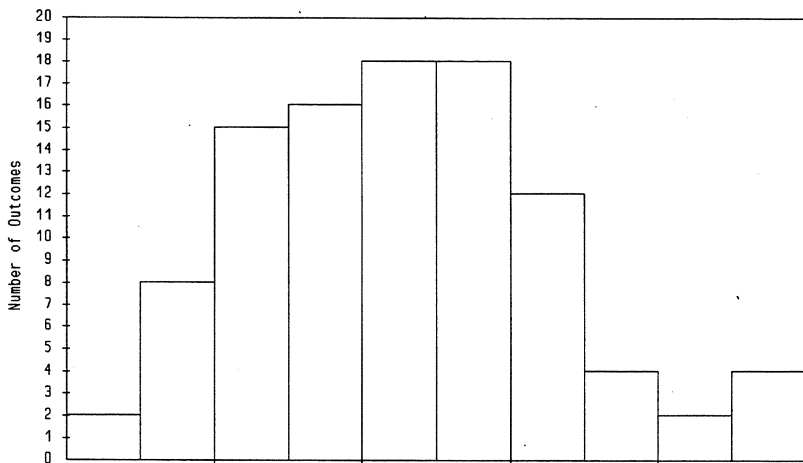


Fig. 12.1 Histogram used to specify the discrete density function.

If DDF is specified, the outcomes are mapped to the discrete density function through the respective cumulative distribution functions. Appropriate scaling and shifting may be performed to preserve the mean and standard deviation.

## 12.7 Sample Distribution

The Sample block allows you to supply a specific sample of outcomes for the purpose of defining statistical parameters.

### Sample Block

#### Syntax

```
Sample
  FORMAT  name1  name2  ...  namen;
          x11    x12    ...  x1n
          x21    x22    ...  x2n
          ...
          xml    xm2    ...  xmm
End
```

where *name1*, *name2*, ..., *namen* represent the names of *n* statistical parameters, *x1i*, ..., *xmi* are *m* outcomes for the *i*th statistical parameter.

Example:

```
Sample
  FORMAT  X1      C2(pF)  RDS;
          4.987  ...      ...
          5.123  ...      ...
          ...
          4.855  ...      ...
End
```

This supplies a sample of outcomes for three statistical parameters identified as X1, C2 and RDS. It also indicates that the outcomes for C2 are supplied in pF.

The Sample block may contain more than one set of data, for example:

```
Sample
  FORMAT  X1      C2(pF)  RDS;
          ...      ...      ...
          ...      ...      ...

  FORMAT  TAU(ps)  LS(nH);
          ...      ...
          ...      ...
End
```

which includes five statistical parameters in two sets of data (two samples).

The sample sizes of different data sets do not have to be identical.

## Statistical Parameters with Sample Distribution

### Syntax:

```
label: nominal (Sample NAME=name);
```

where *name* must have been defined in the Sample block.  
 "NAME=*name*" can be omitted if *name* is identical to *label*.

Example:

```
CX1: 5pF (Sample NAME=C2);
```

```
RDS: 2.7 (Sample);
```

CX1 is defined as a statistical parameter of sample distribution and its outcomes are supplied in the Sample block under the name C2.

RDS is also a statistical parameter of sample distribution. Since the sample name is omitted, it is assumed to be identical to the label name, therefore the Sample block must have included a sample of outcomes under the name RDS.

Sample distributions can also be assigned to circuit model parameters directly:

```
IND 1 2 L = 5nH (Sample NAME=LS);
```

In this case the sample name is required, since no label name is available.

## Generation of Outcomes

For statistical parameters of sample distribution, the outcomes supplied in the Sample block are directly used.

Example:

```
Sample
  FORMAT  X1 .  C2(pF)  ...
           4.987 23.55  ...
           5.123 27.04  ...
           ...
           4.855 21.82  ...
End
```

In Monte Carlo analysis, the outcomes for X1 and C2 are directly taken from the Sample block: 4.987, 5.123, ..., 4.855 for X1, and 23.55pF, 27.04pF, ..., 21.82pF for C2.

The nominal value defined for a statistical parameter of sample distribution is irrelevant in Monte Carlo analysis, since the parameter value is directly given by the outcomes.

## Recycling of the Outcomes

If the number of outcomes specified in the MonteCarlo block for Monte Carlo analysis exceeds the number of outcomes supplied in the Sample block (i.e., the sample size), then the supplied outcomes are recycled (reused).

To illustrate, if 50 outcomes are supplied in the Sample block for a statistical parameter but 200 outcomes are specified in the MonteCarlo block, then in the Monte Carlo analysis the set of 50 outcomes will be used 4 times. For example, the 1st, 51st, 101st and 151st outcomes will have identical values.

## Sample Block Generated by Statistical Modeling

When you use HarPE for statistical modeling, a Sample block is automatically created to contain the extracted parameter values from multi-device measurements. The Sample block generated by HarPE can be used directly in OSA90.

## 12.8 Statistics Block and Correlations

You can define correlations between statistical parameters of a normal distribution. First, you supply a correlation matrix in the Statistics block, and then link the matrix to a set of statistical parameters using the CORRELATION keyword.

### Syntax

#### Statistics

```
CORRELATION: name Dimension=n Format=format;
  c11 c12 ... c1n
  c21 c22 ... c2n
  ...
  cn1 cn2 ... cnm
```

End

#### Model

```
label: nominal (Normal SIGMA= $\sigma$  CORRELATION=name[i]);
  ...
```

End

where *name* is a string identifying a correlation matrix defined in the Statistics block, *c<sub>ij</sub>* is the correlation coefficient between two statistical parameters indexed *i* and *j*.



The values of the correlation coefficients must be between -1 and +1. The matrix must be symmetrical and positive semidefinite, with unit diagonal elements.

### Correlation Matrix Formats

Correlation coefficients can be entered in one of the four available formats, namely, FULL, UPPER, LOWER and SPARSE.

The default is the FULL format, which means that the complete matrix is entered.

Example:

```
CORRELATION: Cor_Mat Dimension=4 Format=FULL;
  1.0  0.2  0.4  0.6
  0.2  1.0  0.8  0.0
  0.4  0.8  1.0  0.1
  0.6  0.0  0.1  1.0
```

**Syntax of the UPPER format**

```
CORRELATION: name Dimension=n Format=UPPER;
      c11 c12 ... c1n
          c22 ... c2n
          ...
          cnn
```

Only the upper-right triangular portion of the correlation matrix is entered.  
The diagonal elements must be included.

Example:

```
CORRELATION: Cor_Mat Dimension=4 Format=UPPER;
      1.0  0.2  0.4  0.6
          1.0  0.8  0.0
              1.0  0.1
                  1.0
```

**Syntax of the LOWER format**

```
CORRELATION: name Dimension=n Format=LOWER;
      c11
      c21 c22
      ...
      cn1 cn2 ... cnn
```

Only the lower-left triangular portion of the correlation matrix is entered.  
The diagonal elements must be included.

Example:

```
CORRELATION: Cor_Mat Dimension=4 Format=LOWER;
      1.0
      0.2  1.0
      0.4  0.8  1.0
      0.6  0.0  0.1  1.0
```



**Syntax of the SPARSE format**

```

CORRELATION: name Dimension=n Format=SPARSE;
      i1 j1 c(i1,j1)
      i2 j2 c(i2,j2)
      ...
      im jm c(im,jm)

```

Only the nonzero and off-diagonal elements are entered.  $i1, i2, \dots, im$  are row indices,  $j1, j2, \dots, jm$  are column indices, and  $c(i1,j1), \dots, c(im,jm)$  are the corresponding elements of the correlation matrix.

Example:

```

CORRELATION: Cor_Mat Dimension=4 Format=SPARSE;
      1 2 0.2
      1 3 0.4
      1 4 0.6
      2 3 0.8
      3 4 0.1

```

**Indices of Correlated Statistical Parameters**

The CORRELATION keyword is used to link a set of statistical parameters to a correlation matrix defined in the Statistics block.

Example:

```

V1: 5 {Normal SIGMA=10% CORRELATION=Cor_Mat[2]};
V2: 2 {Normal SIGMA=5% CORRELATION=Cor_Mat[3]};

```

This indicates that the correlation coefficients of V1 and V2 are given by the 2nd and 3rd rows (columns), respectively, of the correlation matrix defined in the Statistics block and identified by the string name "Cor\_Mat". The correlation coefficient between V1 and V2, for instance, is the element at position (2,3).

**Multiple Correlation Matrices**

If necessary, you can define a number of correlation matrices in the Statistics block, each matrix representing a set of mutually correlated statistical parameters. The dimension of each correlation matrix is equal to the size of the corresponding set of statistical parameters. Each statistical parameter can belong to one set only.

## 12.9 User-Created Hybrid Distributions

You can combine different statistical distributions in an expression to create a hybrid distribution, such as

```
V_Normal: 5 {Normal SIGMA=5%};  
V_Uniform: 3 {Uniform TOL=10%};  
  
V_Hybrid = V_Normal + V_Uniform;
```

Another example:

```
V_Normal: 0 {Normal SIGMA=1};  
V_Computed = 50 + exp(V_Normal);
```

In this example, `V_Normal` represents a normal distribution with a zero mean and a unit standard deviation. `V_Computed` actually represents a lognormal distribution.

Hierarchical (multi-level) distributions can also be defined. For instance,

```
V_Low_Level: 5 {Normal SIGMA=10%};  
  
V_High_Level: V_Low_Level {Normal SIGMA=15%};
```

This is a two-level hierarchical distribution. The nominal of `V_High_Level` is itself a statistical parameter with the normal distribution specified by `V_Low_Level`. `V_Low_Level` may represent, for example, a "global" statistical uncertainty, and `V_High_Level` may model a "local" uncertainty on top of the global uncertainty.

## 12.10 MonteCarlo Block

The MonteCarlo block allows you to define the simulation types, sweep labels, sweep ranges, output labels and yield specifications for statistical Monte Carlo analysis.

### Syntax:

```

MonteCarlo
  sweep_set N_Outcomes=n;
  sweep_set;
  ...

  sweep_set;
End

```

where each *sweep\_set* is a statement delimited by a semicolon ";".

In the first sweep set, you must define the number of statistical outcomes to be generated in the Monte Carlo analysis: *N\_Outcomes=n*, which applies to all sweep sets.

### Syntax of Monte Carlo sweep sets:

```

type: sweep_label_1: sweep_range
      sweep_label_2: sweep_range
      label=expression label=expression ... label=expression
      output_label ...
      output_label < goal ...
      output_label > goal ...;

```

where *sweep\_label\_1* and *sweep\_label\_2* will sweep through multiple values defined by the corresponding *sweep\_range*, each *label* will be assigned a value given by the corresponding *expression*, *output\_label* represents a response of interest, and *goal* represents a specification for yield calculation.

### Example:

```

MonteCarlo
  AC: FREQ: from 2GHZ to 18GHZ step=1GHZ
      MS21 > 2.2, MS21 < 2.6
      N_Outcomes=200;

  HB: Signal Power: -10dBm -5dBm 0dBm
      FREQ: from 5GHZ to 12GHZ step=1GHZ
      Omega=(2.0 * PI * FREQ)
      Bias_Voltage=0.75V
      Pout_dBm[2], Efficiency > 0.75;
End

```

## Number of Outcomes

You must specify the number of random outcomes to be generated in the Monte Carlo analysis. The number of outcomes must be defined in the first sweep set and the same number of outcomes applies to all the sweep sets.

## Simulation Type

The simulation type is defined by a keyword representing the type of circuit analysis (DC, small-signal AC or harmonic balance) involved in the calculation of the output labels. You can omit the simulation type if the sweep set does not require circuit analysis.

## Sweep Labels

Sweep labels represent parameter sweeps (i.e., simulation loops). A sweep label loops through a set of (multiple) values defined in the sweep range. One or two independent sweep labels can be defined for each sweep set.

## Parameter Labels

A parameter label is assigned a single value or expression. In the example shown at the beginning of this section, there are two parameter labels, namely `Omega` and `Bias_Voltage`. `Bias_Voltage` is assigned a single value and `Omega` is to be evaluated from an expression.

## Output Labels

Output labels represent the functions and responses you wish to be calculated and displayed. In the example shown at the beginning of this section, the output labels are `MS21`, `PS21`, `Pout_dBm[2]` and `Efficiency`.

## Specifications for Yield Calculation

You need to define one or more specifications if you wish to calculate the yield. You can define upper specifications, such as

*output\_label < goal*

and lower specifications, such as

*output\_label > goal*

as illustrated in Fig. 12.2.

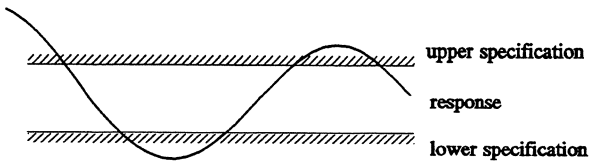


Fig. 12.2 Upper and lower specifications.

The yield estimated by Monte Carlo analysis is given by

$$\text{Yield} = N\_Pass / N\_Outcomes$$

where  $N\_Pass$  represents the number of "passed" outcomes, and  $N\_Outcomes$  is the total number of outcomes.

A "passed" outcome means that for that outcome all the responses (output labels) which have associated goals, from all the sweep sets in the MonteCarlo block, satisfy the specifications at all the sweep points. If any of the output labels associated with a goal from any of the sweep sets violates the specification at any of the sweep points, then that outcome is considered failed.

### Output Labels with Goals Omitted

Monte Carlo sweep sets can contain output labels without a *goal*, which represent responses and functions you wish to be calculated and displayed but not to be included in the yield calculation. Such output labels do not affect the yield, i.e., regardless of their values, they neither cause an outcome to pass nor cause it to fail.

### Single Specifications Are Not Allowed

Single specifications of the form

$$\text{output\_label} = \text{goal}$$

are not allowed because such exact equality specifications will almost always lead to a zero yield which does not provide any useful information. Any single specification in nominal design should be replaced by a pair of upper and lower specifications to allow a "window" appropriate for the response.



There are many similarities between the MonteCarlo, Sweep and Specification blocks. Therefore, you may find many of the details in the following sections already described in Chapters 9 and 11.

## 12.11 Simulation Types

To calculate and display output labels which represent circuit responses or functions of circuit responses, you must indicate the type of required circuit simulation using one of the keywords listed in Table 12.2.

**TABLE 12.2 SIMULATION TYPES**

Keyword	Type of Simulation
DC	DC circuit analysis
AC	small-signal AC circuit analysis
HB	large-signal harmonic balance circuit analysis

The simulation type keyword, followed by a colon, must be the first entry in a sweep set:

DC: ..... ;


AC: ..... ;

HB: ..... ;

Only one simulation type keyword is allowed in a sweep set.

### Built-In Circuit Responses

The simulation type determines which set of built-in circuit responses will be calculated. For instance, if the keyword DC is specified for a sweep set, then DC responses will be calculated for that sweep set.

 DC, small-signal and large-signal circuit responses are described in Chapter 6.

The output labels of a sweep set can involve only one type of circuit responses. For example, if the simulation type of a sweep set is DC, then the output labels in that set may include DC responses and functions of DC responses, but not small-signal or harmonic balance responses.

### Generic Numerical Simulation

If a sweep set does not begin with a simulation type keyword, then generic numerical simulation is assumed. All the output labels must be defined by generic expressions and/or Datapipe. In other words, the output labels cannot be functions of circuit responses.

## Frequency Labels

For AC and HB simulation types, you must specify the frequency values by means of the label `FREQ`.

For AC, `FREQ` specifies the frequency for the small-signal circuit analysis. For HB, `FREQ` specifies the fundamental frequency of the AC sources. During DC analysis, the value of `FREQ` is automatically set to zero.

You can define a sweep range for the label `FREQ` (see Section 12.12):

```
FREQ: from 2GHZ to 16GHZ step=1GHZ
```

or, you can assign it a single value, such as

```
FREQ = 7GHZ
```

## Two-Tone Frequencies

In the case of two-tone excitations, `FREQ` specifies the first tone frequency, and the second tone frequency is specified by the keyword `FREQ2`.

Example:

```
HB: FREQ=900MHZ FREQ2=907MHZ ...;
```

where the tone frequencies are specified as 900MHZ and 907MHZ.


The keyword `FREQ2` cannot be directly defined as a sweep label. However, you can still sweep the second tone frequency, by tying `FREQ2` to another label:

```
Second_Tone_Freq: from 1GHZ to 5GHZ step=1GHZ
FREQ2 = Second_Tone_Freq
```

The following example ties the two tone frequencies together:

```
FREQ: from 1GHZ to 15GHZ step=2GHZ
FREQ2 = (FREQ + IM_Freq)
```

where `IM_Freq` is a user-defined label representing the difference between the two tone frequencies.

 See Chapter 6 for a discussion on the generation of intermodulation frequencies in two-tone simulation.

## The HARM Keyword

In the CIRCUIt statement in the Model block, you can specify the highest harmonic to be included in harmonic balance simulation (see Chapter 6 for further details).

### Syntax

```
CIRCUIt HARM=m ...;
```

*m* is the highest harmonic to be included in harmonic balance simulation,  $m0 \leq m \leq 16$ , where *m0* is the highest harmonic among all the sources.

### Example:

```
Model
...
CIRCUIt ... HARM=5;
...
End
```

This becomes the "global" default which applies to all sweep sets of the HB type.

You can also specify the highest harmonic to be included in harmonic balance simulation for individual sweep sets of the HB type using the keyword HARM.

### Syntax

```
HB: HARM=m1 ...;
```

*m1* specifies the highest harmonic for the sweep set,  $m0 \leq m1 \leq m$ , where *m0* is the highest harmonic among all the sources and *m* is the global default.

### Example:

```
MonteCarlo
HB: HARM=4 ...;
...
End
```

Sweep sets for which the HARM keyword is not explicitly specified will assume the global default.



## The RREF Keyword

For small-signal AC sweep sets (i.e., the simulation type is AC), the  $S$  parameters are, by default, calculated with respect to the actual port terminations (impedances) as specified in the PORT definitions in the Model block of the input file (see Chapter 6). You can override this default by specifying the reference impedance using the keyword RREF.

### Syntax

```
AC: RREF=x ...;
```

where  $x$  specifies the  $S$ -parameter reference impedance for the sweep set, and  $x$  must be a positive constant value or constant label.

### Example:

```
MonteCarlo
  AC: RREF=60 ...;
...
End
```

This will work much the same way as though the impedance of 60 ohm (purely resistive) were specified for each port as its terminating impedance. The difference is that this definition takes effect only for the sweep set in which it is specified.

## 12.12 Parameter Sweeps

Parameter sweeps are defined by means of sweep labels and sweep ranges (values).

### Syntax

*sweep\_label*:  $x_1, x_2, \dots, x_n$

where *sweep\_label* is an existing label and  $x_1, x_2, \dots, x_n$  are numerical values.

Example:

Width: 1.1, 1.2, 1.7, 3.5, 5

where the label Width must have already been defined.



The sweep label must be followed by a colon ":". You cannot use an equal sign "=" instead of the colon. Colon is chosen to designate sweep labels, and equal sign is reserved for assignment of a single value. For example,

Width = 1.1, 1.2, 1.7, 3.5, 5

would be mistaken as an assignment of a single value, namely 1.1, to Width.

## Sweep an Interval with Uniform Step Size

### Syntax

*sweep\_label*: from  $x_1$  to  $x_2$  step= $x_3$

where *sweep\_label* is an existing label and  $x_1, x_2$  and  $x_3$  are numerical values. This translates into a set of values as:  $x_1, x_1+x_3, x_1+2x_3, \dots, x_2$ .

Example:

Width: from 1.5 to 5 step=0.5

This translates into a set of values for Width as 1.5, 2, 2.5, ..., 5.



The upper limit of the interval, namely  $x_2$ , is always included in the translated values, even if the step size does not divide the interval evenly. For example,

Width: from 1.5 to 2.75 step=0.5

The translated values are 1.5, 2, 2.5, 2.75, which include the upper limit 2.75.

You can specify a negative step size, such as

Width: from 5 to 1.5 step=-0.5

The translated values are 5, 4.5, 4, ..., 1.5. In this case, the lower limit must be greater than the upper limit, i.e.,  $x1 > x2$ .

The number of subintervals is  $n = (x2 - x1) / x3$ , if  $x3$  divides  $(x2 - x1)$  evenly, or else  $n = (x2 - x1) / x3 + 1$ , (the number of values is  $n + 1$ ). The limit is  $n \leq 1023$ .

## Specify the Number of Steps

### Syntax

*sweep\_label*: from *x1* to *x2* N=*n*

where *sweep\_label* is an existing label, *x1* and *x2* are numerical values, *n* is an integer,  $1 \leq n \leq 1023$ . The step size is  $(x2 - x1) / n$ . The number of values is  $n + 1$ .

Example:

Width: from 1 to 5 N=8

This translates into a step size of  $(5 - 1) / 8 = 0.5$ , and a set of values as 1, 1.5, 2, ..., 5. Note that the total number of values is 9.

## Sweep an Interval with Exponential Step Size

### Syntax

*sweep\_label*: from *x1* to *x2* ratio=*x3*

where *sweep\_label* is an existing label and *x1*, *x2* and *x3* are numerical values. This translates into a set of values as:  $x1, x1 \cdot x3, x1 \cdot x3^2, \dots, x2$ .

Example:

Width: from 3 to 96 ratio=2

This translates into a set of values for Width as 3, 6, 12, 24, 48, 96.



The upper limit of the interval, namely  $x_2$ , is always included in the translated values, even if it does not coincide with the prescribed ratio. For example,

Width: from 3 to 30 ratio=2

The translated values are 3, 6, 12, 24, 30.

The values  $x_1$ ,  $x_2$  and  $x_3$  must be consistent, i.e.,

$x_3 > 1$  if  $x_2 > x_1$

$x_3 < 1$  if  $x_2 < x_1$

## Specify the Number of Exponential Steps

### Syntax

*sweep\_label*: from  $x_1$  to  $x_2$  NEXP= $n$

where *sweep\_label* is an existing label,  $x_1$  and  $x_2$  are numerical values,  $n$  is an integer,  $1 \leq n \leq 1023$ . The step ratio is  $(x_2 / x_1)^{1/n}$ . The number of values is  $n + 1$ .

Example:

Width: from 3 to 96 NEXP=5

This is equivalent to specifying a step ratio of  $(96 / 3)^{1/5} = 2$ . The values for sweeping Width are 3, 6, 12, 24, 48, 96. Note that the total number of values is 6.

## Combine Sweep Intervals and Points

For a sweep label, you can specify an arbitrary combination of uniformly spaced sweep intervals, exponentially spaced sweep intervals and discrete points. For example:

Width: 0.75 0.8 from 1 to 5 step=1  
from 7 to 15 step=2 19.99

which translates into the following set of values:

0.75 0.8 1 2 3 4 5 7 9 11 13 15 19.99



The set of values assigned to a sweep label will not be sorted by the file parser. During simulation, the values are assigned in the exact order as they appear in the input file. If the values need to be processed in a particular order (ascending, descending, etc.), they must be entered in the desired order.

## What Can Be Sweep Labels

You can use the predefined label FREQ as a sweep label:

```
FREQ: from 4GHZ to 20GHZ step=2GHZ
```

You can also sweep any labels which are defined in the Expression and Model blocks to represent numerical constants or optimization variables.

For example, suppose the following labels are defined in the Expression (or Model) block:

```
Width: 3;
Length: ?10cm?;
Area = Width * Length;
```

You can use Width and Length as sweep labels, because they represent a constant and an optimization variable, respectively. But Area cannot be a sweep label because it represents a formula.

Once a parameter sweep is finished during simulation, the value of the sweep label is reset to its original value, i.e., the constant or nominal value as defined in the Expression or Model block.

## Macro Constants and Expressions

Macro constants, labels and expressions can be used in the definition of sweep ranges (values). For example,


```
#define MIN_X      1.0
#define X_RANGE    8.0

X_Step = 2;
...

X: from MIN_X to (MIN_X + X_RANGE) step=X_Step
```

where MIN\_X and X\_RANGE are macro constants, X\_Step is a label, and the upper limit of the sweep interval is defined by an expression.

Only constant labels can be used. Expressions, if used, must be enclosed in a pair of parentheses to avoid ambiguity.

 See Chapter 3 for further details on text macros.

## Two-Dimensional Sweep

OSA90 permits up to two independent sweep labels per sweep set. If a sweep set contains two sweep labels, the first sweep label is called the **outer sweep label**, and the second one is called the **inner sweep label**.

Example:

```
Width: from 2 to 8 step=2
Length: 3, 4, 5
```

Width is the outer sweep label and Length is the inner sweep label.

The two-dimensional sweep defined in the above example translates into pairs of values for the pair of labels (Width, Length) in the following (row-wise) order:

```
(2,3), (2,4), (2,5)
(4,3), (4,4), (4,5)
(6,3), (6,4), (6,5)
(8,3), (8,4), (8,5)
```

It is equivalent to a double loop in the "C" programming language as

```
for (Width = 2; Width <= 8; Width += 2) {
    for (Length = 3; Length <= 5; Length += 1) {
        ...
    }
}
```

## The Order of Sweeps in Circuit Simulation

The order of the two sweep labels can be significant in circuit simulation. For instance, if the following two-dimensional sweep is requested for small-signal AC simulation

```
Bias_Voltage: from 1V to 3V step=0.5V
FREQ: from 6GHZ to 18GHZ step=2GHZ
```

it is desirable to have the bias voltage as the outer sweep label and the frequency as the inner sweep label (as shown). Because, the nonlinear devices, if used, need to be linearized for small-signal analysis by solving the DC nonlinear equations. A new solution of the DC nonlinear equations is needed only when the bias voltage changes, but not when the frequency changes. Therefore, having the bias voltage as the outer sweep label minimizes the number of nonlinear DC solutions required for the sweep set.

If a large-signal harmonic balance simulation involves a power sweep (i.e., a sweep of the input power of an AC source), it is preferable to have the power sweep as the inner sweep, because once the nonlinear harmonic balance equations are solved at one point, it is relatively easy to obtain a new solution after an incremental change in the power level. It is usually more time-consuming to obtain a new solution after changing other parameters.

## Sweeping Array Indices

One very useful application of parameter sweeps is to sweep indices of arrays. For example,

```

Model
  A[10] = [a1 a2 ... a10];
  B[10] = [b1 b2 ... b10];

  K: 1;

  C = A[K] + B[K] ...;
  ...
end

MonteCarlo
  K: from 1 to 10 step=1 ...;
  ...
end

```

where the sweep label *K* is used as an index of reference to the arrays *A* and *B* in the expression by which *C* is defined.

☞ See Chapter 4 for further details on arrays.

An example of sweeping the indices of matrices:

```

Model
  MS_Data[3,3] = [ ... ];

  ....

  CIRCUIT ...;

  I: 1;
  J: 1;

  MS_Error = MS[I,J] - MS_Data[I,J];
  ...
end

MonteCarlo
  AC: I: from 1 to 3 step=1
      J: from 1 to 3 step=1 MS_Error ...;
  ...
end

```

where the matrix *MS* contains the calculated small-signal *S* parameter magnitudes (built-in response) and *MS\_Data* may represent measured data.

## 12.13 Parameter Labels

In addition to sweep labels, you can also define parameter labels for a sweep set. Parameter labels are not to be swept, rather, they are assigned a single value or to be evaluated from an expression.

### Syntax

$$\text{parameter\_label} = x$$

where *parameter\_label* is a label and *x* can be a value, a label or an expression.

Examples:

$$\text{Drain\_Bias} = 4\text{V}$$

$$\text{Omega} = (2 * \text{PI} * \text{FREQ})$$

where *Drain\_Bias* is a parameter label which is assigned a constant value, and *Omega* is a parameter label to be evaluated from an expression.

Up to 64 parameter labels can be defined in each sweep set.

Similar to sweep labels, parameter labels must be defined in the Model or Expression block and represent numerical constants or optimization variables.

### Parameter Labels as Functions of Sweep Labels

Parameter labels can be functions of the sweep labels defined in the same sweep set.

Example:

$$\text{FREQ: from 1GHZ to 10GHZ step=1GHZ}$$

$$\text{Omega} = (2 * \text{PI} * \text{FREQ})$$

The formula (expression) must be enclosed within parentheses to avoid ambiguity.

Using this concept, you can create synchronous sweeps, such as

$$\text{K: from 0 to 20 step=1}$$

$$\text{XK} = (\text{X0} + \text{K} * \text{DX})$$

$$\text{YK} = (\text{Y0} + \text{K} * \text{DY})$$

$$\text{ZK} = (\text{Z0} + \text{K} * \text{DZ})$$

where a number of parameter labels are tied together through a sweep label. In other words, this creates a one-dimensional sweep of several parameters.



## 12.14 Output Labels and Goals

Output labels are required components of a sweep set. They represent the responses and functions to be calculated during Monte Carlo simulation and subsequently displayed.

Output labels may include any of the labels defined in the Model and Expression blocks: constants, variables, circuit responses, postprocessed functions, outputs from external programs via Datapipe, etc.

Example:

```
MonteCarlo
  DC: ... Idrain_DC N_Outcomes=200;

  AC: ... MS11 < 0.1 MS21_dB > 7 MS21_dB < 9;

  HB: ... MVoutput[2] < 0.05, Conversion_Gain > 2.5;

  ... AA, BB < 2, CC > 3;
end
```

The first sweep set includes Idrain\_DC (DC circuit response) as an output label. The output labels of the second sweep set include MS11 (small-signal  $S$  parameter) and MS21\_dB (postprocessed response), and the output labels of the third sweep set include MVoutput[2] (the second harmonic component of large-signal voltage spectrum) and Conversion\_Gain (postprocessed response). The last sweep set does not have an explicitly specified simulation type, which means that the output labels AA, BB and CC are defined by generic expressions which do not require circuit simulation.



If any circuit responses are included in a sweep set, they must be consistent with the simulation type (DC, AC, HB). See Chapter 6 for the built-in response labels corresponding to each simulation type.

During the Monte Carlo analysis, the output labels are evaluated for each random outcome. The values of an output label for all the outcomes are collectively called a statistical sample (statistical response) of that output label.

### Calculation of the Yield

If you wish to calculate the yield, you need to define specification in the MonteCarlo block.

You can define upper specifications, such as

$$\text{output\_label} < \text{goal}$$

and lower specifications, such as

$$\text{output\_label} > \text{goal}$$

respectively (see also Section 12.10).

The yield estimated by Monte Carlo analysis is given by

$$Yield = N\_Pass / N\_Outcomes$$

where *N\_Pass* represents the number of "passed" outcomes, and *N\_Outcomes* is the total number of outcomes.

A "passed" outcome means that for that outcome all the responses (output labels) which have associated goals, from all the sweep sets in the MonteCarlo block, satisfy the specifications at all the sweep points. If any of the output labels associated with a goal from any of the sweep sets violates the specification at any of the sweep points, then that outcome is considered failed.

### Output Labels with Goals Omitted

Output labels not associated with goals represent responses and functions you wish to be calculated and displayed but not to be included in the yield calculation. Such output labels do not affect the yield, i.e., regardless of their values, they neither cause an outcome to pass nor cause it to fail.

### Goals Defined by Labels and Expressions

Example:

```

Model
  Gain_slope = 5.0 - 0.1 * FREQ;
  ...
End

MonteCarlo
  AC: FREQ: from 2 to 20 step=1
  MS21 > Gain_slope N_Outcomes=500;
End
    
```

Goals can also be defined by expressions enclosed within a pair of parentheses.

Example:

```

MonteCarlo
  AC: FREQ: from 2 to 20 step=1
  MS21 > (5.0 - 0.1 * FREQ) N_Outcomes=500;
End
    
```

## Arrays as Output Labels

Arrays can be used as output labels and goals.

For example, suppose that  $A1[10]$  is an array and a specification is defined as

$$A1 > Goal$$

If  $Goal$  is a scalar, it is equivalent to creating a specification for each array element with the same goal:

$$\begin{aligned} A1[1] &> Goal \\ A1[2] &> Goal \\ &\dots \end{aligned}$$

$$A1[10] > Goal$$

If the  $Goal$  is also an array, it must have the same dimension as  $A1$ . In this case, OSA90 interprets the specification as

$$\begin{aligned} A1[1] &> Goal[1] \\ A1[2] &> Goal[2] \\ &\dots \\ A1[10] &> Goal[10] \end{aligned}$$

## Maximum Number of Output Labels

The maximum number of output labels per sweep set is limited to 512. An output array is equivalent to as many output labels as the number of its elements. If more is needed, you can divide and spread the output labels among a number of sweep sets of the same simulation type.

## 12.15 OSA90 MonteCarlo Menu

The menu options under the “MonteCarlo” menu control Monte Carlo simulation and statistical displays.

The steps needed for Monte Carlo simulation and display are outlined as follows.

- ▶ Prepare an appropriate input file, including one or more statistical parameters and a MonteCarlo block.
- ▶ Compile the input file.
- ▶ Select one of the available display formats listed in Table 12.3 from the “MonteCarlo” menu.

TABLE 12.3 STATISTICAL DISPLAY FORMATS

Menu Option	Brief Description
Xsweep	displays statistical responses versus parameter sweeps
Parametric	displays statistical parametric plots of responses
Histogram	displays histogram of individual statistical responses
Run Chart	displays run chart of individual statistical responses
Yield	displays the yield estimated by Monte Carlo analysis
Sensitivity	displays marginal yield versus parameter sweeps
Max Error	displays the max errors of individual outcomes
Scatter Plot	displays scatter diagram between statistical responses

## Monte Carlo Simulation

When you select a “MonteCarlo” menu option, OSA90 first performs the necessary simulation according to the MonteCarlo block.

During the Monte Carlo simulation, OSA90 displays a progress bar at the bottom of the screen, on the status bar.

While the simulation is in progress, you can interrupt it at any time by clicking of the “Stop” button located on the toolbar. After clicking on the button simulation is stopped immediately.

## Sequence of Calculations

The calculations during Monte Carlo simulation are organized in the following sequence:

- 1 Generate an outcome. Using a random number generator, each statistical parameter is given a statistically perturbed value according to its distribution as defined in the input file. The Monte Carlo simulation is complete when the specified number of outcomes are processed.
- 2 Start from the first sweep set in the MonteCarlo block. After the first sweep set is done, go on to the second sweep set, and so on. After all the sweep sets have been processed, go to step 7.
- 3 The outer sweep loop. Set the outer sweep label to its next available value. After all the values have been assigned, go to step 2 (i.e., go to the next sweep set).
- 4 The inner sweep loop. Set the inner sweep label to its next available value. After all the values have been assigned, go to step 3.
- 5 Set the parameter labels. Each parameter label is either specified with a single value or evaluated from an expression (which may depend on the sweep labels).
- 6 Perform circuit analysis if required. Calculate the output labels. Go to step 4.
- 7 Estimate the contribution to the yield by the current outcome. Go to step 1 (i.e., to generate the next outcome).

## The .mca Files

Monte Carlo analyses can be time-consuming. To avoid duplicated analyses, a feature is built into OSA90 to save the results of the Monte Carlo analysis in an encoded file, which can be retrieved and displayed in a subsequent session.

The encoded file is named after the input file by replacing the extension .ckt with .mca. To ensure that the .mca file is uniquely associated with the input file, a .mca file will be created only if the input file has the extension .ckt.

For example, if you have two input files named example.ckt and example.old, then the sensitivity analysis results for example.ckt will be saved in example.mca, but example.old will not have a .mca file.

Furthermore, to ensure that the information stored is up to date, a "time stamp" is encoded into the .mca file which reflects the date and time of the corresponding input file. OSA90 always checks this time stamp in a .mca file before using it. Therefore, any operation which changes the date and time of the input file after a .mca file is created will invalidate the stored statistical data.

A new .mca file is created in three situations:

- ▶ A new input file is read, and simulated without editing or optimization.
- ▶ An input file is edited or optimized, then saved, and simulated.
- ▶ An input file is edited or optimized, then simulated, and saved.

Here "simulated" refers to Monte Carlo simulation.


## 12.16 Monte Carlo Display Options

When you choose any display option listed in Table 12.3, excluding Yield, you will be presented with a display options dialog box. All of the display dialog boxes contain a common set of options. Certain display dialog boxes contain additional options that are applicable to that particular display type. For a list of display options and the displays to which they are applicable see Table 12.4. Some of the options listed below are context sensitive and will be described separately in the related sections.

TABLE 12.4 DISPLAY OPTIONS AND RELATED DISPLAY TYPES

Option	Applicable Display
Sweep Set	All
Zoom	All
Y-Axis	Xsweep, Parametric, Run Chart, Scatter Plot
X-Axis	Xsweep, Parametric, Histogram, Sensitivity, Scatter Plot
Included in display	Xsweep
Numerical output	Xsweep, Histogram, Run Chart, Sensitivity
Sweep Labels	Xsweep, Parametric, Histogram, Run Chart, Sensitivity, Scatter Plot
Number of bins	Histogram, Max Error
Low truncation	Histogram
High truncation	Histogram
Parametric	Parametric
Display format	Max Error
Show mean and standard deviation	Histogram
Show correlation coefficient	Scatter Plot

### Sweep Set

If you have defined more than one sweep set in the MonteCarlo block, the sweep sets can be selected for display one at a time. Click on the  button, on the same line as "Sweep Set" and select the desired sweep set.

### Zoom

By default, graphics displays are scaled to the range of the output values. You can change the display scale ("zooming") by clicking on the "Zoom" button in the dialog box. For more detailed information on the "Zoom" option please see Chapter 9.

## Numerical Output

The “Numerical output” option, where supported, allows to specify whether or not you want numerical output. The information contained in the “Numerical Output” will vary depending on the type of display chosen. For more information on the “Numerical Output” option, please see Chapter 9.

## Select Values of Sweep Labels

Several display options require fixed values of one and/or two (if defined) sweep labels.

For example, “Histogram” is a type of display that shows the number of outcomes falling into discrete bins and representing values of a response at a single point. Therefore any sweep parameter, such as `FREQ`, must be fixed (selected from an available set of values).

If a sweep label needs to be fixed then the dialog box contains an additional drop down list-box which contains the values you may choose from. This additional option is then titled using the actual name of the sweep label.



## 12.17 Monte Carlo Xsweep Display

The “Xsweep” option under the “MonteCarlo” menu displays the statistical response of one output label versus a sweep parameter. Each curve on the display represents the value of the output label versus the sweep parameter for one random outcome. The total number of curves displayed is equal to the number of outcomes specified in the MonteCarlo block, as illustrated in Fig. 12.3.

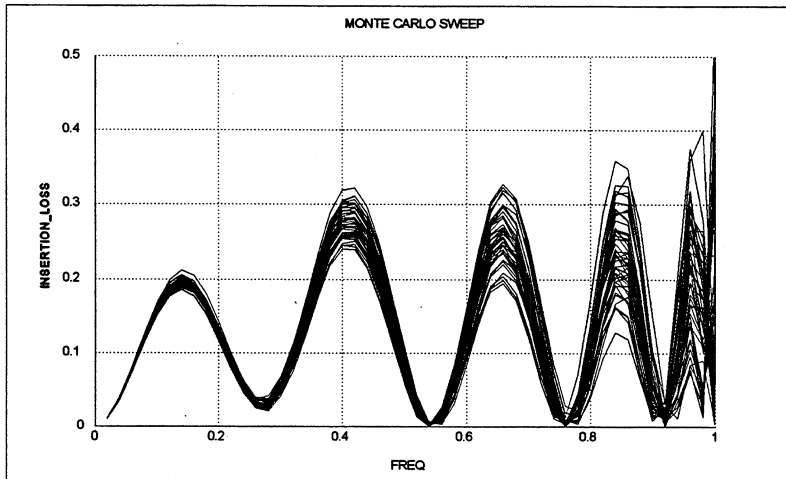


Fig. 12.3 Monte Carlo Xsweep display.

### Define the Y-axis

Xsweep is a rectangular X-Y plot, and you need to define the X-axis (horizontal) and the Y-axis (vertical).

The Y-axis represents a function of the sweep label. If a sweep set contains only one output label, then it is the only choice for the Y-axis.

For sweep sets which contain a number of output labels, an individual output label needs to be selected for display.

The “Xsweep” option can display only one output label at a time. However, you can define Graphical Views to include multiple output labels in a single Monte Carlo Xsweep display (Section 12.25).

## Define the X-axis

For the “Xsweep” display option to be available, the sweep set must contain at least one sweep label. If the sweep set contains one sweep label, then it is the only choice for the X-axis.

For sweep sets which contain two sweep labels, either one can be chosen as X-axis.

For example, consider the sweep set

```
DC:  VG: from -2 to 0 step=0.5
      VD: from 0 to 15 step=1
      ID_MA;
```

which has two sweep labels and either can be chosen for the X-axis.

Suppose that VD is chosen for the X-axis, then you can select for VG a value from those available, namely -2, -1.5, -1, -0.5 or 0. The drop down list-box to make this select will be titled “VG” (see “Select Values of Sweep Labels” in Section 12.16).

## Included in Display

This option allows you to display the statistical responses that include: “All outcomes”, “Passed outcomes” (those for which all specifications are satisfied), or “Failed outcomes” (those for which at least one specification is violated).

## 12.18 Monte Carlo Parametric Display

The "Parametric" option under the "Montecarlo" menu plots one output label versus another output label for all the statistical outcomes.

Example:

```

Expression
  Angle: 0;
  X0: 0 {Uniform TOL=2.5};
  Y0: 0 {Uniform TOL=1};

  X: cos(Angle) + X0;    ! X coordinates of points on a circle
  Y: sin(Angle) + Y0;    ! Y coordinates
end

MonteCarlo
  Angle: from 0 to (2 * PI) N=50 X Y N_outcomes=20;
end

```

The output labels  $X$  and  $Y$  are Cartesian coordinates of the points on a circle as  $\text{Angle}$  sweeps from  $0$  to  $2\pi$ . The center of the circle is defined by  $X0$  and  $Y0$  both of which are statistical parameters. The Monte Carlo parametric display of  $Y$  versus  $X$  with  $\text{Angle}$  as the parametric label shows 20 circles, each centered at one random outcome, as illustrated in Fig. 12.4.

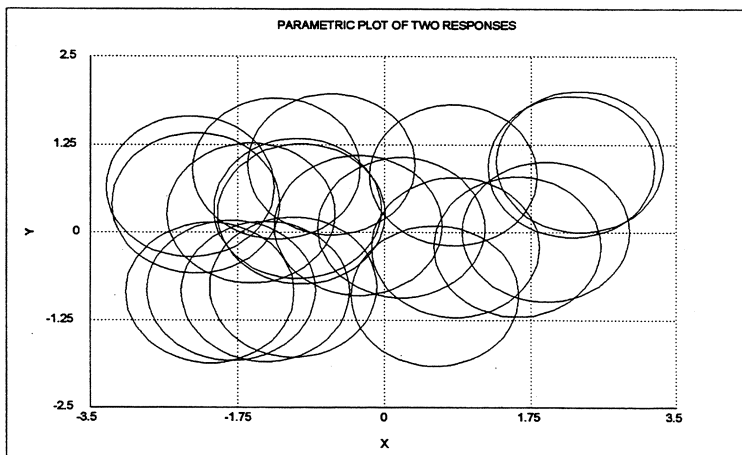


Fig. 12.4 Monte Carlo parametric display of circles.

## Parametric


For the parametric display to be applicable, a sweep set must contain one or two sweep labels. One sweep label is chosen as the parametric label which does not actually appear in the graphics. Rather, both the X-axis and Y-axis represent output labels which are functions of the parametric label.

If the sweep set contains one sweep label, then it is the only choice for the parametric label.

If there are two sweep labels, then either one can be chosen as the parametric label. The other sweep label will be listed in the dialog box to let you select one or all of the values available for that label.

## Define the X-axis and Y-axis

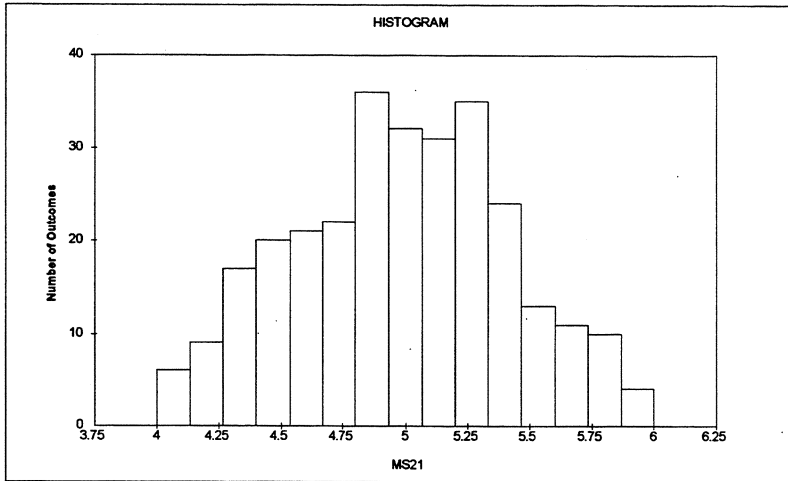
Since the parametric display plots one output label versus another output label, the sweep set must contain two or more output labels. Any one of the output labels can be selected for the X-axis and the Y-axis.

 Chapter 9 contains more illustrations of parametric display.

## 12.19 Histograms

The “Histogram” option under the “MonteCarlo” menu displays the histograms of individual output labels at a selected sweep point (see “Select Values of Sweep Labels” in Section 12.16).

A histogram divides the values of an output label into a number of evenly spaced intervals (bins). The number of values (outcomes) which fall into each bin is counted, and the result is displayed as a bar chart, as illustrated in Fig. 12.5.



*Fig. 12.5 Histogram.*

### Define the X-axis

Any one of the output labels can be selected for the X-axis of the histogram.

## Number of Bins

You can specify the number of intervals (bins) for dividing the values of the selected output label. Available choices are 5, 10, 15 and 20 bins.

## High and Low Truncation

Generally, the limits (minimum and maximum) of the output label values are not rounded figures. Consequently, the width of the bins is not a rounded figure either. For example, if the output label values range from 1.012 to 2.983, and the number of bins is selected as 10, then the width of the bins would be 0.1971.

This can be inconvenient, especially if you try to compare two histograms (such as the histograms of simulated and measured responses).

The “Low truncation” and “High truncation” options in the dialog box allow you to define rounded limits for the bin ranges. For instance, if you round the low limit from 1.012 to 1 and the high limit from 2.983 to 3, and keep the number of bins as 10, then the width of the bins would be 0.2.

The truncation limits of the histogram shown in Fig. 12.5 are 4 and 6.



The truncation limits are separate from the graphics zoom limits. The truncation limits determine the bin width, while the zoom limits determine the scale of the graphics. For example, the zoom limits of the histogram shown in Fig. 12.5 are 3.75 and 6.25, which are different from the truncation limits.

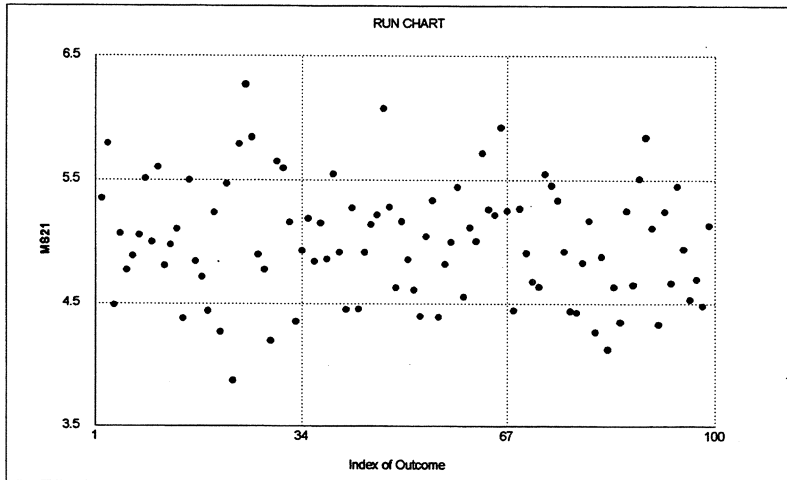
## Show Mean and Standard Deviation

The sample mean and standard deviation of the selected output label can be estimated and shown on the histogram display. You may find this information useful in some situations, but unnecessary in other situations. You can choose whether or not to display the estimated mean and standard deviation through the “Show mean and standard deviation” check box in the dialog box.

## 12.20 Run Charts

The “Run Chart” option under the “MonteCarlo” menu displays the run charts of individual output labels at a selected sweep point (see “Select Values of Sweep Labels” in Section 12.16).

A run chart plots the value of an output label versus the outcome index. The X-axis is the outcome index ranging from 1 to  $n$ , where  $n$  is the number of outcomes. The Y-axis represents the output label, as illustrated in Fig. 12.6.



*Fig. 12.6 Run chart.*

### Define the Y-axis

Any one of the output labels can be selected for the Y-axis of the run chart.

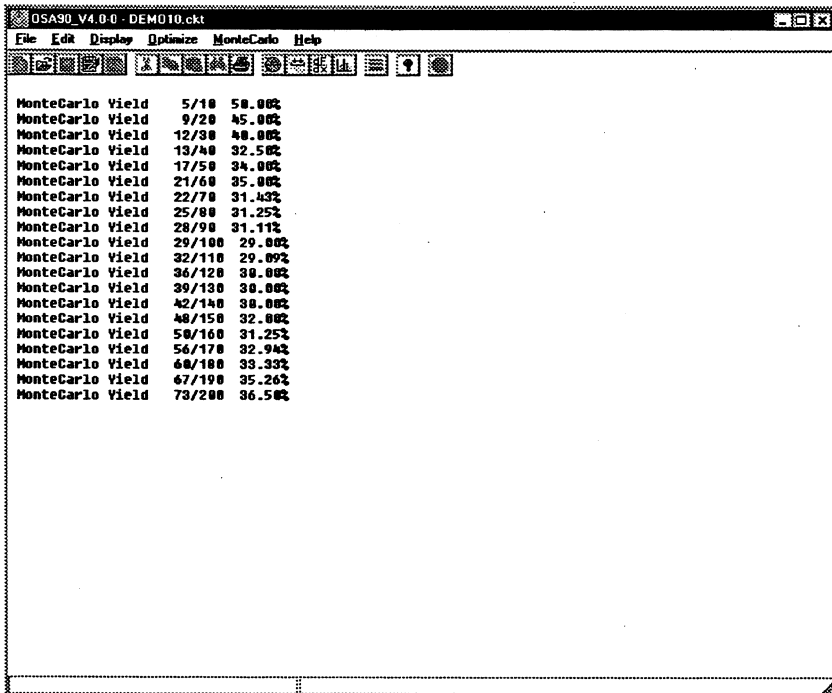
## 12.21 Yield

The “Yield” option under the “MonteCarlo” menu displays the yield numerically on the screen.

The yield is estimated and printed for every 10 outcome interval. The percentage of passed outcomes from the first 10 outcomes is calculated and printed, followed by the percentage from the first 20 outcomes, and so on.

For example, if the total number of outcomes is 200, the display on the screen may look like the illustration in Fig. 12.7.

Only those output labels for which goals are specified are taken into account in the yield calculation (see Section 12.10).



The screenshot shows a window titled "OSA90 V4.0.0 - DEMO10.ckt" with a menu bar (File, Edit, Display, Optimize, MonteCarlo, Help) and a toolbar. The main display area contains the following text:

```
MonteCarlo Yield 5/10 50.00%
MonteCarlo Yield 9/20 45.00%
MonteCarlo Yield 12/30 40.00%
MonteCarlo Yield 13/40 32.50%
MonteCarlo Yield 17/50 34.00%
MonteCarlo Yield 21/60 35.00%
MonteCarlo Yield 22/70 31.43%
MonteCarlo Yield 25/80 31.25%
MonteCarlo Yield 28/90 31.11%
MonteCarlo Yield 29/100 29.00%
MonteCarlo Yield 32/110 29.09%
MonteCarlo Yield 36/120 30.00%
MonteCarlo Yield 39/130 30.00%
MonteCarlo Yield 42/140 30.00%
MonteCarlo Yield 48/150 32.00%
MonteCarlo Yield 50/160 31.25%
MonteCarlo Yield 56/170 32.94%
MonteCarlo Yield 60/180 33.33%
MonteCarlo Yield 67/190 35.26%
MonteCarlo Yield 73/200 36.50%
```

Fig. 12.7 Yield printed numerically.



## 12.22 Yield Sensitivity

The “Sensitivity” option under the “MonteCarlo” menu displays the marginal yield versus a sweep parameter.

For this option to be available, the MonteCarlo block must contain at least one sweep label. The values of the output labels from the Monte Carlo analysis are separated into groups, each corresponding to one of the values of the sweep label. A marginal yield is calculated from each group, and the result is plotted versus the sweep label.

### Detect Critical Values of a Sweep Label

Example:

```
MonteCarlo
AC: FREQ: 1 25 50 75 100
    MS21 > 2.8 MS21 < 3.2 MS11 < 0.4 MS22 < 0.4
    N_Outcomes=500;
end
```

The display of the marginal yield versus the sweep label FREQ is illustrated in Fig. 12.8, which clearly shows that FREQ=1 corresponds to the lowest marginal yield. Since the overall yield cannot exceed the lowest marginal yield, this means that FREQ=1 is the most critical value in terms of its impact on the yield.

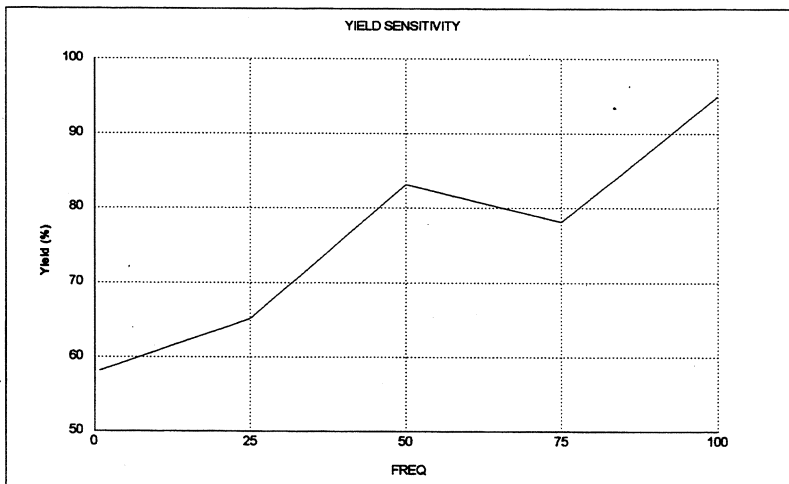


Fig. 12.8 Marginal yield versus FREQ.

## Display Yield versus a Design Variable

The "Sensitivity" feature can be used to investigate the sensitivity of the overall yield with respect to design variables.

Suppose that a design variable X is added to the MonteCarlo block in the preceding example as an additional sweep label:

```

MonteCarlo
  AC: X: from 20 to 28 step=0.5
      FREQ: 1 25 50 75 100
          MS21 > 2.8  MS21 < 3.2  MS11 < 0.4  MS22 < 0.4
          N_Outcomes=500;
end
    
```

The marginal yield for each value of X is actually equivalent to the "total" yield in the preceding example. In other words, displaying the marginal yield versus X is equivalent to displaying the yield defined in the preceding example versus the design variable X. The display is illustrated in Fig. 12.9.

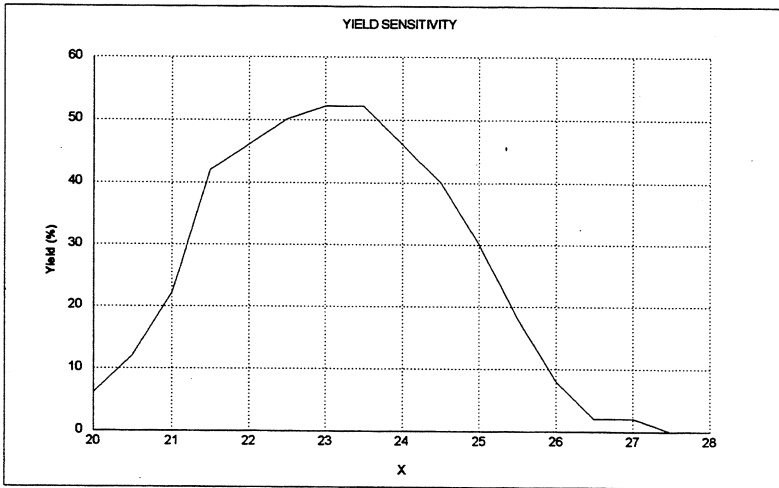


Fig. 12.9 Yield versus the design variable X.

## Display Yield versus Two Sweep Labels

When there are two sweep labels, either one can be selected for the X-axis of the yield sensitivity display.

Furthermore, you can choose to have the marginal yield calculated separately for each value of the other sweep label (i.e., the one not chosen for the X-axis).

Consider the preceding example which has two sweep labels, namely X and FREQ. If you select the sweep label X for the X-axis, the other sweep label, namely FREQ will appear as another option in the dialog box.

If you choose for FREQ the option "Combined Yield", then for each value of X, the marginal yield is calculated over all the values of FREQ, as shown in Fig. 12.9.

Alternatively, you can choose "Separate Curves" for FREQ in the dialog box, then the marginal yield is calculated for each value of X and each value of FREQ. Since the sweep range for X consists of 17 values and the sweep range for FREQ consists of 5 values, there will be a total of 85 marginal yields to be calculated. The result will be displayed as a family of 5 curves, each corresponding to one of the FREQ values, as illustrated in Fig. 12.10.

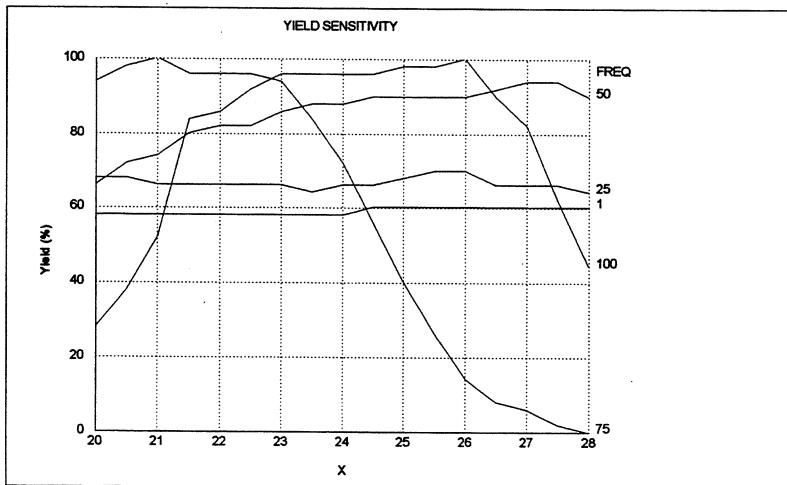


Fig. 12.10 Yield versus two sweep labels.

You can use Fig. 12.10 to analyze how the different combinations of X and FREQ values affect the yield. For instance, it shows that for X=24, the most critical FREQ value is 1, but for X=25, the most critical FREQ value is 75.

## Multiple Sweep Sets

If the MonteCarlo block contains more than one sweep set, then the yield sensitivity feature applies to one sweep set at a time. In other words, one of the sweep sets is selected and the marginal yield is based on the output labels from that set only.

## 12.23 Display of Maximum Errors

The “Max Error” option under the “MonteCarlo” menu displays either a histogram or a run chart of the maximum errors of individual outcomes.

In many cases, more than one design specification is involved in the Monte Carlo analysis. An outcome has to satisfy a number of specifications on different responses simultaneously for it to be considered as acceptable. In such cases, the display of any one response shows only a subset of the specifications, which does not represent the overall yield.

The display of the maximum errors is devised to depict a composite picture of the percentage of acceptable outcomes (i.e., yield) and the distribution of outcomes in terms of how much they satisfy or violate the specifications.

For each Monte Carlo outcome, a set of error functions  $e_i$ ,  $i = 1, 2, \dots, m$ , can be defined according to the specifications, where  $i$  identifies different responses and different sweep points, and  $m$  is the total number of error functions.

The maximum error for a Monte Carlo outcome is given by

$$e_{max} = \max \{ e_i \}.$$

The sign of  $e_{max}$  indicates the acceptability of the outcome:  $e_{max} > 0$  for an unacceptable outcome (one or more specifications violated) or  $e_{max} \leq 0$  for an acceptable outcome.

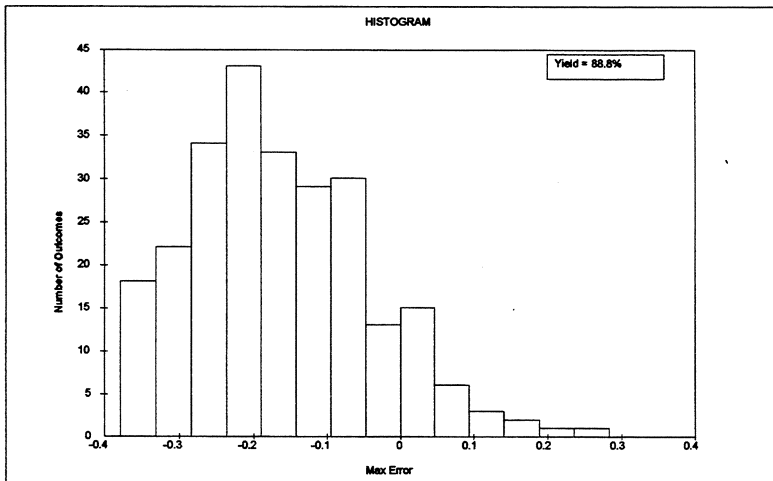


Fig. 12.11 A histogram of the maximum errors of individual outcomes.

## Display Format

This options allows you to select whether you wish the output to be displayed as either a “Histogram” or a “Run Chart”. If you select “Run Chart” the “Number of bins” option will be disabled.

## Number of Bins

You can specify the number of intervals (bins) for dividing the values of the selected output label. Available choices are 5, 10, 15 and 20 bins.

## 12.24 Scatter Diagrams

The "Scatter Plot" option under the "MonteCarlo" menu displays the scatter diagrams between two output labels at a selected sweep point (see "Select Values of Sweep Labels", in Section 12.16). One of the output label is shown as the X-axis and the other output label as the Y-axis.

Scatter diagrams depict graphically the correlation between two statistical responses.

Figs. 12.12 and 12.13 illustrate scatter diagrams. In Fig. 12.12, the correlation between the two responses is weak (the dots scatter fairly evenly). In Fig. 12.13, the correlation between the two responses is strong (the dots form a distinctive pattern).

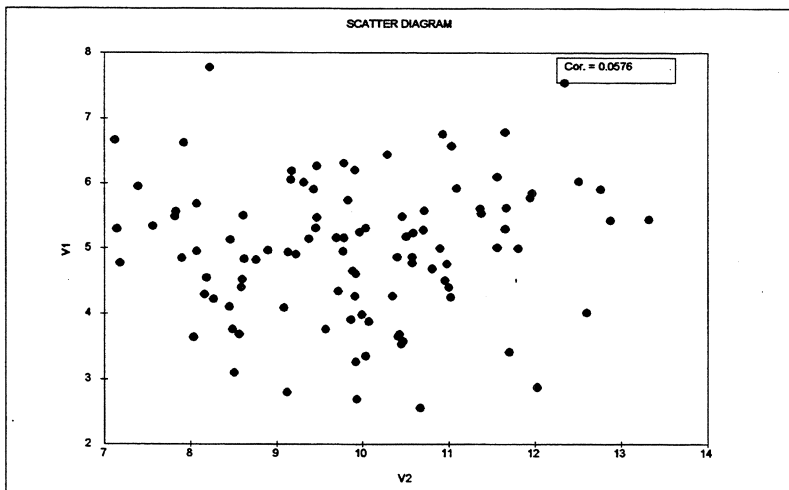
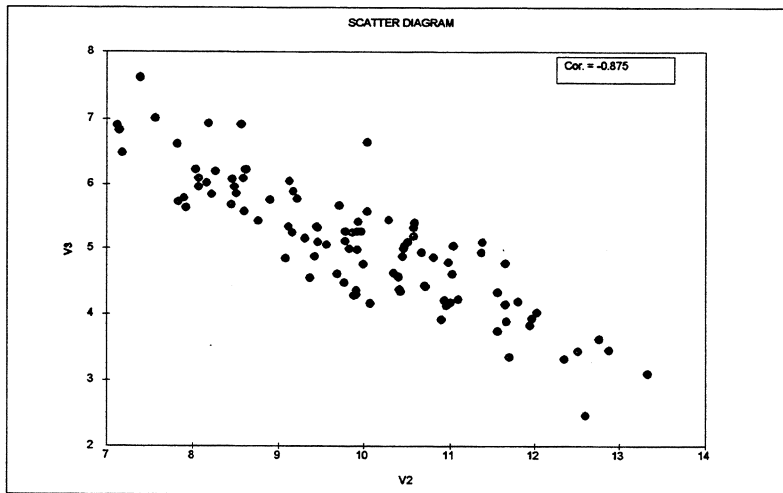


Fig. 12.12 Scatter diagram of two responses with weak correlation.



*Fig. 12.13 Scatter diagram of two responses with strong correlation.*

### Select Output Labels for the X-axis and Y-axis

From the output labels in the sweep set, any pair can be selected for the X-axis and Y-axis of the scatter diagram.

### Show Correlation Coefficient

The correlation coefficient between the selected responses can be estimated and displayed on the scatter diagram. You may find this information useful in some situations, but unnecessary in other situations. You can choose whether or not to show the estimated correlation coefficient through the "Show correlation coefficient" check box in the dialog box.



## 12.25 Views for Monte Carlo Displays

Graphical Views permit user-definable settings of colors, shapes, scales, etc., for graphical displays. Graphical Views for the “Display” menu options are discussed in Chapter 10.

This and the following sections describe Views for Monte Carlo displays. Views can be defined as a part of a sweep set in the MonteCarlo block.

### Syntax:

```
basic_sweep_set
{view_type view_definition}
...
{view_type view_definition};
```

where *basic\_sweep\_set* represents the basic definitions of a sweep set (sweep type, sweep labels, parameter labels and output labels), *view\_type* is a keyword to identify the view type and *view\_definition* defines the view.

TABLE 12.5 MONTE CARLO VIEW TYPES

Keyword	Type of Display
Xsweep	statistical responses versus parameter sweeps
Parametric	statistical parametric plots of responses
Histogram	histogram of individual statistical responses
Run_Chart	run chart of individual statistical responses
Scatter	scatter diagram between statistical responses

### Multiple Output Labels in One Monte Carlo Display

Using Views, you can include multiple output labels in a single Monte Carlo display. The advantage here is that you have precise control over the colors and shapes assigned to the different output labels, in order to make them more easily distinguishable.

## Monte Carlo Xsweep and Parametric Views

The formats for Monte Carlo Xsweep and Parametric Views are identical to those for the "Display" menu options as described in Chapter 10.

### Syntax:

```
{ XSWEEP Y=output_label1.color1.shape1 & output_label2.color2.shape2
      ... & output_labeln.colorn.shapen
      X=sweep_label      other_sweep_label=value
      TITLE=view_title  X_TITLE=x_title  Y_TITLE=y_title
      XMIN=xmin          XMAX=xmax       YMIN=ymin
      YMAX=ymax          NXTICKS=nxticks  NYTICKS=nyticks
      COMMENTS=comment1 & comment2 & ... commentm
      LEGEND=legend_pos }
```

*output\_label1*, ..., *output\_labeln* are a subset of the output labels in the sweep set, and *color1*, ..., *colorn*, *shape1*, ..., *shapen* specify the associated display attributes.

*sweep\_label* is one of the sweep labels in the sweep set, and if the sweep set contains two sweep labels, *other\_sweep\_label* is assigned *value* which is one of the available values defined in the sweep set or the keyword "ALL" for all the available values.

*view\_title*, *x\_title* and *y\_title* are character strings.

*xmin*, *xmax*, *ymin* and *ymax* are numerical constants defining the display scale.

*nxticks* and *nyticks* are integers specifying the number of ticks (grids) on the axes.

*comment1*, ..., *commentm* are character strings.

*legend\_pos* can be ON, OFF, TOP or INSIDE.

All the entries are optional.

### Syntax:

```
{ PARAMETRIC P=parametric_label  other_sweep_label=value
      X=x_axis_label
      Y=output_label1.color1.shape1 &
        output_label2.color2.shape2 & ...
        output_labeln.colorn.shapen
      TITLE=view_title  X_TITLE=x_title  Y_TITLE=y_title
      XMIN=xmin          XMAX=xmax       YMIN=ymin
      YMAX=ymax          NXTICKS=nxticks  NYTICKS=nyticks
      COMMENTS=comment1 & comment2 & ... commentm
      LEGEND=legend_pos }
```

*parametric\_label* must be specified as one of the sweep labels in the sweep set.

*x\_axis\_label* must be one of the output labels in the sweep set.

The definitions of all the other entries are identical to those of Xsweep View.

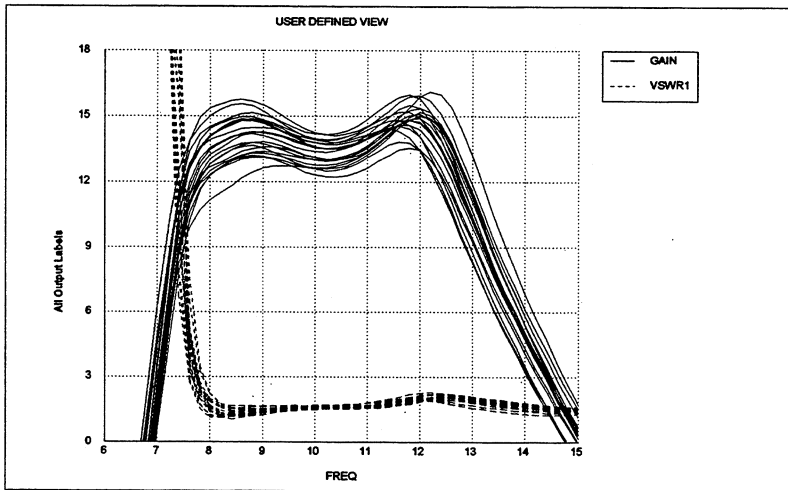
**Example:**

```

AC:  FREQ: from 6GHZ to 15GHZ step=0.2GHZ  N_OUTCOMES=20
    GAIN  VSWR1
    { Xsweep  X=FREQ  Nxticks=9
      Y=GAIN & VSWR1  Ymin=0  Ymax=18  NYTicks=6 };

```


The View is illustrated in Fig. 12.14.



*Fig. 12.14 Illustration of Monte Carlo Xsweep Views.*

## Showing Specifications in Views

Using Views, you can conveniently superimpose specifications on graphical displays.

 Please see Chapter 10 for detailed descriptions.

## 12.26 Histogram Views

### Syntax:

```
{ HISTOGRAM X=output_label1.color1.shape1 &
           output_label2.color2.shape2 & ...
           output_labeln.colorn.shapen
           sweep_label=value sweep_label=value
           TITLE=view_title X TITLE=x_title Y TITLE=y_title
           XMIN=xmin XMAX=xmax YMIN=ymin
           YMAX=ymax NXTICKS=nxticks NYTICKS=nyticks
           LOW=low HIGH=high N_BINS=n_bins
           COMMENTS=comment1 & comment2 & ... commentm
           LEGEND=legend_pos }
```

*output\_label1*, ..., *output\_labeln* are a subset of the output labels in the sweep set, and *color1*, ..., *colorn*, *shape1*, ..., *shapen* specify the associated display attributes.

For each *sweep\_label* in the sweep set, a *value* is selected from the available values defined in the sweep set.

*view\_title*, *x\_title* and *y\_title* are character strings.

*xmin*, *xmax*, *ymin* and *ymax* are numerical constants defining the display scale.

*low* and *high* are numerical constants defining the histogram truncation limits.

*n\_bins* is an integer specifying the number of histogram bins.

*nxticks* and *nyticks* are integers specifying the number of ticks (grids) on the axes.

*comment1*, ..., *commentm* are character strings.

*legend\_pos* can be ON, OFF, TOP or INSIDE.

All the entries are optional.

Many of the Histogram View options, such as the number of bins and low/high truncation limits, have the same meaning as the options in the "Histogram Display Options" dialog box, as described in Section 12.19.



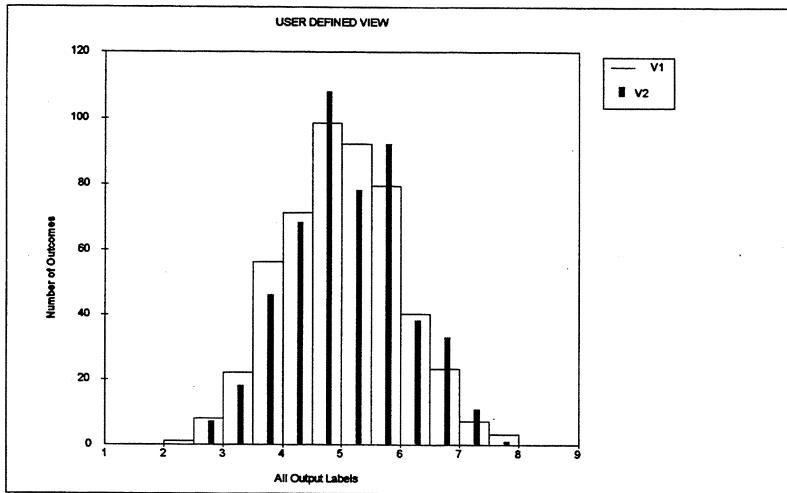
For histograms, output labels are specified for the X-axis. The Y-axis represents the number of outcomes in each of the bins.

One of the advantages of Histogram View is that you can include multiple output labels (responses) in a single histogram display. To clearly identify the different responses, they can be assigned distinct colors and/or shapes in the View definition.

Example:

```
{ HISTOGRAM X=V1 & V2.bar
           Xmin=1 Xmax=9 NXTicks=8 Low=2 High=8 N_BINS=12 }
```

This View includes two output labels in one display, as illustrated in Fig. 12.15. The histogram of the output label V2 is to be displayed as bars. The output label V1 is not assigned a specific shape, so its histogram will be displayed as bins by default.



*Fig. 12.15 Histogram View showing two output labels in different shapes.*

## Legends in Histogram Views

The contents of the legend box depend on the number of output labels in the View.

If there is only one output label in the View, the legend box shows the estimated mean and standard deviation.

When multiple output labels are included in the View, the legend box shows the colors and shapes assigned to each output label.

## 12.27 Run Chart Views

### Syntax:

```
{ RUN_CHART Y=output_label1.color1.shape1 &
            output_label2.color2.shape2 & ...
            output_labeln.colorn.shapen
            sweep_label=value sweep_label=value
            TITLE=view_title X_TITLE=x_title Y_TITLE=y_title
            XMIN=xmin XMAX=xmax YMIN=ymin
            YMAX=ymax NXTICKS=nxticks NYTICKS=nyticks
            COMMENTS=comment1 & comment2 & ... commentm
            LEGEND=legend_pos }
```

*output\_label1*, ..., *output\_labeln* are a subset of the output labels in the sweep set, and *color1*, ..., *colorn*, *shape1*, ..., *shapen* specify the associated display attributes.

For each *sweep\_label* in the sweep set, a *value* is selected from the available values defined in the sweep set.

*view\_title*, *x\_title* and *y\_title* are character strings.

*xmin*, *xmax*, *ymin* and *ymax* are numerical constants defining the display scale.

*nxticks* and *nyticks* are integers specifying the number of ticks (grids) on the axes.

*comment1*, ..., *commentm* are character strings.

*legend\_pos* can be ON, OFF, TOP or INSIDE.

All the entries are optional.

Using Run\_Chart Views you can include multiple output labels (responses) in a single display. To clearly identify the different responses, they can be assigned distinct colors and/or shapes in the View definition.

### Example:

```
{ RUN_CHART Y=V1.circle & V2.square
            NXTicks=5 Ymin=2 Ymax=8 NYTicks=6 }
```

This View includes two output labels in one display, as illustrated in Fig. 12.16. The output label V1 is displayed as circles and the output label V2 as squares.

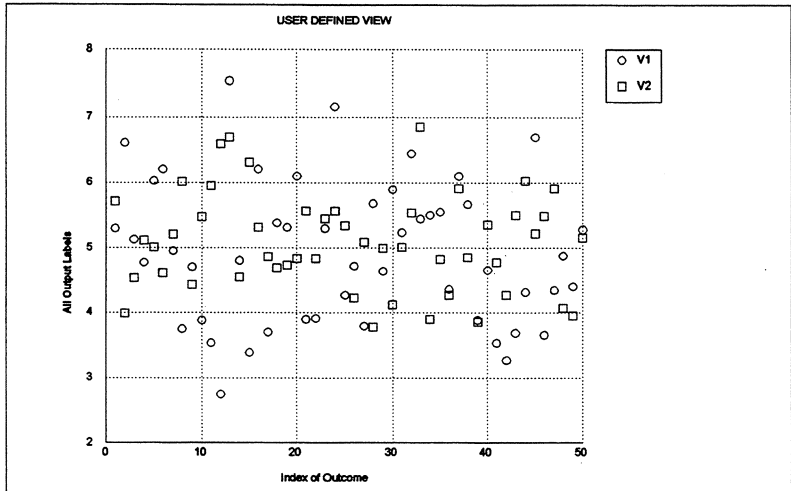


Fig. 12.16 Run Chart View showing two output labels in different shapes.

## 12.28 Scatter Views

### Syntax:

```
{ SCATTER X=x_axis_label
          Y=output_label1.color1.shape1 &
            output_label2.color2.shape2 & ...
            output_labeln.colorn.shapen
          sweep_label=value sweep_label=value
          TITLE=view_title X TITLE=x_title Y TITLE=y_title
          XMIN=xmin XMAX=xmax YMIN=ymin
          YMAX=ymax NXTICKS=nxticks NYTICKS=nyticks
          COMMENTS=comment1 & comment2 & ... commentm
          LEGEND=legend_pos }
```

*x\_axis\_label* must be one of the output labels in the sweep set.

*output\_label1*, ..., *output\_labeln* are a subset of the output labels in the sweep set, and *color1*, ..., *colorn*, *shape1*, ..., *shapen* specify the associated display attributes.

For each *sweep\_label* in the sweep set, a *value* is selected from the available values defined in the sweep set.

*view\_title*, *x\_title* and *y\_title* are character strings.

*xmin*, *xmax*, *ymin* and *ymax* are numerical constants defining the display scale.

*nxticks* and *nyticks* are integers specifying the number of ticks (grids) on the axes.

*comment1*, ..., *commentm* are character strings.

*legend\_pos* can be ON, OFF, TOP or INSIDE.

All the entries are optional.

A scatter diagram displays one output label as the X-axis and one or more output labels as the Y-axis. It depicts graphically the correlation between the output labels.

When multiple output labels are specified for the Y-axis in a Scatter View, they can be assigned distinct colors and/or shapes for clear identification.

Example:

```
{ SCATTER X=V1 Y=V2.x & V3.triangle
          Xmin=3 Xmax=8 NXTicks=5 Ymin=2 Ymax=8 NYTicks=6 }
```

This View displays two output labels, namely V2 and V3, versus the output label V1. As illustrated in Fig. 12.17, V3 (shown as triangles) is strongly correlated to V1 whereas V2 (shown as crosses) and V1 are not correlated.



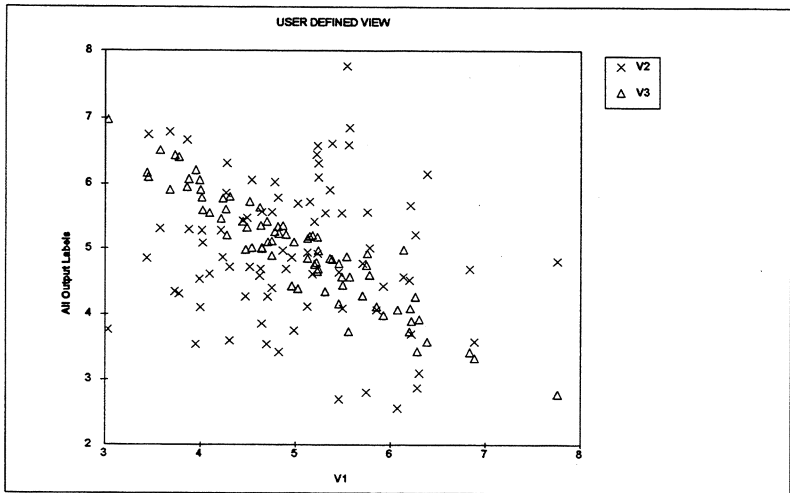


Fig. 12.17 Scatter View showing two output labels in different shapes.



# 13

## Yield Optimization

<b>13.1 Overview</b> .....	13-1
<b>13.2 Yield Optimization Options</b> .....	13-2
<b>13.3 Restart Yield Optimization</b> .....	13-7
<b>13.4 Technical References</b> .....	13-8



## 13

## Yield Optimization

## 13.1 Overview

The statistical yield optimization feature in OSA90 is based on OSA's pioneering research work. It employs a sophisticated and powerful gradient-based, multi-outcome generalized  $\ell_1$  algorithm to optimize circuits in the presence of parameter tolerances, input fluctuations and other statistical uncertainties.

The statistical models that can be used include uniform, normal (Gaussian), lognormal, exponential and correlated multidimensional normal distributions. The syntax of defining statistical parameters is described in Chapter 12.

The discrete yield estimated from a given number of statistical outcomes is approximated by a continuous and smooth generalized  $\ell_1$  objective function suitable for gradient-based optimization. Specially developed optimizers are dedicated to yield optimization.

A unique quadratic modeling technique is offered for yield optimization. It constructs quadratic models of the error functions and derivatives. During yield optimization, the quadratic model is utilized to generate the error functions and derivatives for the Monte Carlo outcomes. For a large number of outcomes, this can lead to a substantial reduction in computation time in comparison with exact analyses. A few key technical references are given at the end of this chapter.

The yield optimization feature can be applied to built-in circuit responses, generic functions defined by expressions and external simulators connected through Datapipe.

Setting up a yield optimization problem involves the following steps:

- ▶ Define a set of optimization variables in the Model or Expression block of the input file (see Chapter 4).
- ▶ Define a set of statistical parameters in the Model or Expression block (see Chapter 12). The set of statistical parameters may overlap with the set of optimization variables. For instance, the nominal value of a statistical parameter can be defined as an optimization variable.
- ▶ Define the simulation types, sweep ranges and specifications in the Specification block (the Sweep block is for simulation, the MonteCarlo block for statistical analysis, and the Specification block for nominal and yield optimization).
- ▶ Select "Yield" from the "Optimize" menu.

## 13.2 Yield Optimization Options

To start yield optimization, select the "Yield" option under the "Optimize" menu. A dialog box will appear.

### Select Objective Function

The yield estimated by Monte Carlo simulation is a discrete function (discontinuous with respect to the optimization variables) and thus unsuitable for gradient-based optimization.

To overcome this difficulty, OSA developed a sophisticated generalized  $\ell_1$  centering approach. The generalized  $\ell_1$  objective function is formulated from the error functions calculated at a number of random outcomes generated according to the statistical parameter distributions. A gradient-based one-sided  $\ell_1$  optimizer is specially developed for the minimization of this objective function. A set of weights is carefully chosen such that the optimization will lead to a centered solution and an enhanced yield[1].

If you select "Generalized L1" objective function then the solution presented by the optimizer will correspond to the lowest value of the generalized  $\ell_1$  objective function. However if you select "Yield" objective function, the algorithm, in addition to minimizing the generalized  $\ell_1$  objective function, will also check the corresponding values of yield. Thus, by selecting "Yield" you will be presented with a solution corresponding the maximum value of yield found by the optimizer.

You can also designate your choice of the objective function in the Control block.

#### Syntax:

```
Control
  Objective_Function = name;
End
```

*name* can be Yield or GL1.

## Number of Iterations

This option in the dialog box allows you to limit the maximum number of iterations. The optimizer will stop once this limit is reached even if a solution within the desired accuracy has not yet been found. If this happens, the best set of values of the variables (corresponding to the highest yield estimate) is presented as the solution.

You can also designate your choice in the Control block.

### Syntax:

```
Control
  N_Iterations = n;
End
```

where  $n$  is one of the choices: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 30, 50, 100, 999, 9999.

Other values of  $n$  will be rounded to the closest choice which is greater than  $n$ , e.g., 45 will be rounded to 50. If  $n > 9999$ , it will be set to 9999.

## Number of Statistical Outcomes

The "Number of outcomes" option allows you to select the number of statistical (random) outcomes for yield optimization. The outcomes are used in the Monte Carlo simulation to estimate yield during each iteration of the optimization.

The yield is estimated by

$$Yield = N_{pass} / N$$

where  $N$  represents the number of outcomes you have selected, and  $N_{pass}$  represents the number of "passed" outcomes, i.e., outcomes which satisfy all the specifications.

The choices for the number of statistical outcomes are 10, 20, 50, 100, 200 and 500. The default is 20 outcomes.

A larger number of outcomes will better represent the statistical parameter distributions and lead to a more accurate estimate of the yield, but it will also demand longer computer time for the Monte Carlo simulation and yield optimization.

Typically, for a new problem, you may start with a choice of 20 or 50 outcomes. At the solution, you can increase the number of outcomes (e.g., to 100 or 200 outcomes) and restart the yield optimization in order to obtain a more accurate result (see also Section 13.3).

You can also designate your choice in the Control block.

**Syntax:**

```
Control
  N_Yield_Outcomes = n;
End
```

where  $n$  is one of the choices: 10, 20, 50, 100, 200, 500.

Other values of  $n$  will be rounded to the closest choice which is greater than  $n$ , e.g., 80 will be rounded to 100. If  $n > 500$ , it will be set to 500.

**Accuracy of Solution**

This option allows you to specify the desired accuracy of the solution. The optimization will stop when the step size separating the current point and the next point is smaller than the specified accuracy. The step size is relative to the norm of the variable vector.

You can also designate your choice in the Control block.

**Syntax:**

```
Control
  Accuracy = v;
End
```

where  $v$  is one of the choices: 0.01, 0.001, 1.0e-4, 1.0e-5, 1.0e-6.

Other values of  $v$  will be rounded to the closest choice which is smaller than  $v$ , e.g., 0.003 will be rounded to 0.001. If  $v < 1.0e-6$ , it will be set to 1.0e-6.

**Show Downhill Iterations Only**

During the yield optimization, OSA90 displays the iteration count, the current yield estimate and the value of the generalized  $l_1$  objective function. For example:

```
Iteration  1/30  Yield= 60%  Objective=8
Iteration  2/30  Yield= 65%  Objective=5.82925
Iteration  3/30  Yield= 65%  Objective=5.6665
...
```

Of the iteration count, the first number is the current iteration and the second one is the maximum number of iterations specified. For example, 12/30 means that the current iteration is the 12th out of a maximum of 30.



Sometimes you may notice the estimated yield and the objective function value fluctuate between the iterations. This is not unusual. It is a part of the optimization algorithm and the solution will always have the highest estimated yield.

By default, this information is displayed at every iteration. You can instruct the program to display only those iterations that lead to a higher estimated yield than the previous ones. You can do this by placing a check mark in the "Show downhill iterations only" check box.

You can also make your request in the Control block.

**Syntax:**

```
Control
    Print_Best_Iterations;
End
```

## Quadratic Modeling of Functions and Gradients

The option "Q-gradient modeling" in the dialog box represents the technique of quadratic modeling of functions and gradients. The theoretical details of this technique are described in the reference [2] listed at the end of this chapter.

By checking or unchecking the "Q-gradient modeling" option, you enable or disable the use of quadratic models in the yield optimization.

If enabled, a quadratic model will be constructed from the error functions and derivatives computed at a number of base points. The quadratic model is then used to generate (modelled) error functions and derivatives at each statistical outcome, replacing the exact calculations otherwise required.

Enabling the "Q-gradient modeling" option usually reduces the computational time needed for yield optimization. The effort involved in constructing the quadratic model depends on the number of statistical parameters. Once the model is established, using it to generate error functions and derivatives takes relatively little effort. The computational time required by quadratic modeling does not increase significantly with the number of outcomes. Therefore, a large number of outcomes (100 or 200) is recommended if you choose to use quadratic modeling.


You can also disable the "Q-gradient model" feature in the Control block.

**Syntax:**

```
Control
    No_Q_Model;
End
```

## Interrupt the Yield Optimization

During the yield optimization, OSA90 displays a progress bar on top of the status bar, located at the bottom of the OSA90 window.

While the optimization is in progress, you can interrupt it at any time by clicking on the “Stop” button  located on the toolbar. This will terminate the optimization without prompting for confirmation.

If you interrupt and terminate the optimization in progress, then the best set of values of the optimization variables obtained up to that point will be presented as the solution.

## Termination of Yield Optimization

Yield optimization is terminated under one of the following conditions:

- ▶ Interrupted and terminated by the user.
- ▶ A local minimum of the generalized  $\ell_1$  objective function is found, i.e., the value of the objective function cannot be further reduced in the vicinity of the solution.
- ▶ The specified accuracy is reached, i.e., the step size separating the current point and the next point suggested by the algorithm is smaller than the specified accuracy. The step size is relative to the norm of the variable vector.
- ▶ The maximum number of optimization iterations is reached.
- ▶ A severe error condition is encountered such as a floating-point divide by zero error.

A message indicating the cause of termination will be displayed. Regardless of the cause the solution is always the best set of values of the variables obtained up to the point of termination.

The input file is automatically updated with the solution, i.e., the variables are updated with their optimized values.



The input file that is updated with the solution is actually the copy kept by OSA90 in the computer's memory. The original disk file is not changed. If you wish to keep the changes, be sure to save the updated file to the disk.

## 13.3 Restart Yield Optimization

Restarting yield optimization (after a solution is obtained) can be advantageous in several situations.

If the number of outcomes used is relatively small, the solution may be unduly biased. This can be verified by a Monte Carlo analysis as follows.

- ▶ Define the same sweep ranges and specifications in the MonteCarlo block as those in the Specification block (i.e., those used in the yield optimization).
- ▶ Specify in the MonteCarlo block a larger number of outcomes than that used in the yield optimization (e.g., you may specify 500 or 1000 outcomes in the Monte Carlo analysis for verifying a solution optimized using 20 outcomes).
- ▶ Select “Yield” from the “MonteCarlo” menu to display the yield estimated by the Monte Carlo analysis. If there is a significant discrepancy between the yield estimated by the Monte Carlo analysis and the yield estimated by the optimization (which is based on a smaller number of outcomes), then the solution is not very accurate and you should consider restarting yield optimization with a larger number of outcomes.

A solution obtained using quadratic modeling should also be verified by the Monte Carlo analysis. It is not unexpected to find the yield estimated by the Monte Carlo analysis different from the yield predicted by the quadratic model, because the quadratic model is an approximation to the exact functions. If necessary, you can restart yield optimization with the quadratic modeling feature disabled. But at least the quadratic modeling method can be used to produce a good starting point.

The generalized  $\ell_1$  centering algorithm in OSA90 includes certain parameters whose values are initialized at the beginning of the yield optimization. These values are not adjusted during optimization in order to maintain continuity and stability. Restarting the yield optimization will provide an opportunity for the algorithm to reset those parameters, which may further improve the solution in some cases.

### Recommended Strategy

- 1 Start with a small number of outcomes (20 or 50), or enable the “Q-gradient modeling” option.
- 2 Perform yield optimization to obtain a solution.
- 3 Verify the solution by the Monte Carlo analysis using a sufficiently large number of outcomes.
- 4 Restart yield optimization (go to step 2) if necessary after increasing the number of outcomes or disabling the “Q-gradient modeling” option.

## 13.4 Technical References

- [1] J.W. Bandler and S.H. Chen, "Circuit optimization: the state of the art", *IEEE Trans. Microwave Theory Tech.*, vol. 36, pp. 424-443, 1988.
- [2] J.W. Bandler, R.M. Biernacki, S.H. Chen, J. Song, S. Ye and Q.J. Zhang, "Gradient quadratic approximation scheme for yield-driven design", *IEEE MTT-S Int. Microwave Symp. Digest* (Boston, MA), pp. 1197-1200, 1991.
- [3] J.W. Bandler, Q.J. Zhang, J. Song and R.M. Biernacki, "FAST gradient based yield optimization of nonlinear circuits", *IEEE Trans. Microwave Theory Tech.*, vol. 38, pp. 1701-1709, 1990.

14

## OSA90 as Datapipe Child



## 14

# OSA90 as Datapipe Child

## Introduction

Datapipe is an innovative feature of OSA90 for establishing functional connections with external simulators (see Chapter 5).

OSA90 itself can be used as a Datapipe child (i.e., as an "external" simulator). In other words, from within OSA90 you can invoke another copy of OSA90 via Datapipe.

This opens up new opportunities for creating simulation/optimization hierarchies of virtually unlimited depth. For instance, you can extend parameter sweeps beyond two-dimensional sweeps. Or, you can have optimization within a simulation loop.



If you are not yet familiar with the general Datapipe concept and syntax, please read Chapter 5 first.



Examples of using OSA90 as Datapipe child can be found in demo57, demo58, demo61 and demo62.

## File Syntax in the Parent OSA90

### Syntax:

```
Datapipe: COM FILE="osa90 -p"
           N_INPUT=n INPUT=(flag, input_file, ...)
           ... ;
```

where *flag* is an integer specifying the type of connection as listed in Table 14.1, and *input\_file* is a string label representing the input file for the child OSA90.

- ▷ The Datapipe definition can be placed in either the Expression block or the Model block of the input file of the parent OSA90.
- ▷ The COM protocol for Datapipe must be used to allow passing an input file to the child OSA90 as a string label.
- ▷ Following the keyword FILE, the child program is specified as "osa90 -p", where the run-time option "-p" signals OSA90 to operate in the Datapipe child mode.
- ▷ The first Datapipe input argument is an integer flag. The child OSA90 interprets this flag to determine what operations are to be performed. The valid choices for this flag are listed in Table 14.1.

TABLE 14.1 FLAGS FOR THE CHILD OSA90

Flag	Operations Performed by the Child OSA90
1	simulation and returning the output labels in the Sweep block
2	repeated simulation and returning the output labels in the Sweep block
3	statistical simulation and returning the output labels in the MonteCarlo block

### Child OSA90 as a Simulator

#### Syntax

```
Datapipe: COM FILE="osa90 -p"
           N_INPUT=2 INPUT=(1, input_file)
           N_OUTPUT=m OUTPUT=(sweep_results);
```

The first input must be an integer flag set to the value of 1.  
*input\_file* must be a string label representing the input file for the child OSA90.

This is the simplest form of using OSA90 as a Datapipe child.

Two arguments are passed to the child. The first one is a flag with the value of 1. The second input argument is expected to be the text of a valid input file.

The following operations are carried out by the child OSA90:

- 1 The input file is parsed and if an error is detected the appropriate error message is returned to the parent OSA90. Although the error message will be displayed by the parent OSA90, the location of the error within the input file cannot be pinpointed. Therefore, you should make sure that the input file syntax is correct before passing it to the child OSA90 (by having the file parsed by the parent OSA90 first).
- 2 Simulation is performed according to the Sweep block of the input file received by the child.
- 3 All the output labels defined in the Sweep block are evaluated at all the sweep points and the results returned to the parent.



Example:

```

Expression
  char InputFile[] = "
    Model
      ...
      ...
    end

    Sweep
      AC: FREQ: from 1 to 10 step=1 MS21;
    end
";

Datapipe: COM FILE="osa90 -p"
           N_INPUT=2 INPUT=(1, InputFile)
           N_OUTPUT=10 OUTPUT=(Gain[10]);
...

```

In this example, the child OSA90 will simulate the circuit defined in the Model block in the input file represented by the string label InputFile and returns the responses defined in the Sweep block, i.e., the values of MS21 at 10 frequency points. The parent OSA90 stores the results in the array Gain.

### Order of the Results Returned from the Child OSA90

The child OSA90 returns the simulation results in the order of appearance of the output labels in the Sweep block of the child input file.

The values of the first output label in the first sweep set at all the sweep points in that sweep set are returned first, followed by the values of the second output label of the first sweep set, and so on. If there are double sweep parameters, the sequence of inner and outer sweeps is explained in Chapter 6. If in doubt, you can store all the results in a single array and then display its contents numerically to help you understand the order.

You can store the results in the parent OSA90 with arbitrary partitions. In other words, it is not necessary to have one Datapipe output array for each output label defined in the child's Sweep block. You can re-organize the results in any way you see fit.

## Label Substitution in the Child OSA90

**Syntax:**

```
Datapipe: COM FILE="osa90 -p"
           N_INPUT=n INPUT=(1, input_file, x1, ..., xk)
           N_OUTPUT=m OUTPUT=(sweep_results);
```

The first input must be an integer flag set to the value of 1.  
*input\_file* must be a string label representing the input file for the child OSA90.  
*x1*, ..., *xk* are values to be assigned to the first *k* labels in *input\_file*.

In this case, more than two arguments are passed to the child OSA90.

The first input is the flag, the second the input file for the child OSA90, and the remaining input arguments are numerical values for label substitution.

The values are assigned to the first *k* labels in the child's input file which are defined as either constant labels or optimizable labels.

**Example:**

```
Expression
  char InputFile[] = "
    Model
      X1: ?1.5?;
      X2: ?-0.3?;
      CGS: 10pF;
      ...
    end
  ";

Vars[3] = [?2.0177? ?-1.46? ?6.43pF?];

Datapipe: COM FILE="osa90 -p"
           N_INPUT=5 INPUT=(2, InputFile, Vars)
           ...;
...

```

The values of Vars[1], Vars[2] and Vars[3] in the parent OSA90 are assigned to the labels X1, X2 and CGS in the child OSA90, respectively.



If input file of the child OSA90 contains both the Expression and Model blocks, then the labels defined in the Expression block are parsed first.

## Repeated Simulation in the Child OSA90

### Syntax

```
Datapipe: COM FILE="osa90 -p"
             N_INPUT=n INPUT=(2, input_file, x1, ..., xk)
             N_OUTPUT=m OUTPUT=(sweep_results);
```

The first input must be an integer flag set to the value of 2.  
*input\_file* must be a string label representing the input file for the child OSA90.  
 Optionally, *x1*, ..., *xk* are one or more values for label substitution.

Often, the child OSA90 is called repeatedly within a simulation or optimization iteration of the parent OSA90. In such cases, you can set the flag (the first Datapipe input argument) to the value of 2, which instructs the child to avoid parsing the input file text repeatedly (i.e., the child will parse the file only once at the first call).

You should not use this method if more than one Datapipe connection to the child OSA90 is opened simultaneously and with different input files.

### Example:

```
char InputFile1[] = " ... ";

Datapipe: COM FILE="osa90 -p"
             N_INPUT=... INPUT=(..., InputFile1, ...)
             ...;

char InputFile2[] = " ... ";

Datapipe: COM FILE="osa90 -p"
             N_INPUT=... INPUT=(..., InputFile2, ...)
             ...;
```

## Monte Carlo Simulation in the Child OSA90

**Syntax:**

```
Datapipe: COM FILE="osa90 -p"
            N_INPUT=n INPUT=(3, input_file, x1, ..., xk)
            N_OUTPUT=m OUTPUT=(sweep_results);
```

The first input must be an integer flag set to the value of 3.  
*input\_file* must be a string label representing the input file for the child OSA90.  
 Optionally, *x1*, ..., *xk* are one or more values for label substitution.

When the flag (the first Datapipe input) is set to 3, the child OSA90 will perform statistical simulation according to the MonteCarlo block.

The following operations are involved:

- 1 The input file is parsed and if an error is detected the appropriate error message is returned to the parent OSA90. Although the error message will be displayed by the parent OSA90, the location of the error within the input file cannot be pinpointed. Therefore, you should make sure that the input file syntax is correct before passing it to the child OSA90 (by having the file parsed by the parent OSA90 first).
- 2 Statistical simulation is performed according to the MonteCarlo block of the input file received by the child.
- 3 All the output labels defined in the MonteCarlo block are evaluated at all the sweep points and all the outcomes. The results returned to the parent.

The child OSA90 returns the statistical simulation results in the order of appearance of the output labels in the MonteCarlo block of the child input file.

Suppose that in the MonteCarlo block there are  $n$  output labels,  $m$  sweep points and  $L$  outcomes.  $Y_{ijk}$  denotes the simulation result for the  $i$ th label at the  $j$ th outcome and the  $k$ th sweep point. The order of the results returned is as follows:

$$\begin{array}{cccccccccccc} Y_{111} & Y_{112} & \dots & Y_{11m} & Y_{121} & Y_{122} & \dots & Y_{12m} & \dots & Y_{1L1} & Y_{1L2} & \dots & Y_{1Lm} \\ Y_{211} & Y_{212} & \dots & Y_{21m} & Y_{221} & Y_{222} & \dots & Y_{22m} & \dots & Y_{2L1} & Y_{2L2} & \dots & Y_{2Lm} \\ \dots & & & & & & & & & & & & \end{array}$$

$$Y_{n11} \quad Y_{n12} \quad \dots \quad Y_{n1m} \quad Y_{n21} \quad Y_{n22} \quad \dots \quad Y_{n2m} \quad \dots \quad Y_{nL1} \quad Y_{nL2} \quad \dots \quad Y_{nLm}$$

## Child OSA90 in Interactive Mode

You can specify the child OSA90 to run interactively using the run-time option "-pi".

**Syntax:**

```
Datapipe: COM FILE="osa90 -pi"  
          ...;
```

The parent and child OSA90 will both run interactively in separate windows.



You must not manually modify the problem set-up in the child OSA90, particularly the output labels in the Sweep or the MonteCarlo block of the child's Input File.



# 15

## Auxiliary Files

15.1 User's Manual Files .....	15-1
15.2 Log Files .....	15-2





# 15

## Auxiliary Files

### 15.1 User's Manual Files

The User's Manual files are contained in the `osa90msg` subdirectory within the OSA Installation Directory (see Chapter 1). The User's Manual is stored in the `**.*nt` and `**.*lp` files.

You can view the User's Manual on-line by pressing `<F1>` from within the OSA90 window, or by selecting "User's Manual" menu option from the "File" menu, or by clicking on the "Help" menu.

## 15.2 Log Files

Each time you run OSA90, a log file is created which chronologically records the major operations of the program. The messages generated by the program, including any error messages, are recorded in the log file. If optimization is performed, the progress of the optimization is also recorded.

Log files are ASCII text files. They are given the name "osa90.log" and are stored in the initial working directory. If you start OSA90 in a directory where an old "osa90.log" file exists then the old file will be overwritten. However, in the case of multiple copies of OSA90 run concurrently from the same initial working directory the file may be interlaced.

Example:

```
OSA90 V4.0 for Windows NT/95 Workstations
(c) Optimization Systems Associates Inc. 1997
Tue Jun 03 12:18:18 1997
12:18:18 Read in File C:\OSA\osa90_examples\MY_FILE1.ckt
12:18:22 Parsing Input File ...
12:18:22 File Parsing Completed
12:18:27 Simulation ... Press any key to interrupt
12:18:28 Ready for Display
12:18:48 Read in File C:\OSA\osa90_examples\MY_FILE2.ckt
12:18:58 Parsing Input File ...
12:18:58 File Parsing Completed
12:19:02 Minimax Optimization
12:19:02 Optimization ... Press any key to interrupt
12:19:03 Iteration 1/30 Max Error=1.35546
12:19:03 Iteration 2/30 Max Error=1.27626
.
.
.
12:19:04 Iteration 9/30 Max Error=-0.577254
12:19:04 Solution Max Error=-0.577254
12:19:04 Optimization completed
12:19:04 Elapsed real time: 00:00:01
12:19:05 Simulation ... Press any key to interrupt
12:19:06 Ready for Display
12:19:14 Save File C:\OSA\osa90_examples\MY_FILE3.ckt
Elapsed Time: 00:00:58
```

# 16

## Examples

<b>16.1 Overview</b> .....	16-1
<b>16.2 Example demo01</b> .....	16-8
<b>16.3 Example demo02</b> .....	16-11
<b>16.4 Example demo03</b> .....	16-14
<b>16.5 Example demo11</b> .....	16-19



# 16

## Examples

### 16.1 Overview

A set of demonstration examples is supplied with OSA90, as summarized in Tables 16.1 and 16.2.

The example files are named `demoxx.ckt`, where `xx` is an index. For those examples which involve optimization, the optimized solution files are named `demoxx_o.ckt`.

**TABLE 16.1 EXAMPLES CATEGORIZED BY APPLICATIONS**

Application	Examples					
Generic expressions and optimization	demo01 demo54	demo02 demo56	demo04 demo59	demo44 demo81	demo45	demo46
Matrices, vectors and interpolations	demo41 demo72	demo42 demo73	demo43	demo51	demo52	demo71
Datapipe connections	demo03 demo10	demo05 demo57	demo06 demo58	demo07	demo08	demo09
Nonlinear circuit harmonic balance simulation and optimization	demo11 demo39	demo14 demo40	demo21 demo47	demo22 demo49	demo23	demo38
Mixers and two-tone intermodulation	demo24	demo25	demo26	demo27	demo28	
Small-signal circuit simulation and design optimization	demo12 demo48	demo13 demo50	demo19 demo53	demo20 demo80	demo33	demo34
Monte Carlo analysis and yield optimization	demo10 demo65	demo35	demo36	demo37	demo55	demo64
Modeling and parameter extraction	demo14 demo32 demo78	demo15 demo63	demo16 demo70	demo17 demo75	demo18 demo76	demo31 demo77
Physics-based device simulation	demo29	demo30	demo48	demo60		
Oscillator analysis and design	demo61	demo62				
Visualization and contour plotting	demo66	demo67	demo68	demo69		

TABLE 16.2 DEMONSTRATION EXAMPLES

Example	Highlights
demo01	Simple expressions, parameter sweep, and parametric display of circles.
demo02	Simple optimization example of matching two functions, $l_1$ optimization.
demo03	Simulation and optimization of a transmission line transformer using an external simulator via Datapipe.
demo04	Minimax optimization of a generic mathematical problem.
demo05	Minimax optimization of the same problem as in demo04 but using an external program via Datapipe with the FUN protocol.
demo06	Time-domain transient analysis using an external simulator via Datapipe.
demo07	Solution of harmonic balance equations using $l_2$ optimization with external simulators via Datapipe.
demo08	Solution of waveform balance equations using $l_2$ optimization. The fact that the harmonic balance formulation in demo07 can be easily modified to the waveform balance formulation in this example demonstrates the flexibility of OSA90.
demo09	Minimax nominal optimization of a low-pass filter via Datapipe using the FDF protocol with gradients supplied by the external simulator.
demo10	Monte Carlo analysis and yield optimization of a low-pass filter using Datapipe, with quadratic modeling, starting from the solution of nominal optimization (demo09).
demo11	Harmonic balance simulation of a simple nonlinear FET circuit. The device is modeled by FETM extracted from <i>MGF1902</i> data.
demo12	Minimax optimization of a small-signal FET amplifier. The nonlinear FET model is extracted from <i>NEC700</i> data.
demo13	Minimax optimization of a small-signal balanced amplifier. This example is similar to demo12, but two FETs are used in a balanced structure with Lange couplers.
demo14	Harmonic balance simulation of a simple FET circuit for estimating the second- and third-order harmonic intercept points. Also illustrated is the preprogrammed Curtice cubic symmetrical FET model using FETU1.
demo15	This example is similar to demo14 but illustrates the preprogrammed Curtice cubic asymmetrical FET model using the user-definable model FETU2.
demo16	This example is adapted from demo12 to illustrate the preprogrammed Materka FET model using the user-definable model FETU2.

TABLE 16.2 DEMONSTRATION EXAMPLES (cont'd)

Example	Highlights
demo17	DC simulation of the preprogrammed Plessey FET model using the user-definable model FETU1. Also illustrated is how to display the match between simulated responses and measured data.
demo18	DC and small-signal AC simulation of the preprogrammed TOM FET model (TriQuint's Own Model) using the user-definable model FETU1. The match between simulated and measured $S$ parameters are displayed on Smith Chart and polar plot.
demo19	Minimax optimization of a distributed (traveling wave) amplifier with 4 FETs. The FETs are modeled by FETR extracted from <i>NEC700</i> data. The design specifications include small-signal gain and input/output VSWRs from 1GHz to 9GHz. Large-signal power gain and 1-dB compression point are also calculated by harmonic balance simulation.
demo20	Minimax optimization of a broadband 2-20GHz distributed (traveling wave) amplifier with 3FETs. The FETs are modeled by FETR extracted from <i>NEC700</i> data. The small-signal gain and return loss are optimized. The large-signal power gain and 1-dB compression at different frequencies are calculated by harmonic balance simulation.
demo21	Harmonic balance simulation of a balanced amplifier with multi-harmonic excitations. The circuit is similar to demo13, but the FETs are modeled by a linearized equivalent circuit about the DC operating point. The input contains multi-harmonic components. Harmonic distortions at the output are simulated for different fundamental frequencies. Also illustrated is the use of imported data representing linear subcircuits.
demo22	Large-signal harmonic balance optimization of an X-band power amplifier. The 2400 $\mu\text{m}$ FET are modeled by the Curtice cubic asymmetrical model. The gate bias voltage and matching circuits are optimized to increase output power and power-added efficiency.
demo23	Large-signal harmonic balance optimization of a Class-B X-band power amplifier. This example is similar to demo22 except that it is designed for Class-B instead of Class-A operation. The 2400 $\mu\text{m}$ FET are modeled by the Curtice cubic asymmetrical model. The gate is biased near the pinch-off voltage. The matching circuits are optimized for maximum power-added efficiency.
demo24	A resistively matched single-ended diode mixer. Nonlinear two-tone simulation with LO = 1.5GHz, RF = 1.57GHz and IF = 0.07GHz.
demo25	A singly-balanced diode mixer. Nonlinear two-tone simulation with LO = 3.75GHz, RF = 4GHz and IF = 0.25GHz. The conversion loss of the diode mixer is calculated versus sweep of the LO power.

TABLE 16.2 DEMONSTRATION EXAMPLES (cont'd)

Example	Highlights
demo26	Large-signal two-tone optimization of a broadband FET gate mixer. The FET model is extracted from <i>NEC700</i> data. The mixer has a fixed LO frequency of 7GHz and broadband IF frequency from 0.1 - 1.5GHz. The gate bias voltage and input matching circuit are optimized to increase the transducer conversion gain.
demo27	Nonlinear two-tone simulation of a narrow-band FET gate mixer. The circuit topology is similar to the broadband mixer in demo26. This example assumes a fixed IF frequency of 1GHz. The conversion gain is calculated versus sweeps of the LO power and gate bias voltage.
demo28	Two-tone intermodulation analysis of a simple nonlinear FET circuit. The FET is modeled by FETM extracted from <i>MGF1902</i> data. The third-order IM products are calculated versus sweep of the input power.
demo29	DC simulation using FETT (physics-based Trew FET model).
demo30	DC simulation using FETT1 (FETT modified for uniform doping).
demo31	Example of using nonlinear controlled sources for user-created models.
demo32	Example of FET parameter extraction by fitting simulated model responses to measured <i>S</i> parameters.
demo33	Minimax optimization of a microstrip line 3:1 impedance transformer.
demo34	Simulation of microstrip filter.
demo35	Monte Carlo simulation of a single-stage feedback amplifier. The FET is represented by a statistical model extracted from measurements. Tolerances are included for the microstrip matching circuits and feedback line.
demo36	Illustration of the yield sensitivity feature using a small-signal amplifier.
demo37	Yield optimization of a nonlinear FET frequency doubler.
demo38	Harmonic balance simulation of an amplifier with square wave excitation.
demo39	Harmonic balance simulation of an amplifier with triangular wave excitation.
demo40	Harmonic balance simulation of an amplifier with pulse train excitations of different duty cycles.
demo41	Illustration of the built-in cubic spline functions by an example of using cubic splines to interpolate a time-domain waveform response.



TABLE 16.2 DEMONSTRATION EXAMPLES (cont'd)

Example	Highlights
demo42	Illustration of the built-in 2D bicubic spline functions by an example of interpolating the DC drain current of a nonlinear FET circuit as a function of the gate and drain bias voltages.
demo43	Application of 2D bicubic spline interpolation to device modeling. A bias-dependent small-signal FET model is created using 2D bicubic splines.
demo44	Example of simple report generation.
demo45	Example of report generation for circuit simulation with parameter sweeps.
demo46	Report generation for displaying arrays and matrices in user-designed format.
demo47	DC and small-signal AC simulation with preprogrammed Raytheon-Statz FET model using FETU1.
demo48	DC and small-signal AC simulation of the physics-based bias-dependent small-signal FET model KTL.
demo49	Illustration of using a linear subcircuit as a port termination. A subcircuit consisting of a resistor and a capacitor in parallel is used as a complex load terminating the output port of a nonlinear FET circuit. Drain load lines at different input power levels and frequencies are shown.
demo50	Illustration of nested linear subcircuits to improve simulation efficiency.
demo51	Using the built-in matrix functions to define and solve the nodal equations of a linear resistive circuit.
demo52	Using the built-in matrix functions to define and solve the complex nodal equations of an RLC circuit.
demo53	Illustration of FET drain load lines of a small-signal amplifier.
demo54	Demonstration of Huber optimization. A sequence of Huber optimization with decreasing threshold values is shown to converge to the $\ell_1$ solution.
demo55	Monte Carlo analysis of a small-signal X-band amplifier. The FETs and the passive components are represented by physics-based models. The FET gate length, gate width, channel thickness and doping density as well as MIM capacitor metal plate areas and spiral inductor turns have been optimized for enhanced yield.
demo56	Testing the errors in inverting a nearly singular matrix. The Monte Carlo method is used to statistically perturb the matrix and the numerical errors at different perturbation levels are analyzed.

TABLE 16.2 DEMONSTRATION EXAMPLES (cont'd)

Example	Highlights
demo57	Illustration of using OSA90 as a child through Datapipe. A separate copy of OSA90 is called as a Datapipe child to compute the drain voltage and current of a nonlinear FET amplifier for displaying load lines.
demo58	Illustration of using OSA90 as a child through Datapipe. A separate copy of OSA90 is called as a Datapipe child to compute the large-signal power gain of a nonlinear FET circuit at different input power levels. The results are used by the parent OSA90 to calculate the 1-dB compression point versus frequency.
demo59	Illustration of using OSA90 for adjoint sensitivity calculations.
demo60	Implementation of a YIG resonator model published by J.A. Mezak and G.D. Vendelin.
demo61	Oscillator analysis using the OSCPORT element.
demo62	Oscillator design optimization using the OSCPORT element. The tuning capacitor of the FET oscillator is optimized to achieve free oscillator at the desired frequency of 15GHz.
demo63	Illustration of using the user-definable bipolar transistor model BJTU.
demo64	Illustration of Graphical Views for the Monte Carlo displays.
demo65	Using Monte Carlo Views to display the statistical responses of a small-signal X-band amplifier.
demo66	Illustration of minimax contour plotting with optimization traces.
demo67	Illustration of contour and trace plotting for minimax optimization with OSA90 as a Datapipe child.
demo68	3D visualization of the "rain drop" function.
demo69	Contours of the Rosenbrock function.
demo70	Illustration of using the Sample block and sample distribution.
demo71	Cubic spline interpolation applied to several functions.
demo72	Illustration of the array functions MEAN, StdDev, SKEW, KURT and CORR.
demo73	Illustration of using the function QR to compute eigenvalues and eigenvectors.
demo74	Illustration of displaying specifications in Graphical Views.

TABLE 16.2 DEMONSTRATION EXAMPLES (cont'd)

Example	Highlights
demo75	Using SPORT element to import data for parameter extraction.
demo76	Illustration of the element model CLIN.
demo77	Illustration of the element model CLINP.
demo78	Illustration of the element model MRSTUB.
demo79	Simulation and parameter extraction of the HEMT model HEMTAC. The parameter extraction involves DC data and $S$ parameters.
demo80	Importing $S$ -parameter files in EEsof format.
demo81	Simulated annealing optimization.

As a case study, four of the examples, namely demo01, demo02, demo03 and demo11 are analyzed in the following sections.

## 16.2 Example demo01

This example uses simple expressions to illustrate parameter sweeps, output labels and parametric displays.

### Input File demo01.ckt

```
! Example demo01.ckt
! a simple example for OSA90
! Xsweep displays X and Y as functions of Angle
! Parametric plots of circles

Expression
  Angle: 0;      ! sweep label
  Center: 0;    ! center of the circles

  X: cos(Angle) + Center;  ! X coordinates of points on a circle
  Y: sin(Angle) - Center;  ! Y coordinates
end

Sweep
  Center: 0 1 2 3 4 5
  Angle: from 0 to (2 * PI) N=100 X Y
  { Parametric P=Angle Center=ALL Title="PARAMETRIC VIEW OF CIRCLES"
    X=X X_title="cos(Angle) + Center" Xmin=-4 Xmax=8 NXticks=12
    Y=Y Y_title="sin(Angle) - Center" Ymin=-7 Ymax=2 NYticks=9 };
end
```

### Step by Step "Guided Tour"

- 1 Start OSA90 and load the file "demo01.ckt".
- 2 After examining the file, select "Compile Input File" from the "File" menu.

- 3 Select "Parametric" from the "Display" menu. OSA90 performs simulation to calculate the output labels, or, if a .haf file is available, the results from a previous simulation will be retrieved (Chapter 9). A dialog box will appear, press <ENTER> or click on "OK". You will see the Parametric View defined in the Sweep block displayed, as illustrated in Fig. 16.1.

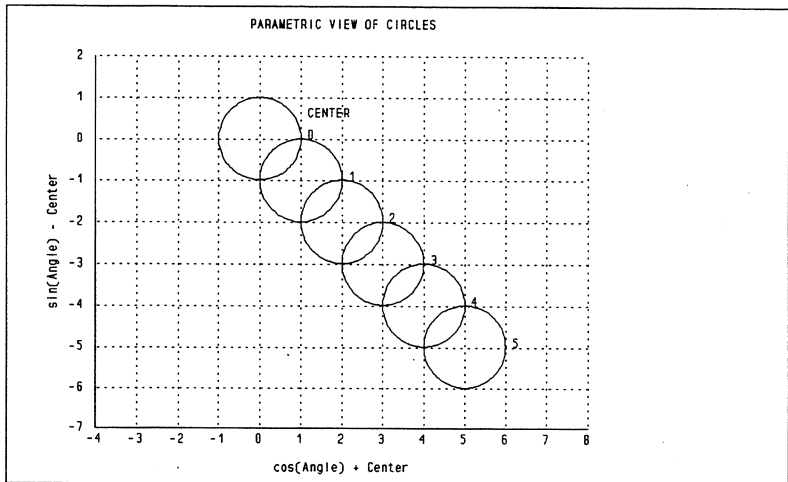


Fig. 16.1 Parametric display of circles.

## Examples

- Now select "Xsweep" from the "Display" menu. A dialog box will appear. Select "All output labels" for the "Y-axis" option. Press <ENTER> or click on "OK", and an Xsweep display of the two output labels versus the sweep label Angle will appear on the screen, as illustrated in Fig. 16.2.

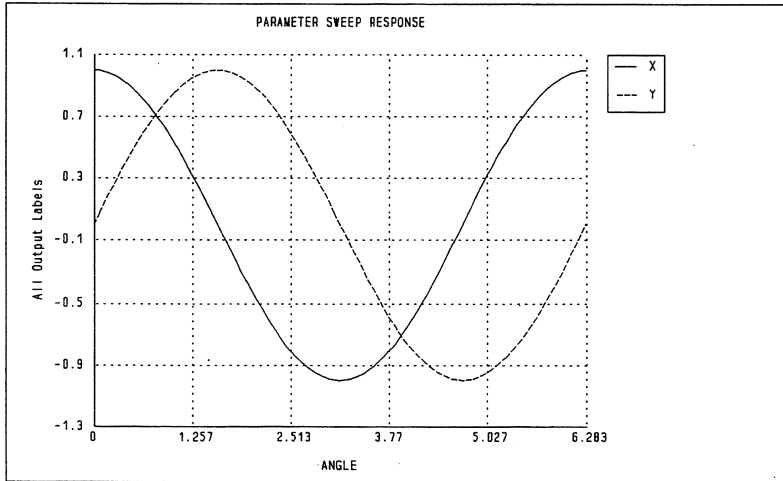


Fig. 16.2 Xsweep display of all output labels.

- You can now either exit OSA90 and view other examples.

## 16.3 Example demo02

This simple example illustrates the definition of optimization variables, functions and goals.

Consider the function

$$\text{Function} = (x_1 t + x_2 t^2) / (1 + x_3 t + x_4 t^2)$$

We wish to optimize the coefficients  $x_1$ ,  $x_2$ ,  $x_3$  and  $x_4$  so that *Function* will match

$$\text{Goal} = \text{sqrt}(t)$$

for  $t = 0, 0.02, 0.04, \dots, 1$ .

The  $\ell_1$  optimizer is suitable for this type of modeling problem (Chapter 11).

### Input File demo02.ckt

```
! Example demo02.ckt
! simple optimization example for OSA90
! matching Goal(t)=sqrt(t) by a rational function Function(t)
! in the interval [0 1] of t
! the coefficients of Function(t) are optimized
! by the L1 optimizer.

Expression
  t: 0;          ! sweep label

  x1: ?10?;     ! coefficients of Function(t)
  x2: ?10?;
  x3: ?10?;
  x4: ?10?;

  Function: (x1 * t + x2 * t * t) / (1 + x3 * t + x4 * t * t);

  Goal: sqrt(t);

  Error = Function - Goal;
end

Sweep
  t: from 0 to 1 step=0.02 Function Goal Error
  {Xsweep Y=Function.white & Goal.white.circle}
  {Xsweep Y=Error Ymin=-0.02 Ymax=0.02 NYTicks=8};
end

Specification
  t: from 0 to 1 step=0.02 Function = Goal;
end
```

## Step by Step "Guided Tour"

It is assumed that you are quite familiar with the general operations of OSA90, otherwise you can practise with demo01, as described in detail in Section 16.2.

- 1 Load and compile the file demo02.ckt.
- 2 Perform an Xsweep display with the default options. According to the View defined in the Sweep block, the output label Function is displayed as a continuous curve and Goal is shown as discrete circles, as illustrated in Fig. 16.3. You can clearly see a substantial mismatch between Function and Goal.

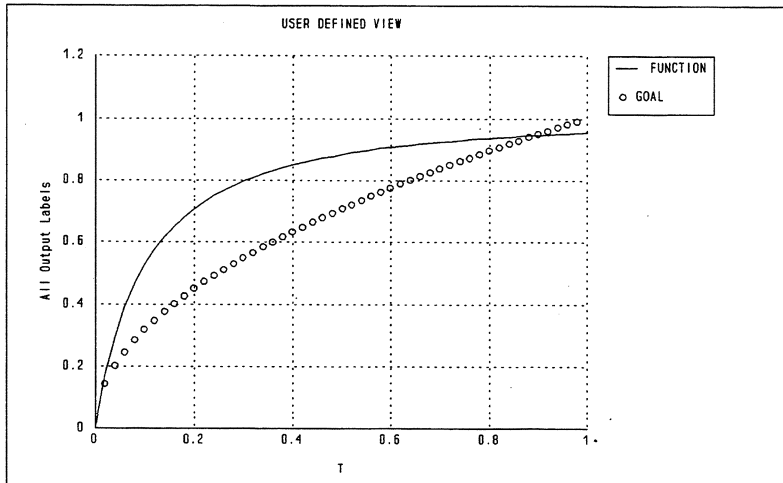


Fig. 16.3 Poor match between Function and Goal at the starting point.



- 3 Click on the "Optimize" toolbar button (see Chapter 2). A dialog box will appear, showing the default options. The  $\ell_1$  optimizer is suitable for this example. Press <ENTER> to start the optimization. The progress of optimization is reported on the screen.
- 4 After the optimization is completed, click on the "File editor" toolbar button. You can see that the variables in the input file (x1, x2, x3 and x4) are updated with the optimized values. The updated file is only a copy in the editor's memory buffer. If you wish to save it to the disk, you may do so (we recommend you specify a new file name). The optimized solution for this example is already saved in the file demo02\_o.ckt.
- 5 Compile the input file and perform another Xsweep display. You will see an excellent match between Function and Goal produced by the optimized solution, as illustrated in Fig. 16.4.

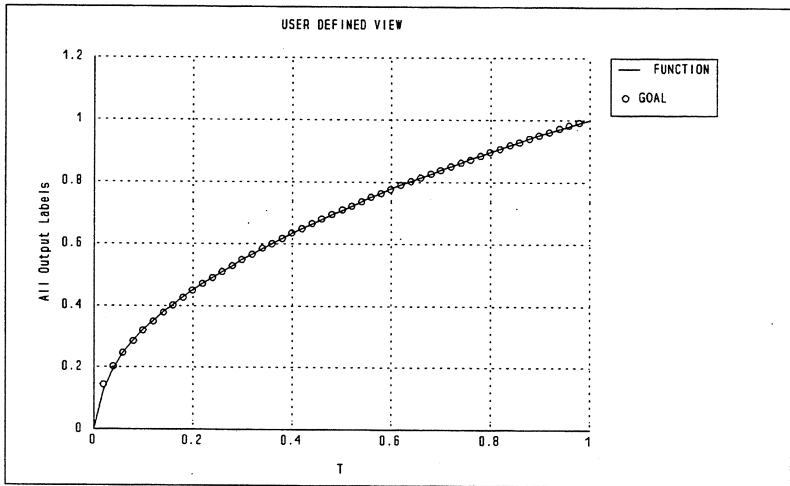


Fig. 16.4 Excellent match between Function and Goal at the solution.

## 16.4 Example demo03

This example illustrates simulation and optimization using an external program connected via Datapipe. Datapipe is described in Chapter 5.

### Input File demo03.ckt

```
! Example demo03.ckt
! Datapipe example for OSA90
! using Datapipe to communicate with an external simulator "transfrm"
! which simulates a three-section transmission line transformer
!
! The child program "transfrm" can handle both "SIM" and "FDF" protocols
! One of the inputs to the child is a flag indicating the protocol used
! With the SIM protocol, the child returns the reflection coefficient
! With the FDF protocol, the child also supplies the exact derivatives.

#define SIM_protocol 0 ! flag to the child indicating the protocol
#define FDF_protocol 1

Expression
! the vector of optimization variables X contains [E1 Z1 E2 Z2 E3 Z3]
! where Ei is the normalized length of the ith transmission line
! and Zi is the characteristic impedance of the ith transmission line
X[6] = [?0.8? ?1.5? ?1.2? ?3.0? ?0.8? ?6.0?];

Datapipe: FDF FILE="transfrm" NAME=Transformer
           N_INPUT=8 INPUT=(X, FREQ, FDF_protocol)
           N_OUTPUT=1;

Datapipe: SIM FILE="transfrm"
           N_INPUT=8 INPUT=(X, FREQ, SIM_protocol)
           N_OUTPUT=1 OUTPUT=(Reflection_Coe);

Return_Loss: if (Reflection_Coe > 0.01) (20 * log10(Reflection_Coe))
              else (-40);

end

Sweep
FREQ: from 0.2 to 1.8 step=0.05 Reflection_Coe, Return_Loss;
end

Specification
Datapipe: FREQ: 0.5, 0.6, 0.7, 0.77, 0.9, 1.0, 1.1,
           1.23, 1.3, 1.4, 1.5
           Transformer;

end

Control
Optimizer=Minimax;
end
```

### The Child Program

The child program "transfrm" simulates a three-section transmission line transformer to calculate the reflection coefficient. It is structured to accommodate both the SIM and FDF protocols for Datapipe communications (Chapter 5). One of the input parameters to the child "transfrm" is a flag indicating which protocol is used in a particular dialogue. According to this flag, the child program will perform the appropriate task. When the SIM protocol is used, the child will return the reflection coefficient. When the FDF protocol is used, the child will also compute and return the gradients.

## Source Code of the Child Program

```

/*****
* File:  transfmr.c
* Date:  31-Jul-1991
*
* Description:  Datapipe child program for demo03.ckt
*              Simulates a three-section transmission line transformer
*
* This program can handle both "SIM" and "FDF" Datapipe protocols
* One of the inputs from OSA90 is a flag indicating the protocol used
* With the SIM protocol, the reflection coefficient is returned
* With the FDF protocol, the exact derivatives are also supplied.
*
*****/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "ippcv2.h"

#define NX      8  /* number of inputs */
#define NY      1  /* number of outputs */
#define K_FREQ  6  /* which input is the frequency */
#define K_FLAG  7  /* which input is the protocol flag */

int transformer ();

main ()
{
    float x[NX], df[NX], ref_coe, freq;
    char *message;
    int nx, ny, protocol_fdf, need_gradient, error;

    df[K_FREQ] = df[K_FLAG] = 0.0;

    for (;;) {
        error = 0;

        pipe_initialize2();
        pipe_read2(&nx, sizeof(int), 1);
        pipe_read2(&ny, sizeof(int), 1);
        pipe_read2(x, sizeof(float), nx);

        if (nx != NX || ny != NY) {
            message = "Incorrect Number of Input/Output";
            error = strlen(message) + 1;
        }
        else {
            if (!(protocol_fdf = x[K_FLAG])) need_gradient = 0;
            else pipe_read2(&need_gradient, sizeof(int), 1);

            freq = x[K_FREQ];
            transformer(x, &ref_coe, df, freq);
        }

        pipe_write2(&error, sizeof(int), 1);
        if (!error) {
            pipe_write2(&ref_coe, sizeof(float), ny);
            if (need_gradient) pipe_write2(df, sizeof(float), nx);
        }
        else pipe_write2(message, 1, error);
    }
}

```

## Examples

```

int transformer ( x, ref_coe, gradients, freq )
float *x, *ref_coe, *gradients, freq;
{
    float length[3], z[3], alpha, beta, t, a, b, c, real, imag, coe_real, coe_imag,
        y_real, y_imag, VG_real, VG_imag, I_real[4], I_imag[4], V_real[4], V_imag[4];
    int i, j;

    beta = 0.2095845 * freq;
    alpha = 7.494813 * beta;
    I_real[3] = 1.0;
    I_imag[3] = 0.0;
    V_real[3] = 10.0;
    V_imag[3] = 0.0;

    for ( i = 0; i < 3; i++) {
        length[i] = 7.494813 * (*x++);
        z[i] = *x++;
    }

    for ( i = 2; i >= 0; i--) {
        t = beta * length[i];
        a = cos(t);
        b = sin(t) * z[i];
        c = sin(t) / z[i];

        j = i + 1;

        V_real[i] = a * V_real[j] - b * I_imag[j];
        V_imag[i] = a * V_imag[j] + b * I_real[j];

        I_real[i] = a * I_real[j] - c * V_imag[j];
        I_imag[i] = a * I_imag[j] + c * V_real[j];
    }

    real = V_real[0] + I_real[0];
    imag = V_imag[0] + I_imag[0];
    a = real * real + imag * imag;

    VG_real = real / a;
    VG_imag = -imag / a;

    coe_real = 1.0 - 2.0 * (I_real[0] * VG_real - I_imag[0] * VG_imag);
    coe_imag = -2.0 * (I_imag[0] * VG_real + I_real[0] * VG_imag);

    *ref_coe = sqrt(coe_real * coe_real + coe_imag * coe_imag);

    real = 2.0 * (coe_real * VG_real + coe_imag * VG_imag);
    imag = 2.0 * (-coe_imag * VG_real + coe_real * VG_imag);
    y_real = (real * VG_real - imag * VG_imag);
    y_imag = (real * VG_imag + imag * VG_real);

    for ( i = 0; i < 3; i++) {
        j = i + 1;

        real = V_real[i] * I_real[j] - V_imag[i] * I_imag[j] -
            V_real[j] * I_real[i] + V_imag[j] * I_imag[i];
        imag = V_real[i] * I_imag[j] + V_imag[i] * I_real[j] -
            V_real[j] * I_imag[i] - V_imag[j] * I_real[i];
        a = sin(beta * length[i]) * (*ref_coe);
        *gradients++ = alpha * (real * y_real - imag * y_imag) / a;

        real = V_real[i] * I_real[i] - V_imag[i] * I_imag[i] -
            V_real[j] * I_real[j] + V_imag[j] * I_imag[j];
        imag = V_real[i] * I_imag[i] + V_imag[i] * I_real[i] -
            V_real[j] * I_imag[j] - V_imag[j] * I_real[j];
        *gradients++ = (real * y_real - imag * y_imag) / (z[i] * (*ref_coe));
    }

    return(0);
}

```

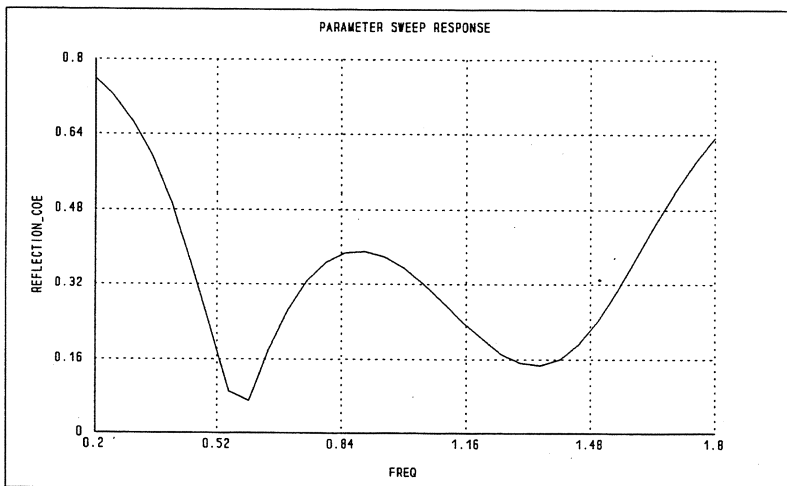
## Compile the Child Program

For information on how to compile the child program, please see the documentation that accompanied your compiler. The source code of the Datapipe server is provided with the software. A copy of the executable file "transfm" is included in the set of demonstration examples.

## Step by Step "Guided Tour"

It is assumed that you are quite familiar with the general operations of OSA90, otherwise you can practise with demo01, as described in detail in Section 16.2.

- 1 Read and compile the file demo03.ckt.
- 2 Perform an Xsweep display using the default options. The reflection coefficient of the transformer as calculated by the child program is displayed, as illustrated in Fig. 16.5.



*Fig. 16.5 Transformer reflection coefficient at the starting point.*

- 3 Select "Minimax" from the "Optimize" menu. This is the default optimizer specified in the Control block of the input file. When the dialog box appear, press <ENTER> or click on "OK" to start optimization.
- 4 Perform another Xsweep display using the default set of options. The optimized reflection coefficient of the transformer is displayed, as illustrated in Fig. 16.6.

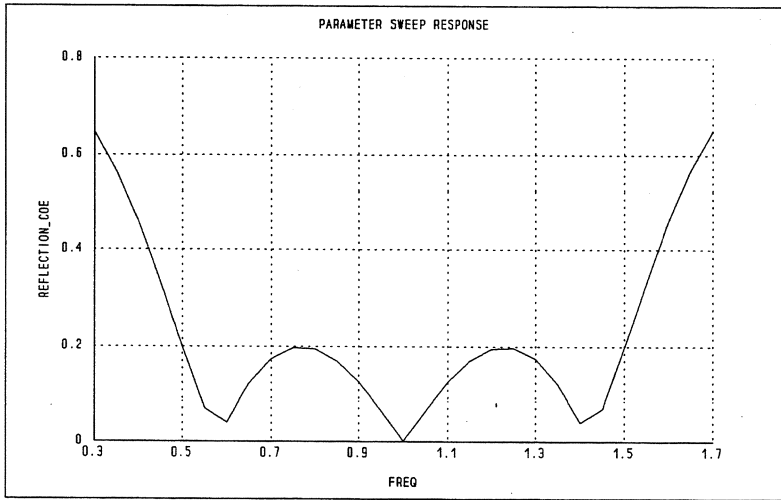


Fig. 16.6 Transformer reflection coefficient at the solution.

## Other Highlights

A few other useful features are also illustrated in the input file demo03.ckt. Text macros are used to define the protocol flags, namely SIM\_protocol and FDF\_protocol. The array X[6] is defined to contain the optimization variables. A postprocessing formula is created to compute Return\_Loss from the Datapipe output label Reflection\_Coe.

## 16.5 Example demo11

This example illustrates the harmonic balance simulation of a simple nonlinear FET circuit.

The circuit schematic is shown in Fig. 16.7. The *MGF1902* FET is represented by the built-in FETM nonlinear model.

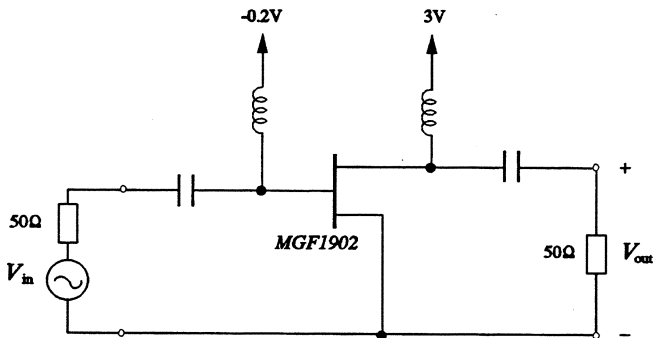


Fig. 16.7 Schematic of the nonlinear circuit.

### Input File demo11.ckt

```
! Example demo11.ckt
! demonstrate circuit simulation
! harmonic balance analysis. power sweep
! the FET model for Mitsubishi MGF1902 is extracted by HarPE

Model
CAP 1 2 C=1000pF;

Extrinsic4 20 30 40 2 3
RG=5 RD=0.2 RS=1.5 LS=0.0948NH
LG=0.53633NH LD=0.936NH GDS=0.002 CDS=0.0278PF
CDE=0.0682PF CX=10pF;

FETM 20 30 40
E=1.4 SL=0.126 IB0=8e-12 ALPHAB=1 VBC=6
GAMMA=-0.06356 K1=2 CF0=0.02PF KP=-0.157
IDSS=0.04716 VP0=-0.7338 R10=0.00343 C10=0.6294PF;

CAP 2 0 C=0.184pF; ! FET parasitic
CAP 3 0 C=0.216pF;

CAP 3 4 C=1000pF;

Input_Power: 0;

PORT 1 0 NAME=input P[1]=Input_Power;

PORT 4 0 NAME=output;
```

## Examples

```
IND 2 200 L=1000nH;
VSOURCE 200 0 NAME=gate VDC=-0.2;

IND 3 300 L=1000nH;
VSOURCE 300 0 NAME=drain VDC=3;

CIRCUIT;

Poutput[0:N_SPECTRA] = if (P#output > 0) (10 * log10(P#output) + 30);
end

Sweep
HB: FREQ=2GHZ Input_Power: from -12dBm to 6DBM step=3dBm
MVoutput PVoutput Poutput[1] Poutput[2] Poutput[3]
{Waveform Spectrum=(MVoutput, PVoutput). "Voutput_T"
Title="Power Sweep Output Voltage Waveforms"
Input_Power=all Tmin=0 Tmax=1 NT=100
Ymin=-2.5 Ymax=2.5 NYticks=5}
{Xsweep Title="Output Power Spectra vs. Input Power Sweep"
Y = Poutput[1] & Poutput[2] & Poutput[3]
Y_Title = "Output Power Spectra"
X_Title = "Available Input Power in dBm"
Input_Power=all
Ymin=-40 Ymax=20 NYticks=6 NXticks=6};
end
```

## The Sweep Block

The Sweep block in this example contains one sweep set. The simulation type of the sweep set is HB (harmonic balance). The sweep set contains one sweep label, namely `Input_Power`, which represents the available power in dBm at the input port.

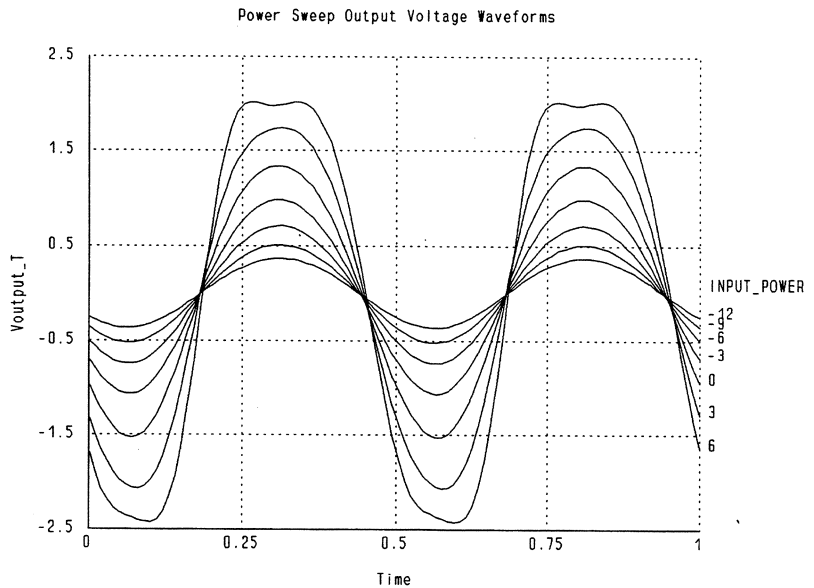
Two Views are defined in the Sweep block: the first one is designed to display the time-domain waveforms and the other one is a Xsweep View of the output power spectrum (`Poutput[k]`) versus the input power.



## Step by Step "Guided Tour"

It is assumed that you are quite familiar with the general operations of OSA90, otherwise you can practise with demo01, as described in detail in Section 16.2.

- 1 Load and compile the file demo11.ckt.
- 2 Select "Waveform" from the "Display" menu and click on "OK" to select the default option. The time-domain waveforms at different input power levels are displayed, as illustrated in Fig. 16.8.



*Fig. 16.8 Time-domain waveforms at different input power levels.*

## Examples

- Now perform an Xsweep display and select the default set of options. The output power spectral components corresponding to the first, second and third harmonics are plotted versus the available input power, as illustrated in Fig. 16.9.

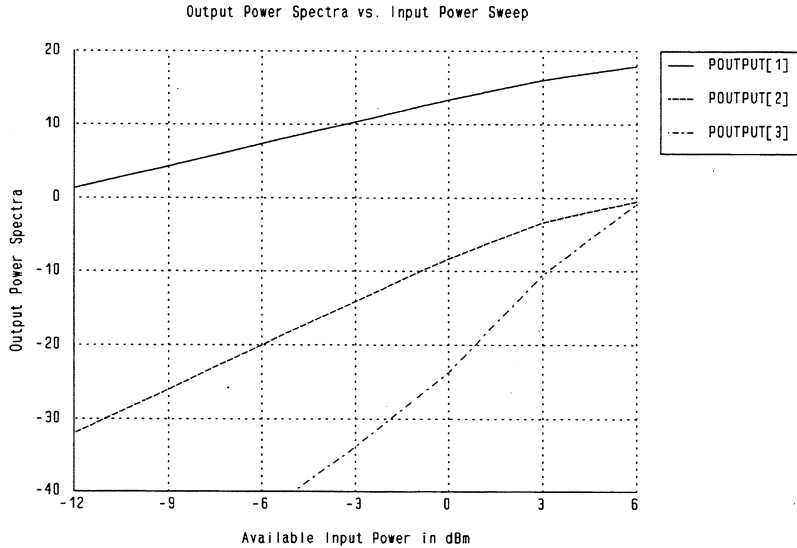


Fig. 16.9 Output power spectrum versus the available input power.

# **Appendix A**

## **Datapipe Server**



# Appendix A

## Datapipe Server

### Overview

The source code of the Datapipe server which consists of the set of functions listed in Table A.1 is provided with the software. The two files "ippcv2.h" and "ippcv2.c" are located the "osa90\_examples" subdirectory.

TABLE A.1 DATAPIPE SERVER FUNCTIONS

Name	Description
pipe_initialize2()	synchronizes dialogues
pipe_initialize3()	synchronizes dialogues with termination notification
pipe_read2()	reads data from Datapipe
pipe_write2()	writes data to Datapipe

Chapter 5 describes how to use the Datapipe server functions in a child program.



# Appendix B

## Diagnostic Messages





# Appendix B

## Diagnostic Messages

### Overview

This appendix lists the diagnostic messages that OSA90 may produce at run time. If an error occurs, you will be notified via a message on the status bar (located at the very bottom of the OSA90 window). The probable cause of each error is discussed and a possible solution is suggested if available.

**Diagnostic message:**

*identifier: message*

where *identifier* is a three-character code which OSA can use to trace the error.

### Internal Errors

**Internal error message:**

*identifier: Internal Error xxxx*

where *xxxx* is an error index which OSA uses to identify the internal error.

If you encounter an internal error message, please contact Optimization Systems Associates Inc. for technical assistance:



Phone 905 628 8228  
Fax 905 628 8225  
Email [osa@osacad.com](mailto:osa@osacad.com)

## Diagnostic Messages

### CFP: A Constant Integer Is Expected

A constant integer is expected following certain keywords.

### CFP: A Constant Value Or Constant Label Is Expected

In the **Specification** block, output labels can be functions of the optimization variables, but the goals must be specified by constant values, labels or expressions not dependent on the optimization variables.

### CFP: A Node Number Or Name Expected

A node number (an integer) or a node name (a string with the prefix @) is expected (Chapter 6). Fewer nodes than expected have been specified.

### CFP: A Pair Of Square Brackets Expected

Square brackets are used to delimit array dimensions, array indices and correlation indices. Check the balance of brackets and the context of usage.

### CFP: A Positive Integer Expected

A positive integer is expected for an array index or following certain keywords.

### CFP: All Input Keywords Must Precede Output Keyword

In a sweep set in the **Sweep, Specification** or **MonteCarlo** block, the definitions of sweep parameters, dependent labels and independent labels must appear before any output labels.

### CFP: Array Name Expected

Array names are expected as arguments for array functions.

### CFP: Character String Entry Expected

A character string entry is expected following a string label definition or certain keywords such as **NAME** or **FILE**.

### CFP: Colon (:) Expected Following The Keyword

In the definition of the dimension of an array, a colon is expected between the two indices. Also, a colon is used to identify a sweep parameter in the **Sweep, Specification** and **MonteCarlo** block.

**CFP: Comma Expected Between Arguments**

When multiple arguments (parameters) are entered (e.g., for a Datapipe or a built-in transformation function), they should be separated by commas.

**CFP: Correlation Matrix Diagonal Entry Must Be 1.0**

A diagonal entry of the correlation matrix represents the "self-correlation" of a statistical parameter, which must be 1.0 by definition.

**CFP: Correlation Matrix Dimension Is Undefined**

The dimension of each correlation matrix in the Statistics block must be explicitly defined.

**CFP: Correlation Matrix Dimension Must Be > 1**

The purpose of a correlation matrix is to represent the correlations among a set of statistical parameters. Obviously, to be meaningful it must involve at least two statistical parameters, i.e., the dimension of the correlation matrix must be greater than one.

**CFP: Correlation Matrix Must Be Symmetrical**

Correlation matrices must be symmetrical by definition.

**CFP: Correlation Name Expected**

In the definition of correlated statistical parameters, following the CORRELATION keyword you must specify the name which uniquely identifies a correlation matrix defined in the Statistics block.

**CFP: Datapipe Type Keyword Expected**

In a Datapipe definition statement, you must identify the type of the Datapipe protocol using one of the keywords: SIM, COM, FUN or FDF.

**CFP: Duplicate Definition Of Keyword Or Label Name**

Most of the keywords can only be specified once in each definition. Each label must be identified by a unique name. Therefore, reference to a keyword which has already been specified or usage of a name which already identifies another label is not allowed.

**CFP: End-Of-Record Symbol Semicolon (;) Expected**

The compiler expects a semicolon marking the end of a statement but encounters an extraneous entry which appears to be out of context.

## **Diagnostic Messages**

### **CFP: Equal Sign (=) Or Colon (:) Missing**

An equal sign "=" or a colon ":" is required between a keyword and a value or between a label name and an expression.

### **CFP: Equal Sign Expected For Single Value Assignment**

In the Sweep, Specification and MonteCarlo blocks, some keywords such as HARM, RREF and FREQ2 must be followed by the equal sign and then a scalar value, label or expression.

### **CFP: Expression Operand Expected**

In an expression, between two operators there must be an operand.

### **CFP: Expression Operator Expected**

In an expression, between two operands there must be an operator.

### **CFP: Expression Too Long**

The length of a single expression is limited by the compiler's internal buffer. Avoid writing an extremely long expression as a single statement. Try to break it up into several smaller expressions. Another likely cause is ambiguous text consisting of multiple statements not properly separated by semicolons ";".

### **CFP: File Name Expected**

A file name is expected to follow certain keywords, such as #include.

### **CFP: FUN Or FDF Datapipe Name Expected**

In the Specification block, for a Datapipe specification set you must provide the name which uniquely identifies a FUN or FDF Datapipe.

### **CFP: Illegal Data Type**

The data type is out of context. For instance, a floating point number appears where an integer is expected, or an optimization variable appears where a constant value is expected.

### **CFP: Illegal Index (Not An Integer Or Exceeds Limit)**

An index must be specified by an integer within the valid range. Likely the offending entry is not an integer, or its value exceeds the limits (e.g., a negative index when it should be positive).

**CFP: Illegal Number Of Ports**

The number of ports defined in a circuit must be between 1 and 64.

**CFP: Illegal Operator Or Reserved Symbol**

User-defined names and other string entries cannot contain characters reserved as operators. See Chapter 4 for the list of reserved characters.

**CFP: Illegal Range Definition**

Improper definition of a sweep range (such as From x1 to x2 Step=x3). Check the consistency between the starting point, end point and step size. Possibly one of the keywords is missing or misspelled.

**CFP: Import Data Incomplete: Whole Matrix Expected**

Imported data are used in OSA90 to represent linear n-port subcircuits. For an n-port, a complex  $n \times n$  S/Y/Z matrix must be supplied.

**CFP: Import Data Must Contain At Least 3 Frequencies**

OSA90 uses cubic splines to interpolate imported data for the frequencies involved in simulation. The algorithm requires at least three distinct points in order to establish the cubic spline coefficients.

**CFP: Inadequate Data**

Incomplete definition. Most likely one or more keywords are missing. Check the list of keywords in the relevant section of the User's Manual.

**CFP: Include File Or Macro Cannot Contain Variables**

Include files and text macros are handled by the preprocessor. The contents of an include file is merged into the input file at compiled time. Text macros are expanded into the appropriate substitution text in the input file. Include files and text macros cannot contain optimization variables, because such variables cannot be properly updated after optimization.

**CFP: Inconsistent Bounds**

Inconsistent bounds in the definition of an optimization variable: the lower bound is larger than the nominal or the nominal is larger than the upper bound.

## Diagnostic Messages

### CFP: Incorrect Number Of Entries

The number of entries is different from what is expected, such as the number of arguments to a Datapipe or a built-in transformation function. The error can also be due to a missing bound in defining an optimization variable: a variable cannot be defined as ?x1 x2?.

### CFP: Input File Block Name Expected

The compiler encounters extraneous text outside any block in the input file. Each block of the input file must be within a block name and the keyword END.

### CFP: Keyword DATA - Expected For Import Data

The DATAPORT element can be used to introduce imported linear subcircuit into the Model block. The set of imported data must be identified using the keyword DATA=*import\_name* (see Chapters 6 and 8).

### CFP: Keyword Illegal Or Out Of Context

The offending keyword is used in a place where it is not applicable.

### CFP: Keyword Incompatible With Those Already Specified

A keyword is found to be in conflict with the other keywords already specified in the same statement. For instance, you may have specified the simulation type as DC and then attempt to refer to an AC response label.

### CFP: Keyword Is Applicable To 2-Ports Only

Some of the response labels are available for 2-ports only. See Chapter 6.

### CFP: Name Must Be Specified For Identification

A character string identifier following the keyword NAME is expected, such as in a FUN or FDF Datapipe.

### CFP: Name Of Import Data Expected

Each set of imported data must be identified by a unique name in the ImportData or the LINEAR Datapipe statement. The same name must be used to refer to the imported data in the Model block (see Chapter 6).

**CFP: Name Of Matrix Expected**

Certain keywords must be followed by the name of a matrix. For example, the keyword DATA of the element SPORT must be followed by the name of a matrix containing the appropriate *S* parameters (see Chapter 8).

**CFP: Name Of The Include File Not Recognized**

When importing the *S* parameters in the Touchstone® format (see Chapter 6) the file name extension must be ".SnP" where *n* is a single digit number between 1 and 9.

**CFP: Nested Include Files Are Not Allowed**

The #include preprocessor directive can be used only in the input file. An input file can have any number of include files. But an include file cannot contain another (nested) include file.

**CFP: No Output Label Or Optimization Goal Specified**

Each specification set in the Specification block must define at least one output label, i.e., at least one error function for optimization.

**CFP: No Statistical Parameter Has Been Defined**

In order to invoke the Monte Carlo analysis feature or yield optimization, at least one statistical parameter must be defined in the input file.

**CFP: Not A Valid Physical Unit**

Unrecognizable unit name for a numerical entry. See Chapter 3 for the list of valid physical units.

**CFP: Number Of Arguments Inconsistently Specified**

The number of arguments specified for a text macro, a Datapipe or a built-in transformation function is incorrect. Or the number of actual inputs or outputs of a Datapipe is inconsistent with the N\_INPUT or N\_OUTPUT specified.

**CFP: Number Of Arguments Must Be Specified First**

The number of inputs or outputs of a Datapipe must be specified as N\_INPUT or N\_OUTPUT before the actual arguments are listed (INPUT or OUTPUT).

## Diagnostic Messages

### CFP: Number Of Optimization Variables Exceeds Limit

Too many optimization variables have been defined. This limit can be changed if necessary (contact OSA).

### CFP: Number Of Ports Mismatched Between Data And Usage

The number of ports in a reference to an imported subcircuit must be consistent with its definition (e.g., an imported 2-port cannot be used as a 3-port).

### CFP: Number Of Statistical Outcomes Must Be Specified

In the MonteCarlo block of the input file, the number of statistical outcomes must be specified in the first sweep set as N\_OUTCOMES (the number of outcomes is the same for all Monte Carlo sweep sets).

### CFP: Number Of Statistical Parameters Exceeds Limit

Too many statistical parameters have been defined. This limit can be changed if necessary (contact OSA).

### CFP: Numerical Entry Expected

A character string entry is found where a numeric entry is expected.

### CFP: Only A Constant Or Optimizable Label Can Be Swept

The sweep parameters in the Sweep and MonteCarlo blocks must be defined in the Expression block as constant labels or optimization variables. In the Specification block, the sweep parameters can only be constant labels.

### CFP: Reference Impedance RREF Must Be Positive

The keyword RREF is used to specify the reference impedance for  $S$  parameters. The reference impedance must be a positive value.

### CFP: Size Of Argument Inconsistent With Definition

The size of an argument for an array function or a built-in transformation function is different than what is expected. See Chapter 4 for the appropriate definition. Especially check the consistency between the input and output arguments.

### CFP: Size Of The Arrays Must Be Consistent

The arrays involved in a vector expression must have the same size.



**CFP: The Last File Block Not Closed With END**

The last block in the input file is not closed with the keyword END.

**CFP: Tolerance Undefined For Statistical Parameter**

The definition of a statistical parameter must include the tolerance or standard deviation.

**CFP: Too Few Distinct Nodes Specified Than Expected**

In the Model block some circuit elements require a minimum number of distinct nodes. For example, a resistor (RES) requires at least one distinct node.

**CFP: Too Many Arguments For Datapipe INPUT/OUTPUT**

The number of input or output arguments is limited. This limit can be changed if necessary (contact OSA).

**CFP: Too Many Input Labels In One Set**

The number of parameter labels in a sweep set in the Sweep, Specification and MonteCarlo blocks is limited to 64.

**CFP: Too Many Nodes Specified Than Expected**

You have specified more nodes than expected for a circuit element in the Model block. For example, you cannot specify three nodes for a resistor (RES).

**CFP: Too Many Output Labels In One Set**

The number of output labels in each sweep set in the Sweep, Specification or MonteCarlo block is limited to 512. If necessary, you can define multiple sweep sets and distribute the output labels among the different sets.

**CFP: Too Many Steps In A Single Range Definition**

The number of steps for a sweep parameter exceeds the limit. In the Sweep, Specification and MonteCarlo block, each sweep parameter can have up to 1024 discrete points (values).

**CFP: Too Many Sweep Labels In One Set**

OSA90 supports one- and two-dimensional sweeps, in other words, only one or two sweep parameters can be defined for each sweep set.

## Diagnostic Messages

### CFP: Too Many Sweep Steps For Labelling

In user-defined Graphical Views the maximum number of different items (e.g., curves) for labelling is 64. Check the number of steps.

### CFP: Too Many Views Defined Within One Sweep Set

The number of user-defined Graphical Views in each sweep set in the Sweep block is limited to 64.

### CFP: Trailing Index Expected For This Entry

In reference to individual array elements, an index in square brackets is required following the array name. In the definition of a correlated statistical parameter, an index to a correlation matrix is required.

### CFP: Unbalanced Or Missing Parenthesis Or Bracket

Parentheses and brackets must appear in pairs. Check for a matching number of left and right parentheses or brackets.

### CFP: Undefined Symbol Referenced

A string entry is found which does not match any valid keywords or the names of the defined labels. Perhaps the name is misspelled.

### CFP: Unrecognizable Entry

A keyword or operator is expected. The entry found does not match any valid keywords or operators.

### CFP: User Entry Expected Following A Keyword

The preceding keyword must be followed by a numerical value, a label name or a string entry.

### CFP: Value Exceeds Limit: Too Large Or Too Small

The value specified exceeds the limits.

### CFP: View Contains Label Not Defined In The Sweep Set

The output labels contained in a Graphical View must have been defined as output labels in the Sweep or MonteCarlo block that contains the View.

**CFP: View Requires Two Sweep Parameters**

The Visual View for 3D visualization can be defined only for sweep sets which contain two sweep parameters (two-dimensional sweeps).

**CFP: View Type Keyword Expected**

The definition of a Graphical View must begin with a View type keyword (see Chapters 10 and 12).

**CFP: Weighting Factors Cannot Be Negative**

Weighting factors in the Specification block can be used to change the relative scales of the error functions. But they should not change the signs of the error functions. Therefore, a negative weight is not allowed.

**CKT: Circuit Definition Incomplete**

Circuit elements and nodes have been defined in the Model block but no CIRCUIT statement is defined.

**CKT: Circuit Definition Is Already Complete**

Circuit elements cannot be defined after the CIRCUIT statement.

**CKT: Data FORMAT Statement Missing**

Each imported data set must have a FORMAT statement which defines the order of the data entries.

**CKT: DC Nonlinear Simulation Did Not Converge**

The Newton iteration failed to solve the nonlinear DC circuit equations. The program has built-in convergence-improving mechanism such as automatic source stepping. Therefore a non-convergence at DC must indicate a very difficult problem. Double check the circuit definition. You may also try again with reduced bias voltages.

**CKT: Different Types Of Responses Cannot Be Mixed**

Different types of circuit responses cannot be mixed in the same sweep set or the same expression. For example, you cannot mix a DC current response with a small-signal  $S$  parameter.

### CKT: Floating Node Encountered In Circuit Definition

A floating node refers to a node in the Model block which is referenced only once. Floating nodes are not allowed. Use the element OPEN if necessary (see Chapter 8).

### CKT: Illegal Definition Of Linear Subcircuit

The definition of a linear subcircuit is illegal. For example, a linear subcircuit must not contain nonlinear device models or independent sources. See Chapter 6.

### CKT: Illegal Use Of VLABEL In Parameter Definition

Voltage and current labels can be used in user-defined device models (see Chapter 6). Their usage may be restricted in other situations. For example, you cannot use voltage or current labels to define the delay parameter of another voltage label, otherwise it can result in circular dependency.

### CKT: Import 1-Port Data Name Expected

If you specify the termination of a port using the Z keyword, then it must be followed by the name of an imported 1-port.

### CKT: Isolated Nonlinear Device(s) Encountered

The compiler detects one or more nonlinear devices defined in the Model block which are completely isolated from the external ports and sources.

### CKT: Mismatch Of Subcircuit Terminals (Nodes)

In reference to an  $n$ -port subcircuit, if the number of nodes specified is less than  $n$  or more than  $n+1$ , then it is considered as a mismatch.

### CKT: No AC Source Has Been Defined

Small-signal and large-signal AC analyses require the definition of at least one AC source.

### CKT: No Circuit Definition Available

Certain keywords in the input file (such as HB and RREF) are relevant only if a circuit is defined. They cannot be used without a circuit definition.

### CKT: No DC Bias Source Has Been Defined

To perform DC analysis (by indicating the DC simulation type), you must define at least one DC source in the circuit.

**CKT: No External Ports Have Been Defined**

To perform small-signal AC analysis (by indicating the AC simulation type), you must define at least one port in the circuit. Otherwise, the *S* parameters cannot be defined.

**CKT: No Frequency Defined For The Second Tone**

If one or more of the AC sources have been defined as two-tone sources, then you must define the second-tone frequency using the FREQ2 keyword for each large-signal sweep set (HB simulation type).

**CKT: No Frequency Has Been Defined**

For each small-signal or large-signal sweep set (AC or HB simulation type), you must define the frequency using the predefined label FREQ.

**CKT: No Input Parameter Specified**

No input parameters have been specified in a reference to a transformation or a library function (see Chapter 4).

**CKT: No Input/Output Ports Or Sources Defined**

The built-in circuit responses consist of the DC/AC currents and voltages at the ports and sources. If no ports or sources are defined, then no response can be calculated and therefore the circuit definition is not meaningful.

**CKT: No Nodes Have Been Defined**

The CIRCUIT statement is found before any node is defined.

**CKT: No Output Labels Specified**

A sweep set in the Sweep, Specification or MonteCarlo block must contain at least one output label. Otherwise no useful results are produced from the simulation.

**CKT: No Source Defined For VLABEL calculation**

In order for OSA90 to calculate AC and/or DC voltage and/or current responses, at least one independent source must be included in the Model block.

**CKT: Number Of Parameters Exceeds Limit**

For user-defined elements, the total number of parameters is limited to 64 per element.

## Diagnostic Messages

### CKT: Number Of Spectral Components Exceeds Limit

The total number of intermodulation products (spectral frequencies) exceed a preset limit. Reduce the highest harmonic index (the HARM keyword).

### CKT: Number Of State Variables Exceeds Limit

The total number of state variables from all nonlinear elements exceed a preset limit. The current limit is 64 at DC. Note that the number of state variables in the (AC) harmonic balance equation is  $(2 * N\_Spectra + 1)$  times the number of DC state variables.

### CKT: OSCPORT Must Be Defined With A Stimulus Voltage

For oscillator analysis using the OSCPORT element, a stimulus voltage must be specified. See Chapter 6.

### CKT: Output Keyword Cannot Be A PARAMETER, Use FORMAT

In the definition of imported data set, the data keywords such as MSij cannot be used in the PARAMETER statement. They are for the FORMAT statement.

### CKT: Output Label Name Conflicts With User Label

One or more of the names of the built-in response labels are already taken by user-defined labels. See Chapter 6.

### CKT: Subcircuit Must Be Defined Before Main Circuit

In the Model block, linear subcircuits and user-defined elements must be defined before the main (top-level) circuit. See Chapter 6.

### CKT: Subcircuit Terminals Must Be Distinct Nodes

In the definition of subcircuits, the nodes for external connection (i.e., the nodes preceding the opening "(") must be distinct, because each of these node defines a port.

### CKT: Y Matrix Singular In VLABEL/ILABEL Calculation

The nodal Y matrix is found to be singular during the calculation of voltage and/or current labels.

### FPE: Floating Point Exception

Floating-point operation results in an unspecified error.

**FPE: Floating Point Overflow**

Floating-point operation results in overflow.

**FPE: Math Function Argument Domain Invalid**

Floating-point error: argument for a math library function is out of its domain of definition (e.g., LOG(x) with  $x < 0$ ).

**FPE: Math Function Argument Singular**

Floating-point error: argument for a math library function is "singular" (such as TAN(x) with  $x = \pi/2$ ).

**G00: Cannot Open File**

The specified disk file cannot be opened for reading or writing because the file does not exist, or the file/path is protected from the intended operation.

**G00: Circuit Size Exceeds Licensed Limit**

Your copy of OSA90 can handle limited size circuits only. If necessary contact OSA.

**G00: Error Writing To Disk File**

This error is likely due to file protection or insufficient disk space.

**G00: File Not Found**

The specified file cannot be found.

**G00: Not Enough Memory For The Intended Operation**

No memory sufficient for the intended operation can be allocated. Perhaps the size of the problem defined demands excessive memory space. Try to reduce the number of variables and/or sweep steps.

**G00: Number Of Variables Exceeds Licensed Limit**

Your copy of OSA90 can handle limited size problems only. You need to reduce the number of optimization or statistical variables to proceed. If you still want to perform optimization with a larger number of variables than the licensed limit consider subsets of all variables on a fewer at a time basis and invoke optimization several times. Contact OSA if necessary.

## Diagnostic Messages

### G00: Size Of File Exceeds Limit

The size of an input file exceeds the limit. This limit applies to the size of a single file, therefore try to break up a large file into several smaller ones and use the #include feature to link them together. Also, this limit can be changed if necessary (contact OSA).

### G00: This Feature Is Not Licensed

Your copy of OSA90 does not include the attempted feature. If necessary contact OSA.

### G00: Unknown Run-Time Option

Run-time options can be specified when you start the OSA90 program.

### GRF: Array Must Contain Constant Data

User-defined arrays for Graphical Views must contain constant data.

### GRF: Keyword Must Be Followed By ON Or OFF

Certain keywords, such as the Graphical View keyword LEGEND, are used to switch on or off certain options. They must be followed by either ON or OFF.

### GRF: Log Scale Cannot Include Zero Or Negative Values

If you choose to display a sweep parameter (X-axis) or a response (Y-axis) in log scale (see Chapters 9 and 10), the value range of that parameter or response must be positive, since log cannot be applied to zero or negative values.

### GRF: No Data Range (All Points Have The Same Value)

The X-axis range of a graphical display must not be zero. If all the points have the same X-axis coordinates, then they cannot be plotted.

### GRF: No Sweep Definition Available

The option "Parametric" under the "Display" menu requires at least one sweep parameter be defined in the sweep set.

### IPC: Cannot Execute The Specified Child

The external program (child) specified for Datapipe connection cannot be invoked (executed). Check the file name, path name and file protection (OSA90 must have the privilege to execute the child program).



**IPC: Datapipe Output String Size Exceeds Definition**

The size of an output string from the child program exceeds the string length defined in the input file.

**IPC: Failed To Open Inter-Process Pipes**

The attempt to create inter-process pipes between OSA90 and the child program has failed.

**IPC: Illegal Datapipe Output Data Type**

The data type (IPPC\_DATA\_CHAR or IPPC\_DATA\_FLOAT) of an output returned by the child program does not match the definition in the Datapipe statement in the input file.

**IPC: Operation Terminated By Child Program**

The external program (child) invoked through Datapipe connection has forced abnormal termination.

**LSM: Complex Value Encountered For DC Calculations**

The controlling coefficient or impedance of a linear controlled source (CCCS, CCVS, VCCS or VCVS) is found to have a nonzero imaginary part at DC.

**LSM: Invalid External Or Bias Port Configuration**

The topological definition of the external ports and/or sources in the Model block causes difficulties in formulating the linear subcircuit Y matrices. The ports and sources should not overlap or form a closed circle.

**LSM: Linear Subcircuit Singular**

The linear subcircuit defined in the Model block or an imported linear n-port is singular.

**LSM: Port Termination Singular: Open Circuit**

The termination of a port is singular (either an open circuit or the impedance value exceeds the floating-point range).

**LSM: Relative Dielectric Constant Less Than One**

A less than one value of the relative or effective dielectric constant has been encountered during simulation or optimization. Check the input file. If the parameter is assigned by a label, make sure that the label always evaluates to a proper value.

## **Dagnostic Messages**

### **LSM: Zero Characteristic Impedance**

The characteristic impedance of a transmission line is found to have a zero or negative value during circuit simulation or optimization. Check the input file. If the parameter is assigned by a label, make sure that the label evaluates to a proper value.

### **LSM: Zero Or Negative Element Parameter Value**

A circuit element parameter is found to have a negative or zero value during simulation or optimization. Check the input file for erroneous entry. Make sure that the labels used for parameter assignment would not evaluate to a negative or zero value.

### **LSM: Zero Or Negative Reference Frequency For Scaling**

The reference frequency for scaling the transmission line electrical length or attenuation is found to have a zero or negative value during circuit simulation or optimization. If the parameter is assigned by a label, make sure that the label evaluates to a proper value.

### **MSL: Dielectric Constant Value Too Large**

The dielectric constant of a microstrip element is found to exceed the upper limit of the model validity range. Check the input file. If the parameter is assigned by a label, make sure that the label evaluates to a proper value.

### **MSL: Dielectric Constant Value Too Small**

The dielectric constant of a microstrip element is found to exceed the lower limit of the model validity range. Check the input file. If the parameter is assigned by a label, make sure that the label evaluates to a proper value.

### **MSL: Frequency Too High For Microstrip Components**

The simulation frequency for a microstrip element is found to exceed the upper limit of the model validity range.

### **MSL: Microstrip Shielding Height < Substrate Thickness**

The shielding height of a microstrip element is found to be smaller than the substrate thickness. Check the input file. If the parameter is assigned by a label, make sure that the label evaluates to a proper value.

### **MSL: Microstrip Slit Too Deep**

The depth of a microstrip slit is found to exceed the upper limit of the model validity range. Check the input file. If the parameter is assigned by a label, make sure that the label evaluates to a proper value.

**MSL: Microstrip Slit Too Wide**

The width of a microstrip slit is found to exceed the upper limit of the model validity range. Check the input file. If the parameter is assigned by a label, make sure that the label evaluates to a proper value.

**MSL: Microstrip Step Has Same Widths ( $W1 = W2$ )**

The widths of a microstrip step cannot be defined with the same value.

**MSL: Microstrip Width-Height Ratio Too Large**

The width-height ratio of a microstrip element is found to exceed the upper limit of the model validity range. Check the input file. If the parameter is assigned by a label, make sure that the label evaluates to a proper value.

**MSL: Microstrip Width-Height Ratio Too Small**

The width-height ratio of a microstrip element is found to exceed the lower limit of the model validity range. Check the input file. If the parameter is assigned by a label, make sure that the label evaluates to a proper value.

**MTH: Divide By Zero In Insertion Loss Calculation**

The calculation of the insertion loss is singular when, for example, the input port is open or the load power is zero.

**MTH: Insufficient Data For Spline Interpolation**

In processing imported data, the cubic spline interpolation employed by OSA90 requires a minimum of three frequency points be supplied.

**MTH: Invalid Reference Impedance Rref For S-Pars**

The reference impedance for  $S$  parameters ( $RREF$ ) must be greater than zero.

**MTH: LU Factors With Pivoting Cannot Be Extracted**

The array functions `EXTRACT_L()` and `EXTRACT_U()` can only be applied to results obtained from `LUF()` (see Chapter 4). They cannot be applied to the results from `LU()` which may involve pivoting in the factorization.

## Diagnostic Messages

### MTH: Matrix Singular In LU Factorization

The matrix to be LU factorized is found to be singular. This can be the Y matrix of the linearized circuit for small-signal analysis.

### MTH: Y Matrix Singular In Partial Elimination

The nodal Y matrix is found to be singular during DC or harmonic balance simulation.

### OPT: Simulation Did Not Converge

During analysis invoked by the option "Sensitivity Analysis" under the "Optimize" menu, the circuit simulation failed to converge at the nominal point. Check the circuit definition. Reduce the bias voltages and/or input power levels. Try simulation first.

### SED: A Label Name Is Expected

In the Report block, the dollar sign is used to delimit references to labels. A label name is expected following a dollar sign.

### SED: Label In Report Must Be Included In A Sweep Set

If the Report block contains references to labels which represent or depend on circuit responses, then those labels must have been included in the Sweep block for specific simulation types and parameters.

### SED: Number Of Report Sections Exceeds Limit

Typically, this indicates that the Report block contains too many references to labels and/or too many printing formats. Try to simplify the structure of the report.

### SED: Report Set $\${...}\$$ Does Not Match Any Sweep Set

In the Report block, the delimiter  $\${...}\$$  indicates the reporting of labels versus parameter sweeps. Each set must correspond to one or more sweep sets defined in the Sweep block.

### SED: Unbalanced Dollar Sign In Report Block

In the Report block, the dollar signs, used to delimit references to labels, must appear in pairs.

### STM: Correlation Matrix Is Not Positive Semi-Definite

To be mathematically correct, correlation matrices must be positive semi-definite.

**STM: FORMAT Statement Required**

The Sample block must include at least one **FORMAT** statement which defines the names of the statistical variables and the units of the sample data (see Chapter 12).

**STM: Inaccurate Factorization Of Correlation Matrix**

One or more correlation matrices are numerically ill-conditioned.

**STM: Sample Name Required**

When a statistical variable is defined with the **SAMPLE** distribution, the name of the label must match a name given in the Sample block (see Chapter 12).

**STM: Sample Size Exceeds Limit**

The size of statistical sample contained in the Sample block is limited to 512.

**STM: Sampled Variable Must Have A Constant Nominal**

Statistical variables defined with the **SAMPLE** distribution must have constant nominal values. During Monte Carlo analysis, the values of such statistical variables are directly retrieved from the Sample block (see Chapter 12).

**TLP: A Text Macro Argument Name Expected**

In the definition of a macro function, commas are used to separate the arguments. This message indicates an extraneous comma not followed by an argument name. In a macro reference, the message indicates that an expected argument is missing.

**TLP: Illegal Or Missing Text Macro Name**

Following the preprocessor directive **#define**, a text macro name must be specified by a valid and unique string identifier.

**TLP: Length Of Text Record Exceeds Limit**

The length of a text record (e.g., the body of a text macro, a single statement) is limited. This limit can be changed if necessary (contact OSA).

## Diagnostic Messages

### TLP: Macro Cannot Define Explicit Variable: Use Label

Text macros are merely a preprocessor feature for the user's convenience. They are expanded into their substitution text at the compiling time. A text macro cannot contain an explicit definition of an optimization variable, because it cannot be properly updated after optimization, and because of the ambiguity which will arise from multiple references.

### TLP: Nested Definition Of Text Macros Is Illegal

The definition of a text macro may contain references to other macros which are already defined, but it cannot embed another macro definition.

### TLP: Not A Valid Number

The offending entry begins with a numeral character but it cannot be converted into a numerical value. The leading character of a string identifier cannot be a numeral character (digit, period, plus or minus sign).

### TLP: Too Many Arguments For A Text Macro

The number of arguments for a text macro function is limited to 512.

### TLP: Too Many Nested Levels Of Text

The number of levels of nested text macro references (i.e., the definition of a text macro contains references to other macros) is limited to 8.

### TLP: Too Many Tokens (Fields) In A Single Record

Too many tokens (fields, entries) on a single text line or in a single record (the limit is 4096 tokens per record). Most likely this is caused by a long section of text running continuously without semicolons to separate it into statements.

### TLP: Unbalanced Quotation Mark (Not Closed)

A pair of quotation marks delimits a literal string entry. The closing mark is missing.

# Index

.haf files, 9-43  
 .mca files, 12-34  
 .sen files, 11-49  
 \$ argument, 3-11

## A

Absolute values of array elements, 4-20  
 Accuracy of solution,  
   Newton iteration, 3-21  
   optimization, 3-14, 11-35, 13-4  
 Adjoint analysis, 3-16, 11-41  
 Algebraic operators, 4-8  
 Alternative nonlinear solver, 3-15  
 Array  
   definition, 4-13  
   display, 9-31  
   expressions, 4-17  
   functions, 4-19  
   index of element, 4-33  
   inputs and outputs, 5-7  
   maximum element, 4-21  
   minimum element, 4-22  
   output labels, 9-14, 11-18, 12-31  
 Auxiliary files, Chapter 15

## B

Bar charts, 9-38, 10-4, 12-41  
 Bicubic spline functions, 4-51  
 Bipolar transistor models, 7-2, 7-47, 7-49  
 Bounds on variables, 4-6  
 Built-in circuit responses, 9-3, 11-5, 12-18

## C

Capacitor models, 8-2, 8-38, 8-40, 8-42  
 Child program,  
   definition, 5-2

  file name, 5-7  
   templates, 5-14, 5-22, 5-28, 5-31  
 Circuit models, Chapter 6  
 Circuit statement, 6-18  
 Circulator model, 8-5  
 Clear the file buffer, 2-8  
 Colors of output labels, 10-3  
 Column of a matrix, 4-27  
 COM Datapipe  
   protocol, 5-15  
   template for child programs, 5-22  
 COMD Datapipe  
   protocol, 5,15  
 Conditional expressions, 4-10  
 Conjugate gradient optimizer, 11-33  
 Constant labels, 4-3  
 Continuation lines for text macros, 3-10  
 Control block, 3-13  
 Controlled sources  
   linear, 8-3, 8-4, 8-45, 8-46  
   nonlinear, 7-45, 7-46  
 Contours, 9-32, 9-40, 10-32, 10-35  
 Cooling ratio, 3-15, 11-38  
 Correlation  
   coefficient, 4-37  
   indices, 12-11, 12-13  
   matrix, 4-37, 12-11  
   scatter diagram, 12-51, 12-60  
 Coupled transmission line models, Chapter 8  
 CPU time, 11-43  
 Cubic spline functions, 4-49  
 Current labels, 6-16, 6-21, 6-34  
 Current sources, 6-7, 6-13  
 Current spectra of sources, 6-33  
 Curtice FET models, 7-10, 7-12  
 Curtice HEMT models, 7-31, 7-34  
 Customized  
   keywords, 6-58  
   linear models, 6-41  
   nonlinear models, 6-44, Chapter 7  
   responses, 6-36  
   statistical distributions, 12-14

## D

## Datapipe

- cascading datapipes, 5-33
  - child program, 5-2
  - COM protocol, 5-15
  - COMD protocol, 5-15
  - dialogues, 5-2
  - error flag, 5-12
  - error message, 5-12
  - DFD protocol, 5-25, 11-6, 11-24
  - FUN protocol, 5-24, 11-6, 11-21
  - general descriptions, Chapter 5
  - LINEAR protocol, 5-29, 6-56
  - OSA90 as child, Chapter 14
  - protocol, 5-2, 5-6
  - server, 5-2, 5-3, Appendix A
  - SIM protocol, 5-9
  - time out, 5-8
- DC currents of sources, 6-20
- DC responses, 6-19
- DC voltages and currents of ports, 6-19
- Default
- bounds on variables, 3-21, 4-6, 11-40
  - number of outcomes, 3-22, 13-4
  - optimizer, 3-23, 11-34
  - physical units, 3-26
- Demonstration examples, Chapter 16
- Device models, Chapter 7
- DFT
- frequency to time, 4-46
  - time to frequency, 4-45
- Diagnostic messages, Appendix B
- Diode models, 7-3, 7-7, 7-9
- Disable adjoint, 3-16, 11-41
- Discrete density function, 12-7
- Display all sweep and output labels, 9-21
- Distribution
- correlated, 12-11
  - exponential, 12-4
  - lognormal, 12-5
  - normal, 12-6
  - sample, 12-8
  - uniform, 12-3
  - user-defined, 12-14
- Draw types, 9-27, 10-4

## E

- Editor hot keys, 2-9
- EESof data file, 6-55
- Eigenvalues and eigenvectors, 4-34
- ELEMENT, 6-41
- Elements, 6-4
- Empipe, 2-4
- Empipe3D, 2-4
- Equality threshold, 3-17, 4-11
- Error functions, 11-19
- Estimation of the yield, 12-29, 12-44
- Examples, Chapter 16
- Exponential distribution, 12-4
- Exponential sweep, 9-8, 11-10, 12-23
- Expressions, Chapter 4

## F

## DFD Datapipe

- in the Specification block, 11-25
  - protocol, 5-25, 11-6, 11-24
  - template for child programs, 5-27
  - with parameter sweeps, 11-26
- FET models, Chapter 7
- File blocks
- Control, 3-13
  - Expression, Chapter 4
  - general description, 3-2
  - ImportData, 6-48
  - Model, Chapter 6
  - MonteCarlo, 12-15
  - Sample, 12-8
  - Specification, Chapter 11
  - Statistics, 12-11
  - Sweep, Chapter 9
  - Trace, 9-42, 11-45
- Find, 2-7, 2-9
- Formatted display, 9-23
- FREQ label, 4-4, 9-4, 11-6, 12-19
- FUN Datapipe
- in the Specification block, 11-22
  - protocol, 5-24, 11-6, 11-21



**G**

Generalized L1 objective function, 13-2  
 Generalized L2 objective function, 11-29  
 Goals, 11-16, 11-17, 12-29, 12-30  
 Gradients  
   FDf Datapipe, 11-25  
   perturbation scale, 3-24, 11-40  
   two-sided, 3-25, 11-39  
 Graphical Views, Chapter 10, 12-53  
 Graphics zoom, 9-24, 12-35  
 Group delay, 3-17, 3-23, 6-24  
 Gummel and Poon models, 7-47, 7-49

**H**

Hardware lock, 1-3  
 HARM keyword, 9-5, 11-7, 12-20  
 Harmonic and spectral frequencies, 6-26  
 Harmonic excitations, 6-8, 6-14  
 HarPE, 2-4  
 HBT model, 7-28  
 HEMT models, 7-31, 7-34, 7-37, 7-40  
 Highest harmonic, 6-26  
 Histograms, 12-41, 12-49, 12-56  
 Host acceleration files, 9-43  
 Huber objective functions, 11-30  
 Huber threshold, 3-18, 11-37

**I**

Ideal sources, 6-6  
 If - else, 4-10  
 Import data  
   EEsof data file, 6-55  
   ImportData block, 6-48  
   interpolation, 3-19, 6-51  
   port terminations, 6-12, 6-54, 6-57  
 Include files, 3-12  
 Index of array element, 4-33  
 Inductor models, 8-12, 8-41, 8-42  
 Initialization of child programs, 5-34  
 Inner product of two vectors, 4-24  
 Input file  
   blocks, 3-2, *also see File blocks*  
   general descriptions, Chapter 3

  templates, 3-3

Inputs to the child, 5-7  
 Installation, Chapter 1  
 Integer, checking, 4-9  
 Interrupt  
   simulation, 9-16  
   optimization, 11-42  
   yield optimization, 13-6  
 INTMOD, 6-27  
 Inverse of a matrix, 4-28, 4-31  
 Iterations per T, 3-19, 11-38

**J**

Jacobians, 3-20, 3-24

**K**

Keywords, 3-1, 6-58  
 Kurtosis, 4-39

**L**

Large coupler models, 8-20, 8-22, 8-24  
 L1 objective function, 11-28  
 L2 objective function, 11-28  
 Label names, 4-2  
 Large-change sensitivities, 11-46  
 Large-signal response labels, 6-31  
 Large-signal responses, 6-26  
 Legend of display, 10-8  
 LINEAR Datapipe  
   protocol, 5-29, 6-56  
   template for child programs, 5-31  
 Linear elements, Chapter 8  
 Log files, 15-2  
 Log scale plots, 9-26, 10-7  
 Logical conditions, 4-10  
 Lognormal distribution, 12-5  
 LU factorization, 4-30, 4-32

**M**

Macro  
   constants, 3-6, 9-10, 11-12, 12-25

- expressions, 3-7
- functions, 3-8
- subcircuits, 3-11, 6-58
- Materka FET model, 7-14
- Mathematical function library, 4-9
- Matrix, 4-14, *also see Array*
- Maximum error histogram, 12-49
- Mean value, 4-35, 12-6
- Menu options, 2-6
- Microstrip element models, Chapter 8
- Minimax objective function, 11-27
- MIXER, 6-27
- Model
  - linear elements, Chapter 8
  - Model block, Chapter 6
  - nonlinear elements, Chapter 7
- Monte Carlo
  - MonteCarlo block, 12-15
  - number of outcomes, 12-16, 13-3
  - MonteCarlo menu options, 12-32
  - simulation, 12-33
- Monte Carlo displays
  - histogram, 12-41, 12-56
  - maximum error histogram, 12-49
  - parametric display, 12-39, 12-54
  - run chart, 12-43, 12-58
  - scatter diagram, 12-51, 12-60
  - Views, 12-53
  - Xsweep display, 12-37, 12-54
  - yield, 12-29, 12-44
  - yield sensitivity, 12-45
- Multi-purpose child programs, 5-35
- Multiple references to the same child, 5-33

**N**

- N\_Spectra, 6-27
- NAN, 3-20, 4-4
- Newton iteration, 3-15, 3-21
- No\_default\_bounds, 3-2, 11-40
- Nodes, 6-3
- Non\_microwave\_units, 3-21, 3-27
- Nonlinear controlled sources, 6-46
- Nonlinear elements, Chapter 7
- Normal distribution, 12-6
- Number of histogram bins, 12-42, 12-56
- Number of decimal digits, 3-16

- Number of error functions, 11-20
- Number of iterations, 3-22, 11-35, 13-3
- Number of outcomes, 3-22, 12-16, 13-3
- Numerical append, 9-22
- Numerical display, 9-22, 12-36

**O**

- Objective functions
  - general descriptions, 3-23, 11-27, 11-37
  - generalized L1, 13-2
  - generalized L2, 11-29
  - Huber, 11-30
  - L1, 11-28
  - L2, 11-28
  - minimax, 11-27
  - one-sided L1, 11-31
  - sum of functions, 11-27
- Online Manual, 2-3, 2-7, 2-8, 2-9, 15-1
- Open 2-7, 2-9
- Optimization
  - general descriptions, Chapter 11
  - starting point, 11-44
  - variables, 4-6
- Optimizers, 3-23, 11-32
- OSA90 Installation Directory, 1-4
- OSA90 Display Menu, 9-15
- OSA90 Montecarlo Menu, 12-32
- OSA90 Optimize Menu, 11-32
- osa90msg directory, 1-4
- Oscillator, 6-61
- Outcomes, 12-3, 12-4, 12-5, 12-6, 12-9
- Output labels
  - Monte Carlo, 12-16, 12-17, 12-29, 12-30
  - optimization, 11-3, 11-4, 11-16, 11-17
  - simulation, 9-14,
- Output power, 6-31, 6-36
- Outputs from the child, 5-7

**P**

- Parameter labels, 9-13, 11-15, 12-28
- Parameter sweeps, 9-7, 11-9, 12-22
- Parameters defined by expressions, 6-5
- Parameters evaluated externally, 6-5
- Parametric display, 9-28, 12-39, 12-54

Parametric labels, 10-19  
 Parametric Views, 10-18  
 Perturbation scale  
   gradient, 3-24, 11-40  
   Jacobian, 3-20  
 Perturbed analysis index, 4-5  
 Physical units  
   default, 3-27  
   list, 3-26  
   non\_microwave\_units, 3-27  
 Physics-based FET models, 7-19, 7-23, 7-40  
 Piece-wise linear interpolation, 4-52  
 Polar plot Views, 10-29  
 Polar to rectangular transformation, 4-43  
 Ports, 6-11  
 Postprocessing responses, 6-36  
 Power spectrum, 6-26  
 Practice Session, 2-12  
 Predefined labels, 4-4  
 Preprocessor macros, 3-6  
 Preprogrammed models, 6-47  
 Print 2-7, 2-9  
 Product of  
   a matrix and a vector, 4-24  
   a transposed matrix and a vector, 4-25  
   two matrices, 4-25

**Q**

QR factorization, 4-34  
 Quadratic modeling, 3-22, 13-5  
 Quasi-Newton optimizer, 11-33  
 QUIET keyword, 6-49, 6-52

**R**

Radial stub model, 8-27  
 Random optimizer, 11-33  
 Raytheon FET model, 7-17  
 Rectangular to polar transformation, 4-44  
 Reference impedance, 6-11, 6-24, 9-6, 11-8, 12-21  
 Reference to imported data, 6-53  
 Report generation, 9-44  
 Resistor models, 8-38 to 8-42  
 Response labels, 6-18, 6-19, 6-22, 6-31  
 Response spectral indices, 6-35

Restart yield optimization, 13-7  
 Row of a matrix, 4-27  
 RREF keyword, 9-6, 11-8, 12-21  
 Run charts, 12-43, 12-58

**S**

S parameters, 6-22, 6-48, 6-56, 8-48  
 Sample block, 12-8  
 Save, 2-7, 2-9  
 Scatter diagram, 12-51, 12-60  
 Select a sweep set, 9-18, 9-29  
 Select the parametric label, 9-29  
 Sensitivity  
   Analysis, 11-46  
   display options, 11-47  
   selecting variables, 11-48  
   verifying optimized solution, 11-49  
   yield, 12-45  
 Sequence of calculations  
   impact of adjoint analysis, 3-16  
   Monte Carlo simulation, 12-33  
   optimization, 11-41  
   simulation, 9-16  
 Show downhill iterations only, 11-36, 13-4  
 Sign of error functions, 11-20  
 SIM protocol, 5-9  
 Simulation  
   general descriptions, Chapter 9  
   interrupt, 9-16  
   types, 9-3, 11-3, 11-5, 12-16, 12-18  
 Simplex optimizer, 11-33  
 Simulated annealing, 11-33  
 Single specifications, 11-4, 11-16  
 Skewness, 4-38  
 Small-signal responses, 6-22  
 Smith Chart View, 10-26  
 SMmodel, 11-57  
 SMproject, 11-53  
 Solving a linear system of equations, 4-30  
 Source  
   associated with ports, 6-13, 6-14  
   available power, 6-14  
   current sources, 6-7, 6-13  
   frequency, 6-8  
   voltage sources, 6-6, 6-13  
 Space Mapping, 11-51

## Specification

- for yield calculation, 12-16
- general description, Chapter 11
- in Views, 10-14
- single, 11-4, 11-16
- upper and lower, 11-4
- Spectra in rectangular form, 6-36
- Spectral index mapping, 6-29
- Spectral\_Freq, 6-27
- Spline functions, 4-49
- Standard deviation, 4-36, 12-6
- Starting point, 11-44
- Statements, 3-5
- Statistical
  - correlation, 4-37, 12-6, 12-11
  - distributions, 12-2
  - general descriptions, Chapter 12
  - Kurtosis, 4-39
  - mean, 4-35, 12-6
  - optimization, Chapter 13
  - samples, 12-8
  - skewness, 4-38
  - statistical parameters, 12-2
  - Statistics block, 12-11
- String labels, 4-41
- String outputs, 5-17
- Subcircuits, 3-11, 6-38, 6-58
- Sub-matrix, 4-15, 4-26
- Substrate, microstrip, 8-35
- Sub-vector, 4-15, 4-26
- SUM of array elements, 4-23
- Sum of functions, 11-27
- Super keywords, 6-51
- Sweep block, Chapter 9
- Sweep label
  - array indices, 9-12, 11-14, 12-27
  - exponential step, 9-8, 11-10, 12-23
  - intervals and points, 9-9, 11-11, 12-24
  - Monte Carlo, 12-16, 12-25
  - number of steps, 9-8, 11-10, 12-23
  - optimization, 11-3, 11-12
  - simulation, 9-10
  - uniform step, 9-7, 11-9, 12-22

## T

- Temperature, 3-18, 11-37
- Template for Datapipe child programs
  - COM, 5-22
  - FDF, 5-28
  - FUN, 5-14
  - LINEAR, 5-31
  - SIM, 5-14
- Template for input files, 3-3
- Termination of
  - optimization, 11-42
  - osa90, 2-20
  - yield optimization, 13-6
- Threshold, 3-18, 11-30, 11-37
- Time interval, 10-23
- Time out, 5-8
- Time-domain waveforms, 6-37
- Titles of Views, 10-7, 10-20, 10-24, 10-28
- Trace of variables, 9-42, 11-45
- Transformations
  - DFT, 4-45
  - polar to rectangular, 4-43
  - rectangular to polar, 4-44
- Transmission line models, Chapter 8
- Transposition of a matrix, 4-28
- Two-dimensional sweep, 9-11, 11-13, 12-26
- Two-ports responses, 6-25
- Two-sided perturbation, 3-25, 11-39
- Two-tone frequencies, 9-4, 11-6, 12-19
- Two-tone sources, 6-10, 6-15

## U

- Undo, 2-7
- Unformatted displays, 9-23
- Uniform distribution, 12-3
- Uninstalling, 1-5
- Unit of the phase, 6-23
- User comments, 10-12, 10-20, 10-24, 10-28
- User-defined
  - keywords, 6-58
  - linear models, 6-41
  - nonlinear models, 6-44, Chapter 7
  - responses, 6-36
  - statistical distributions, 12-14

**V**

Vectors, 4-13

**Views**

display scale, 10-7, 10-20, 10-24, 10-31

general descriptions, Chapter 10, 12-53

histogram, 12-56

Monte Carlo, 12-53

parametric, 10-18, 12-53

polar plot, 10-29

run chart, 12-58

scatter, 12-60

title, 10-7, 10-20, 10-24, 10-28

Visual, 10-32

waveform, 10-21

Xsweep, 10-2

Visualization, 9-32, 10-32

Voltage and current spectra of ports, 6-32

Voltage labels, 6-16, 6-21, 6-34, 6-44

**W**

Waveforms, 6-37

Waveform Views, 10-21

Weighting factors, 11-19

**X**

Xsweep display, 9-17, 12-37, 12-54

Xsweep Views, 10-2

**Y**

Y parameters, 6-22, 6-48, 6-56, 8-54

**Yield**

Monte Carlo, 12-29, 12-44

optimization, Chapter 13

optimization strategy, 13-7

restart optimization, 13-7

**Yield sensitivity**

display, 12-45

detect critical values, 12-45

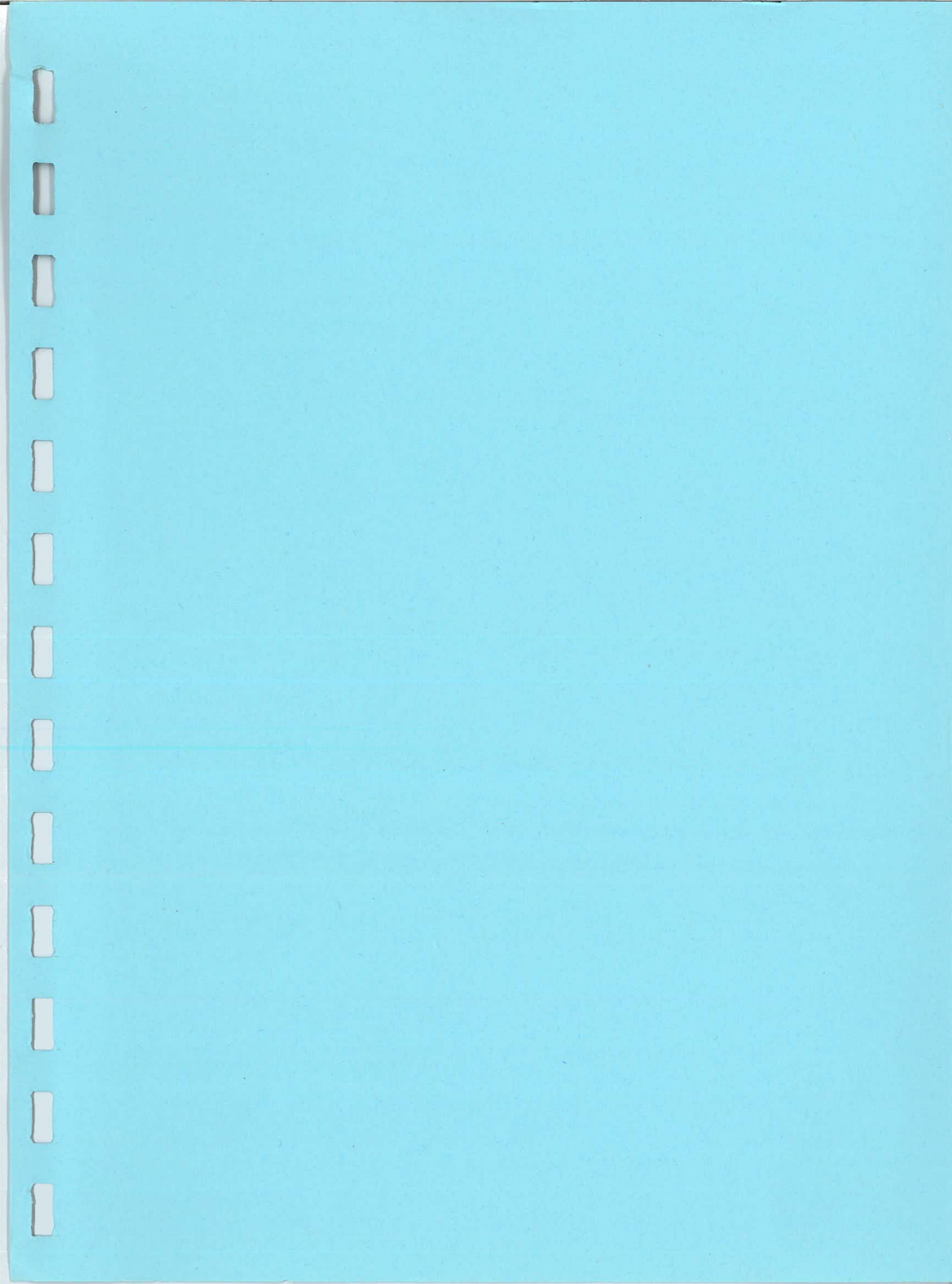
versus a design variable, 12-46

versus two sweep labels, 12-47

**Z**

Z parameters, 6-22, 6-48, 6-56, 8-60







**OSA**

**Dundas, Ontario, Canada**