AGILE USER MANUAL

REVISION: NEW VERSION EDITION

Date: April 2016

CONTENTS

Table of Figures

1 INTRODUCTION

AGILE is a computer program primarily used for the analysis, optimization and statistical analysis/design of linear and nonlinear systems. It contains internal models for components relating to microwave and RF circuit and system design and is designed to solve very generalized analysis problems. These networks may be of virtually any size or complexity. This is accomplished by defining networks in a nodal topology, making **all** "size" parameters dynamically allocated or adjustable at run-time, incorporating an extensive circuit component library and allowing the user to create their own components directly in AGILE language called NDL. AGILE is designed to be an *interactive* program. This version is a graphical user interface (GUI) version, that provides features in a windowed application. A summary of AGILE features is:

- **Advanced Network Descriptive Language (NDL)**. Allows network parameters to be defined in terms that relate algebraically.
- Parametric, hierarchical circuit or system description.
- AGILE is both a **circuit and system** level simulation and optimization tool.
- **True multi-port analysis** allowing networks of any size to be analyzed in a convenient way.
- **Unlimited** numbers of ports. frequencies. and all other attributes. AGILE allows circuits of any size or complexity to be analyzed.
- **Nonlinear circuit elements** may be included at any sub-level in the circuit.
- **Output expressions** allow network designers the ability to express the outputs as any algebraic combination of built-in output quantities.
- **Sub-modeling** allows users to create component models as any interconnection of AGILE circuit components or other sub-models.
- **Multi-port tabulated networks** allow fixed data to be included in analyses.
- **Graphical output** of AGILE as either rectangular or polar data plotting
- **Algebraically defined optimization goals** allow users to very flexibly create network performance objectives for use with optimization in either or both time and frequency domains.
- User may choose any of **several optimizers**.
- Advanced **time domain** features allow users to apply time domain signals to one or many ports and inspect either time domain or frequency spectra at any port.
- **Statistical features** include sensitivity and Monte-Carlo analysis, and a variety of meta-models and design of experiments.
- Advanced **noise-analysis** features for linear circuits. including handling of multiports and a unique correlated noise current generator component.
- Graphical user interface.

1.1 LANGUAGE FEATURES

AGILE allows *any* value to be defined as an algebraic expression. This feature allows network designers to express interrelationships between network parameters that often exist in analog networks. These expressions may be frequency-dependent, which further generalizes the ability to define network components and parameters. Values that describe a network need not be specified as constant values, but can be based upon expressions of other values. This permits, for example, that the value of a capacitor be assigned a value equal to one-half of the square-root of

some inductor. Network output parameters, such as S-. Y-. and Z-parameters, and load-dependent network parameters, such as PGAIN (power-gain), VSWR (Voltage Standing Wave Ratio), as well as many others are available as outputs. In addition, these outputs may be algebraically manipulated to form any output desired. These outputs may be real- or complex-valued. Thus, users of AGILE can directly calculate quantities such as the phase difference between the ports of a coupler, or compute the average magnitude of the reflection coefficient over each port of a twelve-way power divider, directly as an output of AGILE.

Optimization, the varying of values to achieve a goal, is done using multiple goal statements which are also algebraically assigned. There are several forms for goal definition using either complex- or real-valued expressions. A significant feature of AGILE is the ability to visualize these goals after definition, enabling network designers to verify the objective of optimization. These goals may then be combined with any other factors to create an error value for optimization.

Several features related to statistical analysis are available in the tolerance section. This includes the ability to define statistical properties of components and parameters and numerically compute sensitivity and output value distributions. This is taken a step further for ‘cost’ optimization which then applies optimization techniques to statistical results to find improved settings that result in better yield.

1.2 INSIDE AGILE

This version of AGILE runs on wither Windows or Linux platforms, making use of the Qt system for portability in its graphical user interface (GUI). AGILE can be configured to meet a variety of network analysis requirements. All of the parameters relevant to the configuration of AGILE are either user adjustable at run time or are automatically allocated, allowing networks of arbitrary size, number of frequencies, etc. to be analyzed and optimized.

1.3 USER-CREATED MODELS

AGILE allows users to create networks and embed them in another network, creating a *sub-model*. Sub-models may be incorporated into other sub-models, etc. allowing a very general, user-defined topology of networks to be created. Sub-models allow the network designer to create networks in a "building-block' fashion. Each functional piece of the overall network can then be created and modified until it meets specifications, before its inclusion into the entire system. Sub-models are also useful when several network designers contribute to a design; each designer can create a part of the overall design.

Parameters (values) may be passed between these sub-models. The sub-model serves as a user-defined component. which is arbitrarily constructed of circuit components or other sub-models. The user is able to create a sub-model as a function of parameters, and then "invoke" this model from within another network. When the sub-model is required in another network, the values at which the model is to be used is passed to the sub-model, the sub-model then uses these values to evaluate its own components, and then its nodes are available for connection in the requesting network. All of these features are accessible *directly in* AGILE language, with absolutely no programming whatsoever.

Even though AGILE is capable of analyzing multiple, interconnected networks, analysis of a single network is performed easily and directly. Simple commands are used to establish the network and list or plot simulation results.

1.4 TIME DOMAIN FEATURES

Many microwave circuit simulators provide frequency domain analysis results. AGILE also allows the circuit designer to apply a voltage or current *signal* defined in either the time or frequency domain to any port and inspect results in either (or both) the time or frequency domain(s). This feature is very generalized; any time domain signal is composed by the user using algebraic expressions as functions of the special variable `time`. Outputs may be observed in either the time or frequency domains. Please note that the non-linear features and analysis is harmonic balance related – this limits consideration to periodic signals – Agile is not a general time-domain simulator.

Optimization goals may be in the time domain as well, in fact optimization goals can be in both the time and frequency domains simultaneously. This means, for example, that the user may establish design criteria that minimizes the reflection coefficient at an input (frequency domain), while trying to achieve some time domain function at the output.

1.5 TOLERANCE ANALYSIS AND STATISTICAL FEATURES

AGILE allows the study of circuits and systems with *statistical properties*. Specifically, variables may be defined as distributions, rather than as fixed values. These variables are referred to as *statistical variables* or *variates* in this manual. A very general means for describing a statistical variable directly in NDL is supported, allowing for very generalized statistical circuit/system definition. Each statistical variable defined as such is *independent*, however, using the algebraic features of AGILE NDL, these variables can be combined to form *dependent variables*.

After defining a circuit with statistical variables. AGILE can then perform "Monte-Carlo" analysis (and/or sensitivity analysis). After an analysis of this type, the user has great flexibility in displaying the results - for example, histograms of both input variables and output results are supported.

1.6 NOISE ANALYSIS

AGILE incorporates some of the most generalized techniques used in the noise analysis of linear networks. Some of the features that a generalized approach enables are: setting temperature of sub-models individually and parametrically, handling multiport circuits directly in computing noise outputs. and allowing a unique current generator component which generates *correlated* noise currents. These features are implemented in such a way that they are used just as conveniently as normal network analysis.

1.7 USING THIS MANUAL

This manual is organized into nine chapters, plus appendices. It is recommended that the novice user read through chapter 4 before attempting to use AGILE – which is described in Chapter 9. There are Appendices as well that provide additional information and summaries. The Appendix has its own table of contents on page 133, see that for details. Other than this chapter, here is a breakdown of the chapters:

Chapter 2, *AGILE NETWORK DESCRIPTIVE LANGUAGE*, describes the basics of the AGILE Network Descriptive Language (NDL), the language with which we describe our networks to AGILE.

Chapter 3, *ADVANCED NDL*, describes additional NDL capabilities with the exception of optimization goal statements, statistical variable definition and time domain features. This chapter includes how to non-ideal port-

termination networks, the use of the sub-modeling abilities of AGILE, definition of global variables, and the use of various special variable types. In addition, the use of tabulated network data, including the PLANA format (measured data), is detailed.

Chapter 4, *THE CIRCUIT COMPONENT LIBRARY*, describes in detail all of the circuit components available at the present time, this library will certainly expand in future releases. For this reason, the circuit component "sheets" are presented at the end of the chapter to allow simple updating. After this section, it is probably a good idea to skip to Chapter 9 and try to use Agile some. Then come back and read about further topics.

Chapter 5, *OPTIMIZATION*, details the definition of goals, the figure of merit. and the use of the optimize command.

In chapter 6, *APPLIED SIGNALS AND TIME DOMAIN OUTPUT*, all topics relevant to time domain calculations are presented. This includes how to optimize in the time domain, a topic omitted from the optimization chapter. For users requiring only frequency domain simulation results chapter 6 may be skipped entirely.

Chapter 7, *STATISTICAL FEATURES*, explains how to define a variable such that its values are a distribution, rather than a fixed value during "Monte-Carlo" analysis. Also described in this chapter is how to invoke both sensitivity and Monte-Carlo analyses, as well as how to display the results of such analysis. Those not interested in statistical (Monte-Carlo) effects may still want to read the first section in that chapter, this concerns sensitivity analysis only.

Chapter 8, *NOISE ANALYSIS*, explains the use of the noise analysis features of AGILE, including enhancements to tabulated data format to describe noise parameters

In **chapter 9**, *USING AGILE*, the commands used to create and modify the network description, as well as commands to instruct AGILE to perform network analysis or plot data, etc. are described. After this chapter, you will be able to attempt some examples, being sufficiently familiar with AGILE NDL and the various commands. At that point, the details of the circuit component library have not been fully discussed, but you should still be familiar enough with this to be able to try using AGILE.

2 NETWORK DESCRIPTIVE LANGUAGE (NDL)

In order to analyze networks, the network designer must describe the network in terms that AGILE understands. The Network Descriptive Language (NDL) is the language that AGILE uses to describe networks; NDL consists of statements which define all aspects of the network such as the components used, the topology (interconnection of components) of the network, the external ports, the frequency points for analysis, the outputs desired, etc. All of these constituents are necessary in the definition of the network.

In this chapter we will begin to describe the NDL used by AGILE. We are not discussing the use of AGILE -- only the language used to describe networks in AGILE NDL. The commands used to direct AGILE operation will be discussed in chapter 3.

2.1 VARIABLE NAMES

AGILE uses *variables* extensively. AGILE variables are simply names which may represent values or circuit components. In later sections, we will see how to use variables in defining values and network parameters. Valid names of variables start with an alpha character (any letter), optionally followed by more alpha-characters or numeric characters (any digit). Note: other than letters, only the underscore character "_" is also considered an alpha-character. Some examples of valid and invalid names for variables are:

```
Valid      Invalid
x          x@3          -- contains a non alpha-numeric character
F5yz       34x          -- does not begin with an alpha
Val_47                  -- OK
```

Notice that either upper- or lower-case is acceptable; AGILE makes no distinction between cases; variable names are **case-insensitive**. This means, for example, that the variable `XyZ` is identical to the variable `xYZ` in AGILE NDL. NOTE: not all possible variable names specified with this convention can be used as variables. AGILE reserves the use of some character sequences for special purposes. These reserved names will be defined in the ensuing sections.

2.2 SIMPLE ASSIGNMENT STATEMENTS

Variables in AGILE have several uses. One use of a variable is to represent a quantity (value). An algebraic statement is used in the assignment of a value to a variable. The syntax used is almost identical to many other computer languages, but we are not using any programming in defining these variables -- we are simply using AGILE NDL. The syntax used is: variable name, followed by an equal (=) sign, followed by the expression with which the variable should be equated. Thus, some simple assignment statements would appear in AGILE as:

```
x = 3
another = 5.7
q37 = another - x
```

The evaluation of the expression on the right-hand side of the equal sign is assigned to the variable on the left-hand side. In the case shown, the variable `x` would take on the value *3*, the variables `another` and `q37` would evaluate to `5.7` and `2.7`, respectively. As shown in this example, variables may be defined in terms of other variables as well as constants.

2.3 ORDER OF STATEMENTS

AGILE defines a circuit or network in terms of a *collection* of statements. Since these statements may occur in any order, NDL might be referred to as non-procedural. Displayed are two fragments of AGILE NDL, both of which have the same meaning since order does not matter:

```
x = 3                          t1 = x + 10.7
t1 = x + 10.7                  x = 3
```

We are defining two variables, `x` and `t1`, their values would be `3` and `13.7`, respectively. In AGILE, we are defining a *static* definition representing the interrelation of quantities, rather than an ordered sequence of statements to be evaluated as in conventional programming languages. This means that each variable in AGILE may appear only once on the left-hand side of an assignment statement. AGILE will generate an error message[1] if statements such as:

```
x = 34.5          -- use of both of these statements is illegal
x = aquantity*PI  -- error: x cannot be defined twice
```

are encountered in NDL (`aquantity` and `PI` would be defined elsewhere). This is because the variable `x` can represent only a single value, it is not assigned values sequentially. Another situation which is illegal in AGILE is one where a variable is defined in terms of itself:

```
x = x - 1         -- variable may not be defined in terms of itself
```

Although statements such as these frequently appear in programming languages, they are illegal in AGILE.

2.4 ASSIGNMENT STATEMENTS, FUNCTIONS, AND EVALUATION

Although not formally introduced, we have already seen illustrations using some of the binary in-fix operators[2]

```
AND             Logical AND      } lowest precedence
OR              Logical OR
< <= >= > ==    Relationals
MAX             Maximum operator
MIN             Minimum operator
+               Addition
-               Subtraction
*               Multiplication
/               Division
^               Exponentiation   } highest precedence
```

These operators have a *precedence* as shown. Expressions involving operators of mixed precedence will be evaluated such that operations involving operators of highest precedence will be performed before operations of lower precedence. That is, the expression:

```
x = 45.8*32.5-1.0e3/25.0
```

[1] See appendix for alphabetic list of error messages
[2] AGILE uses infix notation for MIN and MAX operators.

will evaluate to `1448.5`. Also note the use of scientific notation, namely `1.0e3` = 1.0 x 10^3. Operations of equal precedence are evaluated left-to-right (left-associative), with the exception of exponentiation, which is evaluated right-to-left (right-associative):

```
y = 20/5*4          evaluates as (20/5)*4=16, not as 20/(5*4)=1
z = 3**3**3         evaluates as 3**(3**3) ≅ 7.62e12, not as (3**3)**3=19683
```

Parenthesis may be used to alter the evaluation order. This is illustrated by contrasting:

```
x = 45.8*(32.5-1.0e3)/25.0
```

which evaluates to `-1772.46`, with the previous evaluation of `x`. An example containing binary operators from the different precedence levels, and the resulting grouping is:

```
z = x-1 MAX 6-y**2 >= 17        groups as:  z = (((x-1) MAX (6-y**2)) >= 17)
```

In addition to binary operators, AGILE variables can be assigned values through functions. A wide range of functions is available, since NDL functions are frequently added, a complete list and description of functions is contained in the appendix. Here is an abbreviated table of functions (see Appendix for complete list):

Name	Comment		Range (* = system dependent)
ABS(x)	Absolute Value		all x
COS(x)	Cosine (argument in radians)		-4096π< x < 4096π*
SIN(x)	Sine (argument in radians)		-4096π< x < 4096π*
TAN(x)	Tangent (argument in radians)		-4096π < x < 4096π*
LOG(x)	Natural Logarithm		x > 0
LOG10(x)	Logarithm base 10		x > 0
SQRT(x)	Square-Root		x > 0
COSH(x)	Hyperbolic Cosine		x < 88.8*
SINH(x)	Hyperbolic Sine		x < 88.8*
TANH(x)	Hyperbolic Tangent		x < 88.8*
FLOOR(x)	Next Lowest Integer Function		all x
ARCCOS(x)	Arc-Cosine		-1 < x < 1
ARCSIN(x)	Arc-Sine		-1 < x < 1
ARCTAN(x)	Arc-Tangent		all x
EXP(x)	Natural Exponentiation e^x		-88.8 < x < 88.8*
APHASE(x)	Positive Phase Difference =	x if x ≥ 0 x+360 if x < 0	all *x*

As examples, the following AGILE NDL fragment:

```
o1 = 1/sqrt(5)
o2 = 57.29578*COS(o1) + SQRT(10.0e-1)
o3 = (1.0 - o2)/ABS(1.0 - o1)
```

would set `o1` to `0.44721`, `o2` to `52.661`, and `o3` to `-93.456` (approximately). Note the use of a negative exponent `10.0e-1` in a constant (i.e., `10.0e-1 = 1`).

A thorough discussion of variable evaluations is beyond the scope of this manual. The syntax, operator precedence, and correct range for functions for AGILE assignment statements are similar to that of many languages such as C.[3] For users not familiar with C expression syntax, hopefully enough information was provided in this section for you to be able to exploit the algebraic expression syntax inherent to AGILE.

2.5 FREQUENCY STATEMENTS - LINEAR NETWORKS

AGILE uses the frequency statement when performing an analysis of linear networks. The harmonic statement is used to define frequencies and harmonics of interest for nonlinear circuits, described in the next section. There are three forms of statements used to define frequencies in AGILE -- a single-frequency form, a start-stop-step form, and an exponentially swept form. The syntax follows the forms:

```
10 GHz
12 15 1 GHz
10 20 6 EXP Ghz
```

In this case, the first line specifies 10 GHz and the second line specifies start at 12 GHz, step by 1 GHz, and stop at 15 GHz. The third line specifies a start of 10 GHz, a stop of 20 GHz with 6 points spaced exponentially apart (these are approximately 10, 11.29, 12.86, 14.78, 17.13, 20). Thus, the user has specified that analysis should be performed at 10, 11.29, 12, 12.86, 13, 14, 14.78, 15, 17.13 and 20 Giga-Hertz. The user may specify as many frequency-defining lines as desired; The frequencies indicated by each line are combined to form the complete set that will be used for analysis. Any duplicate frequency points specified will be analyzed just once. The units available for frequency-definition are:

`HZ`	***Hertz***
`KHZ`	***1,000 Hertz***
`MHZ`	***1,000,000 Hertz***
`GHz`	***1,000,000,000 Hertz***

These four units allow specification of frequency in a form which is most convenient to the user. As specified previously, AGILE makes no distinction between upper- or lowercase, so `mhz, MHZ, MHz` mean the same thing. You are free to mix statements with different units, but not on the same line:

```
10 20 2 GHz          means 10, 12, 14, 15, 16, 18, and 20 GHz
15000 MHz

10 GHz
10000 MHz            all mean 10 GHz
10e9 HZ
```

As mentioned in section 2.1, *VARIABLE NAMES*, not all sequences of characters can be used as variable names. Frequency units, as well as the functions named in section 2.4 (i.e. SQRT), are examples of names not to be used

[3] with the exception of the names for functions, and the MIN and MAX operators.

as variable names. In general, any of the special character sequences mentioned may not be used as variable names -- they are reserved words. This means that:

```
sqrt = 9          are both illegal
x = 6 - HZ
```

are not allowed and will generate an error message. In this example, `sqrt` and `HZ` are used as variable names; they may not be, they may only be used as previously specified.

2.6 HARMONICS STATEMENT - NONLINEAR NETWORKS

While the frequency statement described above is used to define frequency points for a linear circuit or system, AGILE uses the harmonics statement to define frequency and harmonic points. There are two basic formats for the harmonics statement, depending upon the number of base frequencies used. The first format is based on the fact that the number of base frequencies is one:

```
harmonics freq-unit 1 base-freq harmonic-list

    where:
           freq-unit      is HZ, KHZ, MHZ, GHZ
           base-freq      is a number
        harmonic-list     is a list of ints >= 0

    examples are:
        harmonics ghz 1 12 1 2 3  which means a base freq of 12 ghz
                                        with 1st (12 ghz), 2nd (24 ghz) and
                                        3rd (36 ghz) harmonic points.
         HARMONICS ghz 1 10 0 1 4 which are 0th (DC), 1st (10 ghz) and 4th (40 ghz) points.
```

Note that the integer list may contain the term 0, which implies the D.C. term. In nonlinear analysis, the DC biasing of the devices occurs whenever the 0th harmonic is present, and this is a balanced harmonic (DC balancing occurs). Otherwise arguments in the component model set and fix the biasing.

The second form of harmonics statement simplifies the generation of mixing products for harmonic balance analysis. In this case, "n" base frequencies may be specified and instead of defining a list of harmonics of interest, the user specifies the order of mixing and lower and upper frequency limits. Examples are:

```
harmonics ghz 2  12 12.5   3   0.1 30
       which states that there are 2 base freqs (12,12.5), and we want to generate 3rd-order
       mixing terms >= 0.1 and <= 30 (GHz)  these are: 0.5, 11.5, 12, 12.5, 13, 24, 24.5, 25
```

Important notes concerning the HARMONICS statement: the harmonics statement is used to generate the frequencies and harmonics of interest, while the signal statement is used to define the applied signals (sections 6.2 and 6.3). Since these are independent, user controlled inputs, AGILE does not "know" whether they are "consistent". For example, the user might apply a 12 GHz signal to the circuit (using the signal statement), but, using the harmonic statement, define a base frequency of say 11 GHz. Very poor (erroroneous) results will be computed in this case. It is the user's responsibility to properly select correct harmonics and signal frequencies. Although as many frequency statements (linear) may be used as desired, **only one harmonic statement is allowed**. A final note, the harmonics statement, if present, will "override" the linear frequency statements even for a linear circuit.

For either harmonic form, there is a further generalization allowed, that is the ability to specify the first base frequency as being *swept*. This is called an F1SWEEP in AGILE. As an example:

```
harmonics ghz 1 [12:18:1] 1 2 3
```

which means a nominal base freq of 12 ghz with 1st (12 ghz),
2nd (24 ghz) and 3rd (36 ghz) harmonic points. This however is "sweepable"
using the LIST/PLOT VS command described in 9.9.2

In this case, nonlinear analysis will occur nominally at 12 GHZ (and at the 2nd and 3rd harmonics), however, there is a command to make AGILE sweep this base frequency through the range specified. For example, at the next point of the sweep the frequencies of analysis will be 13, 26, and 39 GHz.

This format also is applicable to the "mixing products" form, however, only the first frequency may be swept. Extending the previous example:

```
harmonics ghz 2 [12:16:1] 12.5    3   0.1 30
harmonics ghz 3 [12:16:1] 13.5 13 3   0.1 30
```

In the first case above, the first mixing frequency may be swept from 12 to 16 GHz in steps of 1 GHz. Note that the second frequency 12.5 GHz remains fixed. In the situation above, we are comparing two harmonics statements – remember that only one can be put in a network at a time. A special variable F1BASE is used to keep track of this sweeping base frequency. This will be described further in section 9.9.2.

2.7 FREQUENCY-DEPENDENT VALUES

The reason for introducing the frequency definition statements above was to complete the discussion of variable evaluations. Any variable may be assigned a value which is *frequency-dependent*. The reserved word:

```
FREQ
```

represents frequency in expressions. The value used for FREQ is ***always* in *Hertz***, regardless of the units used to define frequency. In this example:

```
10 20 5 GHZ
o1 = FREQ/1e9
O2 = LOG10( freq ) + 2.0
O3 = 01-1
```

the values that o1, o2 and o3 would (approximately) take on at each frequency would be:

```
Frequency   O1      O2      O3
10 GHz     10.0   12.000  9.0
15 GHz     15.0   12.176 14.0
20 GHz     20.0   12.301 19.0
```

We will see that frequency-dependent values may be used to define any value, such as capacitance values, dielectrics of transmission lines, the magnitude of the transconductance of a voltage-controlled current-generator, etc.

The symbolic variable:

```
NUMFREQ
```

Will provide the number of frequencies defined. In the example above `10 20 5 ghz,` the value of `NUMFREQ` as used in NDL would be 3.

2.8 RANGE VARIABLE ASSIGNMENT STATEMENTS

Another type of assignment may be made to variables, namely a variable may be assigned a range of values rather than a single value. The syntax is: variable name, followed by an equal sign, followed by a number, followed by a left-bracket ([), followed by two or three expressions separated by colons, and finished with a right-bracket (]). When the three number form is used the variable becomes a *range with step variable*, the two-number form is simply called a *range variable*. Some examples are:

```
Caprange =  45 [  30 : 60.5 ]
Range2   = -20 [ -20 : 20 : 10 ]
```

In the example, `Caprange` is a simple range variable, `45` is referred to as the *current-value*, the value `30` is referred to as the *start-value*, and the number `60.5` is referred to as the *stop-value*. Note that spaces around the bracket characters are not necessary and are used for clarity. The range with step variable, `Range2`, has a current-value of `-20`, a start-value of `-20`, a stop-value of `20`, and a step-value of `10`. For range variables, the current-value should be specified as a value between the start- and stop-values, *inclusive*. For a range with step variable, the step-value should be an integer division of the difference of the stop- and start-values, and the current-value should be (but doesn't have to be) "on one of the steps". The first example below is illegal, while the second and third line are allowed but not recommended:

```
Wrong1 = 10 [ -10: 0 ]          illegal: current-value not in range -10 to 0
Wrong2 = 10 [ 10: 20: 45 ]      not recommended: step-value does not divide (20-10)
```

The bracketed parts of the range with step variable need not be fixed constants, and may be any non-frequency dependent expression. However, the current-value must be a constant. Correct illustrations of this are:

```
ystep = 2 [ 2: 4: 2 ]
X1 = 12 [ 0, 20, ystep ]
x2 = 40 [ (x1/2)**2, (x1/2)**3 ]
x3 = x2 + 10
x4 = 18.43 [ sqrt(x3), sqrt(x3)+20 ]
```

In general, AGILE uses the current-value when evaluating variables based upon a *rangetype* variable. Range variables may be used freely in the definition of network components and other variables, and as many range variables as desired may be defined. There are instructions which will inform AGILE to "sequentially" evaluate variables or do network analysis at each point of a range with step variable. These instructions will be discussed in section 9.9.2, *DATA OUTPUT -- {LIST/PLOT/DISPLAY}*. The optimization capabilities of AGILE also use range variables, these become the variables in the optimization. The use of range variables in this context are discussed in chapter 5, *OPTIMIZATION*.

2.9 CIRCUIT COMPONENT STATEMENTS

Circuit components many times require several parameters in their definition. All components are defined through AGILE *circuit component functions*. Circuit component functions, or just components, need several parameters

(arguments) in their definition. For example, a transmission line might be defined by its characteristic impedance and electrical length at a particular frequency. A component such as this is defined with a syntax such as:

```
T1 = TRL(Z = 50 ohms, EL = 50 deg, CF = 1.5 GHz )
```

In this example, `TRL` is the circuit component function or just component, `Zo, EL,` and `CF` are circuit component parameters or just component parameters, `ohms, deg` and `GHz` are units, and `T1` is a user-named component. A complete description of available components and their parameters is presented in chapter 4, *THE CIRCUIT COMPONENT LIBRARY* and the associated separate document that defines models. We are presently emphasizing the syntax and integration of these components into AGILE NDL.

Component parameters may be specified in arbitrary order with the same meaning. Therefore, these two:

```
T1 = TRL( Z = 50 ohms, PL = 15 mils, ER = 2 )
                          is the same as
T1 = TRL( PL = 15 mils, Er = 2, Z = 50 ohms )
```

have exactly the same effect. There are other points to notice in this example. Notice that `ER` is in mixed case in one and in all upper-case in the other, keeping with our case insensitivity rule. Also notice that the component parameter `ER` has no units, as would be expected since `ER` represents relative dielectric constant. Therefore, the general form for components is *component function*, followed by an `(`, followed by a *component parameter list*, and completed by a `)`. The component parameter list is made up of one or several parameter phrases, separated by commas (which are optional). Each component parameter phrase is a *component parameter*, followed by an equal sign (=), followed by an expression and optionally concluded with a *unit*. The necessity of units depends upon the *component parameter* and whether or not it requires a unit. In the above example, leaving the *ohms* off of the phrase for `Z` would default it to *ohms* anyway. Adding a unit to the `ER` phrase would result in an error. Again, chapter 4 will deal with the details of the circuit component library. In any event, you may choose from among any suitable units allowing definition of each component parameter in an arbitrary scale. For example, physical length (`PL` above) may be defined in: `uin` (micro-inches), `mils` (thousandths of an inch), `in` (inches), `ft` (feet), `um` (micro-meters), `mm` (milli-meters), `cm` (centi-meters), `m` (meters). In the example below, all of the physical lengths (`PL`) would be one meter:

```
... , PL = 1 m , ...
... , PL = 100 cm , ...
... , PL = 3.937009e7 uin , ...
... , PL = 3.28084 ft , ...
          and so forth
```

The arguments in the expression part of the component parameter phrases need not be restricted to constants as in the examples given so far; they may be *arbitrary, frequency dependent expressions*. This example illustrates this point:

```
x3 = 8.2
T1=TRL(Z= .47+sqrt(x3) Kohms, ER= 2.2+1/x3, PL=SQRT(freq/1e9) cm)
```

of course, `freq` represents frequency in Hertz. Note that variables, in themselves, are not assigned units, but units are assigned in component parameter lists. The first case below is illegal while the second case shows the correct method:

```
x = 8.2 mils
T1 = TRL( • • •, PL = x,• • • )          these statements are illegal
T2 = TRL( • • •, PL = x mils,• • •)

x = 8.2                                  these are correct
T1 = TRL( • • •, PL = x mils,• • • )
```

In the first group of lines, the definition of `T2` is syntactically correct, but the definition of `x` is illegal. Since `T2` uses `x`, the definition of `T2` illegal. The second group of lines shown is the correct method for achieving that result.

This is a good time to discuss blank characters (spaces) in the input lines. In general, spaces are not required, but can be used to "beautify" the input. The next two examples are identical as far as definition of components is concerned:

```
T1 = TRL( Z = 50 ohms, Er = 2, PL = 45 mils )
                is the same as
T1=TRL(Z=50ohms,Er=2,PL=45mils)
```

Either of these will be understood correctly by AGILE, however, there are cases when spaces are necessary. Consider these three cases:

```
T1 = TRL( Z = 50 ohms, Er = 2, PL = xlength mils )
    and
T1 = TRL( Z = 50 ohms, Er = 2, PL = xlengthmils )
    and
T1 = TRL( Z = 50 ohms, Er = 2, PL = (xlength)mils )
```

we have changed the `PL` parameter to a variable `xlength`. The first instance will be correctly understood, but the second case will not. In the second case above, AGILE will consider the expression part to be a single variable `xlengthmils` and complain that the units are missing (since it happens that units are required for PL) in attempting to define physical length. Obviously in this case, spaces in the input NDL *are* relevant. The third case above is acceptable, using parenthesis to delimit the expression part of the component parameter phrase. The "tab" character is equivalent to a space in NDL descriptions. A concise description of when spaces are and are not needed is not necessary at this time, suffice it to say that using spaces to separate the various parts of the language is a good idea and improves readability anyway.

2.10 ENVIRONMENT STATEMENTS

Many times in the definition of networks, the component parameter lists for several components might be very similar, differing only in maybe one or two phrases. Take for example a case where four transmission lines are defined:

```
T1=MSTRL(HT=40 mils,W=10.1 mils,ER=10,PL=100 cm)
T2=MSTRL(HT=40 mils,W=13.5 mils,ER=10,PL=100 cm)
T3=MSTRL(HT=40 mils,W=6.74 mils,ER=10,PL=100 cm)
T4=MSTRL(HT=40 mils,W=14.8 mils,ER=10,PL=100 cm)
```

The only difference in these component definitions is in the width parameter. In cases such as this, it is convenient to be able to define an *environment* in AGILE. Environments allow several component parameter phrases to be

defined into a single name, called a *user-named environment*, just called an environment for short. An example syntax for environment definition is:

```
EV1 : HT = 40 mils, ER = 10
```

defining the user-named environment `EV1`. The environment being defined is `EV1`, followed by a colon, followed by *any* component parameter list. This environment could then be used to simplify the four definitions above:

```
EV1 : HT = 40 mils, ER = 10
T1 = MSTRL( ENV=Ev1, W=10.1 mils, PL=100 cm)
T2 = MSTRL( ENV=Ev1, W=13.5 mils, PL=100 cm)
T3 = MSTRL( ENV=Ev1, W=6.74 mils, PL=100 cm)
T4 = MSTRL( ENV=Ev1, W=14.8 mils, PL=100 cm)
```

The definition of the environment `Ev1` is shown as the first line. The "invocation" of the environment is done with the `ENV = ev1` syntax within each of the component definitions. We have arbitrarily chosen not to include `PL` in the environment even though it could have been since it is constant throughout the definitions. This feature provides several benefits, including a reduced amount of typing and ease of change. That is, by changing the `HT` specification in the environment, `Ev1`, that change would be reflected throughout all components using that environment.

Arbitrary frequency-dependent expressions may be used in the expression part of the component parameter phrases in environment definition. This fragment would result in `T1, T2, T3, T4` having a similar definition to those above, the difference being that we changed the height `HT` specification to a frequency-dependent value. We are also using two variables, `xht` and `die`, to aid in the definition:

```
xht = 40
die = 10
EV1 : HT = xht - freq/1e9 mils, ER = die
T1 = MSTRL( ENV=Ev1, W=10.1 mils, PL=100 cm)
T2 = MSTRL( ENV=Ev1, W=13.5 mils, PL=100 cm)
T3 = MSTRL( ENV=Ev1, W=6.74 mils, PL=100 cm)
T4 = MSTRL( ENV=Ev1, W=14.8 mils, PL=100 cm)
```

As many environments as needed may be used, there is no restriction to just one. This example demonstrates this:

```
xht = 40
die = 10
EV1 : HT = xht mils, ER = die
Ev2 : PL = 10*die cm
T1 = MSTRL( ENV=Ev1, W=10.1 mils, Env = Ev2)
T2 = MSTRL( ENV=Ev1, W=13.5 mils, Env = Ev2)
T3 = MSTRL( ENV=Ev1, W=6.74 mils, Env = Ev2)
T4 = MSTRL( ENV=Ev1, W=14.8 mils, Env = Ev2)
```

The result is again the same as for the four transmission lines shown previously. The only difference between the two previous fragments is that we have used two environments in the definition of the transmission lines in the last instance.

As many environments as necessary can be defined, however, there is one restriction on the use of environments. An environment may not contain another environment; that is, environments don't "nest". This restriction may be eliminated in future versions of AGILE as the need arises.

Environment specifications may be placed anywhere in the component parameter list. Remembering that AGILE NDL is *non-procedural*, environments may be defined before or after their use. Revisiting our four transmission lines for the last time:

```
xht = 40
EV1 : HT = xht - freq/1e9 mils, ER = die
T1 = MSTRL( ENV=Ev1, W=10.1 mils, Env = Ev2)
T2 = MSTRL( W=13.5 mils,ENV=Ev1 , Env = Ev2)
die = 10
T3 = MSTRL( env=ev1, Env = Ev2, W=6.74 mils)
Ev2 : PL = 10*die cm
T4 = MSTRL( env=EV1, w=14.8 MILS, env = EV2)
```

still generates the same result.

2.11 CONNECTION STATEMENTS - NODE NUMBERS/NAMES

So far we have shown how to define components, either lumped or as component functions but have given no indication of how to connect these together to form a network. AGILE uses a *nodal* representation in the connection of components. A nodal form of topology allows any component to be connected anywhere in the network. Each circuit component, as well as each lumped component, has an *inherent* number of *nodes* associated with it. For example, all of the lumped components have two nodes and all of the transmission lines have three nodes, this is a physical property of these electrical components. The diagram below graphically illustrates this:

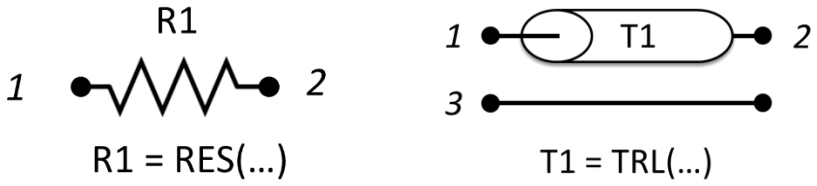
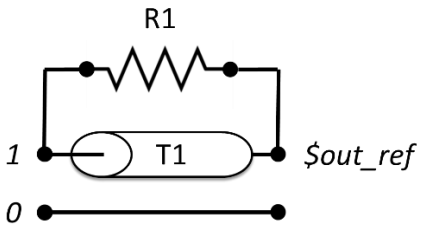


Figure 2.1 Illustration of a resistor and transmission line

In this case and the ensuing examples, the lower dots "…" represent a component parameter phrase. Chapter 4 and the separate model document will formally define all of the components, the number of nodes that each has, etc., at this point we wish to define how syntactically NDL *connects* components.

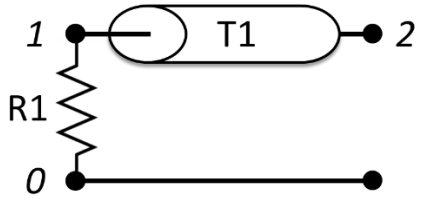
Each circuit component node is connected to a *network node*. Each network node is represented by an integer greater than or equal to zero or an alphanumeric string beginning with a dollar sign ($). We connect components to nodes using an AGILE NDL *connection statement*. The form of the connection statement is: *component-name* followed by *nodes*, or *component-definition* followed by *nodes*. The number of nodes required depends upon the component itself, that is, how many inherent nodes it has. Two example connection statements are:




```
R1 1 $out_ref
T1 1 $out_ref 0
```

Figure 2.2 Connection Statements

In this case we have defined that the component `R1` should be connected to network nodes `1` and `$out_ref`, and that the component `T1` be connected to nodes `1, $out_ref` and `0`. Notice in this example that we have not shown what `R1` or `T1` are, just specified connections for them. Indeed, from just these two connection statements we cannot determine whether the number of connections specified for each of these components is correct. In fact, we do not even know if `R1` and `T1` are components, or are even defined at all -- *the connection statement only defines connectivity to the network*. However, by showing the definition and connection of the resistor `R1` and transmission line `T1,` we could get:

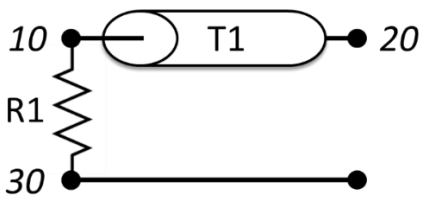



```
R1=RES(…)
T1=TRL(…)

R1 1 0
T1 1 2 0
```

Figure 2.3 Connecting a resistor and transmission line

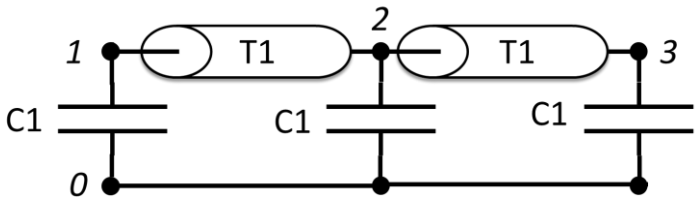
The next example emphasizes the fact that node numbers are arbitrary non-negative integers. This is the same as the example above except that the node numbers are different:




```
R1=RES(…)
T1=TRL(…)

R1 10 30
T1 10 20 30
```

Figure 2.4 Connecting a resistor and transmission line again

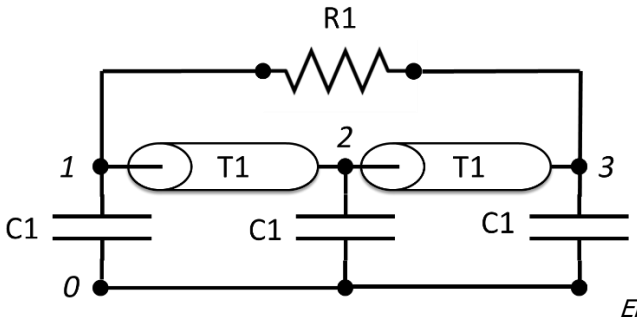



```
C1 1 0
T1 1 2 0
C1 2 0
T1 2 3 0
C1 3 0
```

Figure 2.5 Using many connection statements, assuming C1 is a capacitor and T1 is a transmission line.

In the previous examples, each of the defined components was connected just once (in a single connection statement). However, any component may be connected as many times as you wish, simply by specifying multiple connection statements. This can be illustrated by:

The *order* of these statements is irrelevant, again since AGILE NDL is non-procedural. Thus, both of the NDL fragments below in Figure 2.6 generate the same topology:

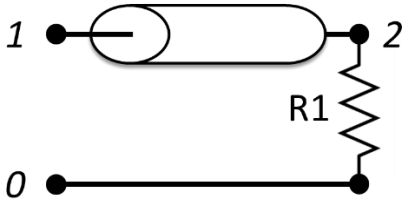



```
C1 1 0          C1 1 0
T1 1 2 0        C1 2 0
C1 2 0          C1 3 0
T1 2 3 0        R1 1 3
C1 3 0          T1 1 2 0
R1 1 3          T1 2 3 0
```

Either side has the same meaning

Figure 2.6 Order-Independence of connection statements, assumes C1 is a capacitor, R1 is a resistor, and T1 is a transmission line.

In the examples shown so far, we have defined user-named components and then connected them together using connection statements. There is another form of connection statement, namely the *immediate* connection statement. The definition of an immediate connection statement allows the definition of a component and its network connection to be done simultaneously (on the same line). The immediate connection statement has a syntax as:




```
TRL(...) 1 2 0
R1 = res(...) 2 0
```

Figure 2.7 Immediate connections.

The immediate form of the connection statement is convenient when you do not wish to name components or use them repeatedly. Note however, that both forms may be forms may be used at the same time. In the case of the resistor in Figure 2.7, it is immediately connected and named (R1) at the same time. The non-immediate form allows a flexible method for network construction, by allowing the definition and connection to be done separately. By making a single change in the component description, we change all of the instances where it is connected. Consider the case in Figure 2.6, by changing a definition for `C1`:

```
C1 = CAP(c=47 pF)      to:    C1 = cap(c=470 pF)
```

would effectively change three capacitors in the network.

The connection statement is more generalized than implied by the above, the general syntax is:

```
[ ref-name = ] component-name parameters node-list
   OR
[ ref-name = ] component-name node-list parameters      WHERE:  parameters are in ()
```

which allows the node list to be after the component name (like in SPICE and some other programs) or at the end. This is enabled by enforcing that the parameter list be enclosed in parenthesis (not like SPICE). Remember from the previous section that the parameter list can have commas or not. Some examples are:

```
TRL 1 2 0 (Z=50 PL=10 mils, ER=2)
RES 2 3 (r=10)
C1 = CAP 2 3 (C=6)
C1 4 5
L1 = IND (L=5) 2 5
```

The AGILE NDL method for defining components gives great flexibility to the network designer, allowing definition of user-named components and connections independently. Although this is a powerful ability, it is not *forced* -- you may specify all aspects of the component and its connection simultaneously if you wish with the immediate connection form. Connections of immediate or user-named components may be freely used in combination, allowing some components to be defined as named components and others to be just connected.

2.12 PORT STATEMENTS

Having shown how to define network topology through connection of components, ***ports*** define where external connections are made. Analysis of microwave/RF networks is done on the basis of ports. S-, Y- and Z-parameters are defined in terms of the ports of the network, where each port represents a *pair of network nodes*. In AGILE we assign a *port name or number* to a pair of nodes using a *port definition statement*. A port definition statement defines a port name and the pair of nodes associated with it. The figure below illustrates its use:

```
#1 1 0
#out $drain 3
```

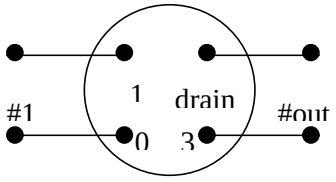


Figure 2.8 Defining Ports in NDL

Port `1` is defined as nodes `1` and `0`, and port `out` is defined as nodes `$drain` and `3`. You may arbitrarily assign port names to pairs of nodes, the port numbers need not be sequential or in order. We will define how these port names are used in relation to the analysis output results in the next two sections, *OUTPUT VARIABLES* and *OUTPUT STATEMENTS*. The case above would be termed a two-port network, since it obviously has two ports. In AGILE, networks may have any number of ports. In the two-port example above, four unique nodes (`0,1,$drain,3`) where used in the definition of two ports. The next example shows a three-port network being defined using only four nodes:

```
#1 1 0
#2 2 0
#3 3 0
```

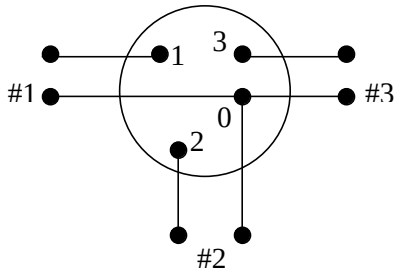


Figure 2.9 Defining a three-port network.

In this case, all three ports share a common reference node `0`. AGILE allows any or all ports to be based upon a common reference, but this is not necessary. To illustrate this point, we may define a three-port network as:

```
#1 1 2
#2 3 4
#3 5 6
```

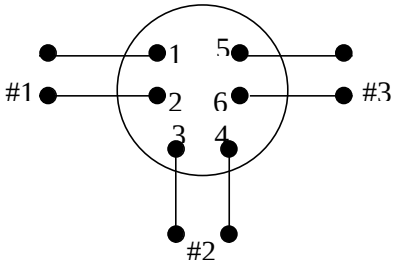


Figure 2.10 Defining a three-port network with six nodes.

AGILE allows many configurations in the definition of ports, but there are some constraints. Since AGILE defines ports as pairs of network nodes, at most 2*N* nodes of the network nodes are used in defining an *N*-port network. While 2*N* of the network nodes are used in defining an *N*-port at the maximum, at least *N+1* unique nodes are necessary. In addition, each port must have at least one node which is unique to it. Thus, each port defined for a network must have at least one unique network node which is associated with only that port. The example below is illegal:

```
#1 1 0      is illegal
#2 0 1
```

because an attempt is made to define port 2 as the "inverse" of port 1. Another example of illegal port definitions is:

```
#1 1 0
#2 1 2      is illegal
#3 2 0
```

because only three nodes are being used to define three ports. In such cases where less than *N+1* nodes are used and each port does not have a network node with which it only is associated, the network parameters, which are port-relative, are not independent of each other.

The form of the port definition statement shown allows only the port number and a pair of nodes to be specified. In general, a *port-termination network* may be specified for each port. However, in this simple form of the port definition statement, the *system impedance*[4] is assumed to be attached to each port. Another form of the port definition statement exists where an entire AGILE network may be specified -- this is described in Chapter 3, section 4, *NON-DEFAULT PORT TERMINATIONS.*

2.13 OUTPUT VARIABLES

An *output variable* in AGILE NDL is a variable which depends upon the result of network analysis. Outputs fall into three broad classes: *network* outputs, *port termination dependent*, and *voltage/current* outputs. The network outputs ***do not depend*** upon the *port terminations* presented at each port, only upon the network itself. The port termination-dependent outputs specifically depend upon the port termination networks at each port. Examples of currently implemented network outputs are: S-, Y- and Z-parameters. Examples of port termination-dependent outputs are *Voltage Standing Wave Ratio* and *Power Gain*.

The previous section, *PORT STATEMENTS*, illustrated the method used to define port names and their associated node pairs, namely, the port definition statement. Output variables in AGILE relate to these defined ports. Basically, to request any particular output, the name of the output is needed as well as the port(s) associated with it. The syntax used to represent outputs in AGILE is: an *output-name*, followed by a node, and optionally (usually) followed by a second node, all appearing together with no intervening spaces. Since `S` is an AGILE output name (which represents the network S-parameters), the NDL variable `S11` represents the output S11. All of the network outputs have a syntax as:

```
S11
Y$out$in
Z23
```

which represent the appropriate S-, Y- or Z-parameter, respectively. All of the network outputs have two nodes associated with them. Before moving on to discuss the port termination-dependent parameters, there is one syntactical point to clear first. Since AGILE can represent ports greater than 10, how would the S-parameter between ports 3 and 21 be represented, *S321?* This presents an ambiguity in that `S321` could represent the S-parameter between ports 3 and 21 or between ports 32 and 1. This is resolved with an extension of syntax allowing a dot (.) to be placed within the numeric part of an output variable name for the purpose of disambiguation. Examples demonstrate the syntax:

`S3.21`	***S-parameter between ports 3 and 21***
`S32.1`	***S-parameter between ports 32 and 1***
`S321`	***illegal***
`Z1.1`	***Z-parameter at port 1***
`Z11`	***Z-parameter at port 1 -- same as above***

[4] the system impedance is normally set to 50 ohms, however you can SET ZSYS to any desired value on the settings page (or in the config file).

Soutin ***S-parameter between ports in and out***
S$out.$in ***same as above***

This generalized syntax is also used for the port termination-dependent outputs.

In the port termination-dependent class, outputs may be either *reflective* or *transmissive*. Reflective outputs (VSWR is an example) require only a single port for their definition, while the transmissive outputs require two. AGILE defines reflective-type outputs based upon a single port which requires only a single port name for its definition. Two port names are needed to define a transmissive output. Examples are:

VSWR1 *reflective*
PGAIN21 *transmissive*
PGAIN$out.1 *transmissive*

which represent Voltage Standing Wave Ratio at port 1 and Power Gain between ports 2 and 1, and Power Gain between ports out and 1, respectively. The next table summarizes some of the outputs available (for a complete list see the appendix):

OUTPUT VARIABLE DETAILS
(partial list, see appendix)
NETWORK OUTPUTS

Name	Type	Represents	Format	Examples
S		S-parameters	(real, imag)	S11 S21 S44
Y		Y-parameters	(real, imag)	Y11 Y16.2 Y13
Z		Z-parameters	(real, imag)	Z11 Z22 Z14
•••		•••	•••	•••
PORT TERMINATION-DEPENDENT OUTPUTS **(partial list, see appendix)**				
VGAIN	T	Voltage Gain	(real, imag)	Vgain21
IGAIN	T	Current Gain	(real, imag)	Igain23 IGAIN34.67
PGAIN	T	Power Gain	(dB, Angle)	Pgain12 PGAIN12.3
ZIN	R	Input Impedance	(real, imag)	Zin1 Zin14.
RC	R	Reflection-Coefficient	(real, imag)	RC1 rc3
VSWR	R	Voltage Standing Wave Ratio	(real)	VSWR1 VSWR3 vswr4
STAB	T	Rollet's Stability Factor	(real)	STAB12 Stab34
GMAX	T	Maximum Available Gain	(dB)	Gmax12
MATCH	T	Conjugate network match between ports	(real, imag)	Match12 MATCH10.2
•••	•••	•••	•••	•••

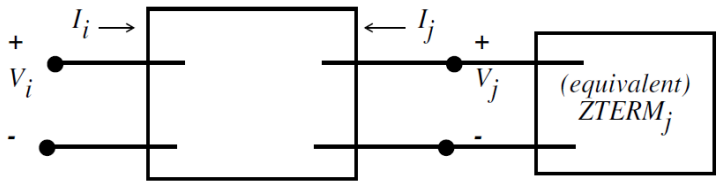
As implied by example in the table above, output variable names may be in upper- or lower-case. The *type* specification in the table specifies for the port termination dependent outputs either *T* or *R* standing for *T*ransmissive or *R*eflective where applicable. All of the reflective outputs are followed by a single port it represents. The transmissive outputs are followed by two port names, using the *(.)* notation described above (as *PGAIN12.3*) if

necessary. Type specification does not apply to the network outputs; they are always defined with two ports. Potentially *S1* could stand for S11, since it is a reflective-type output. We do not allow it, preferring to follow more conventional notation.

Output variables representing the S-, Y- and Z-parameters are always in rectangular-form. This is indicated in the fourth column of the table under *Format.* The means that the output variable *S11* represents the network analysis result of the S11 parameter in rectangular form. All outputs are presented in rectangular form, with the exceptions of *pgain* and *vswr*. `Pgain` is presented as a complex number with its real part the magnitude in decibels and its complex part in degrees. This was done for the convenience of the user only, in anticipation of the desired format for this value. The output variable, `vswr`, is real-valued only as indicated in the table above. In the following section we will define how any output variable might be manipulated into other formats using *output definition statements*.

It is also essential to note -- ***all outputs (except V/ISPECTRUM and V/ISIGNAL) are defined based upon ports not nodes***. Although this fact has been previously stated, it must be emphasized.

Additional detail must be provided in defining exactly what output variables represent. The Z-parameters are presented in ohms and Y-parameters are presented in mhos; it is felt that these are "self-defining". The unitless S-parameters are also self-defining with an important point to note: in the common usage of S-parameters, `S21` represents the wave gain from port 1 to port 2.[5] However, the port termination dependent outputs are not used in this sense, that is `pgain21` represents the "power gained" from port 2 to port 1. In a loose sense, `pgain12` "corresponds" with `S21`. The outputs, *ZIN*, *YIN*, *RC* and *VSWR* are also self-explanatory; they are in units of ohms, mhos, no units and no units, respectively. Please refer to the appendix. In Figure 2.10, voltage and current definitions are shown.



$$Vgain_{ij} = \frac{V_j}{V_i} \Big\} \; all\ ports\ terminated, independent\ of\ port_i\ termination$$

$$Igain_{ij} = \frac{I_j}{I_i} \Big\} \; all\ ports\ terminated, independent\ of\ port_i\ termination$$

$$Pgain_{ij} = \frac{Power\ into\ real\ part\ of\ ZTERM_j}{Available\ power\ at\ Port_i} \Big\} \; all\ ports\ terminated$$

Figure 2.11 Defining the voltage and current directions used in some port termination-dependent output calculations.

[5] this "ordering" also arises from the matrix formulation of S-parameters.

The figure illustrates a two-port network, but **all network output calculations may be performed on a network with any number of ports**. In cases such as this, all ports (other than the "i,j" ports) are terminated, thus computing the correct "terminated" value(s).

We have fully described the output variables without stating how they are used. The next section will define the use of output variables in output definition statements.

2.14 OUTPUT STATEMENTS

An *output statement* is any assignment statement in AGILE NDL which involves an output variable. A key word in this definition is that of *involves*, this means either directly or indirectly. Thus, any assignment statement which depends upon the results of network analysis is an output statement. Output statements follow the same rules as regular assignment statements, however there are some extensions. Depending upon the output variable, it may be complex-valued. Complex operations include all of the binary operations previously discussed, i.e., `MIN`, `MAX`, `+`, `-`, `*`, `/`, and `**`. AGILE correctly performs *mixed-mode* operations, involving both real and complex values, on output assignment statements (and any other statements). A partial table of complex operators is:

Some of the COMPLEX FUNCTIONS (partial list, see appendix)			
Name	Argument	Result	Comments
RTOP	complex	complex	Rectangular to Polar Conversion
PTOR	complex	complex	Polar to Rectangular Conversion
MAG	complex	real	Magnitude
PHASE	complex	real	Phase in degrees
DB	complex	real	magnitude in decibels
REAL	complex	real	the real part
IMAG	complex	real	the imaginary part
•••	**•••**	**•••**	**•••**

The syntax used is identical with that of the real-type functions. Specifically, a *user-named output*, followed by an equal sign, followed by an output expression. Two example output definition statements are:

```
out1 = s11
out2 = phase(s21)
```

In this case, the variables `out1` and `out2` are user-named outputs. Remember from the previous section that `S11` represents the rectangular form of the S-parameter. Therefore, the user-named output `out1` would represent the rectangular form of S11. The user-named output *out2* would represent the angle in degrees of the `S21` output variable. All user-named variables are computed and listed in their own form after network analysis; however, we are not describing how to ask AGILE to enact this until chapter 3. More examples of output definition statements are:

```
angleresult = PHASE(s11)/57.29578
alsoanoutput = -angleresult
phasediffer = ABS( angleresult - PHASE(s22)/57.29578 )
```

A complete description of these examples will elucidate output definition statements. The variable `angleresult` is an user-named output because it depends upon the output variable `s11`. Since `alsoanoutput` depends upon `angleresult` it too is an output, even though it does not explicitly depend upon an output variable. `Phasediffer` is an usernamed output for two reasons, it both depends upon an output explicitly (`s22`) and since it depends upon `angleresult`. The evaluation of `angleresult` can be thought of as: the output variable `s11` is returned as a complex value in rectangular form, the complex function PHASE computes the angle of this in degrees. This is now a *real value* (see the `PHASE` function in chart) which is divided by a real constant (*57.29578*) thus maintaining a real value. Thus `angleresult` is a real-value, which happens to be the phase of S11 in radians. The computation of `alsoanoutput` results in a real value because it is simply the negative of `angleresult`. The value of `phasediffer` is computed as: the value of `PHASE(s22)/57.29578` (real) is subtracted from `angleresult` and then the absolute value is taken of this. Notice the use of a real-function here, namely the absolute value function `ABS`. Since the argument of `ABS` is real, namely, `angleresult – PHASE(s22)/ 57.29578`, a real function may be applied to it, even though complex values where involved in the computation.

Some functions in AGILE are *generic*, meaning that they wil accept either real or complex arguments. For example, the "SQRT" function will take a real-valued argument and produce the real square-root, or a complex argument and produce a complex-valued result. A complete list of the functions available is in the appendix.

We have already seen real constants (like `3.14159, 2.0e5`, etc.) but *complex constants* may also be defined. Complex constants follow rules along the lines of FORTRAN; examples appear as:

```
(0.0,1.0)
(3.4e7,-7.8)
```

Integer, real, or exponential forms of numbers may be used. Complex constants may be used as in:

```
O45 = dB(s11) + (0,0.0174533)*PHASE(s11) o46 = s21*(0,1)
```

The user-named output `O45` will be complex with the real-part the magnitude of S11 in db and the imaginary part representing the angle of `s11` in radians. The output `o46` will represent the complex form of S21 rotated 90°. The following is provided without comment as examples of possible output expressions:

```
phasedif = PHASE(s31) - PHASE(s41)
Bphasedif = APHASE( phasedif )
Magdif = ABS( dB(S21) - dB(S41) )
taper = SQRT( Mag(s21)/(1+dist**2) + Mag(s31)/(1+dist**2) )
ComplexOut = s11/s22 - s21/s12
Ratio = MAG(ComplexOut/taper)
```

2.15 CONTINUATION LINES

AGILE is a line oriented format, meaning that each statement appears on a line (rather than have a custom statement separator as in C). But AGILE does allow longer lines to be specified by continuing text onto the next line. The continuation character, ampersand (`&`), may be placed at the end of a line indicating that a continuation of the current line follows. As many continuation lines as needed can be specified, keeping within the total. An example is:

```
TwoLineDef = 1 + 2 + 3 + 4 + 5   &
```

```
                + 6 + 7 + 8 + 10
```

The ampersand must be the last character. Its effect is similar to that of a space character. For instance, it may not be placed in the middle of a variable name:

```
IwantThis = Operand1 + Operand2
WillNotWork = Operand1 + Oper   &
                           and2
```

It may be used for any type of statement, for instance:

```
T1 = MSTRL(  HT = 300+SQRT(freq/1.0e9) mils,   &
             ER = 2.8 - LOG10( freq/1e9+1.2 ), &
             PL = 10 cm,                       &
             WT = 30 + 3.4*(freq/1e9)**3 mils )
```

demonstrates four lines of continuation.

2.16 COMMENT LINES

Sometimes we may want to place text in the network description for the sole purpose of documentation. AGILE allows comment lines to appear in the network description which are completely ignored. *Comment lines* begin with either an asterisk character, `*`, or the exclamation character, `!`. The asterisk may be placed in any column but it must be the first non-blank character. Some correct and incorrect examples are:

```
* This is a legitimate comment
      * so is this
but * this is not -- it does not begin with an *.
```

Comments may be placed on any line within the file with no effect. One caveat is that comment lines may not be continued. That is:

```
* This is a comment line, lets try to continue it -- &
but this is illegal because comment lines do not continue
! This line and the next are OK, but the &
* at the end of the line above does nothing.
```

An entirely blank line is also considered to be a comment line. The comment line should not contain any semi-colons as these are NDL statement separators (discussed in section 2.18).

2.17 KEYWORD STATEMENTS AND NETWORK NAMES

Keyword statements are lines in the input which may be used to describe the input. The form of a keyword statement is any keyword followed by a colon character (`:`), with the exception of the `END` keyword which appears alone without the colon. Current keywords are:

```
NETWORK          FREQUENCIES   PORTS          ENVIRONMENT  COMPONENTS
CONNECTIONS      TABULATED     DATA           OUTPUTS
GOALS            END           SIGNAL
PASS             GLOBAL        TEMPERATURE    HARMONICS
```

Recognition of these keywords is upper- and lower-case independent. However, they must appear in singular or plural form as shown. Some of the keywords above have not yet been described, those that haven't been described will be in the appropriate chapter.

The keyword `NETWORK` is used to give a *name* to the network. The syntax above is extended by allowing the form:

```
NETWORK : network-name
```

where *network-name* is any character string that conforms to NDL variable naming.

> Tip: It is very important that the network name be the same as the file name in order for Agile to find networks as they may be referenced as subnets. Although network names are case-insensitive, the **file name** for either Windows or Linux should be **in all lower case**.

Many of the keywords above don't actually do anything, they are used only in a role similar to that of comment lines, that is to "document" and "organize" the network description. We have described several types of statements (i.e., frequency-definition, port-definition, and assignment statements) and we may use these keyword statements to "group" them. An example is:

```
FREQUENCIES:
  10 20 2 GHz
  15 GHz
COMPONENTS:
  C1 = CAP(c=67 pF)
  R1 = RES(r=454 ohms)
CONNECTIONS:
  C1 1 2
  C1 2 3
  R1 2 0
OUTPUTS:
  o11 = rtop(s11)
  o21 = dB(S21) + (0,1)*PHASE(s21) END
```

but remember, those keyword phrases do absolutely nothing, they are used above for visual grouping purposes only. AGILE recognizes each statement on its own basis, not on how or where it appears in the input; therefore, the following means the same thing to AGILE:

```
10 20 2 GHz
15 GHz
C1 = CAP(c=67 pF)
R1 = RES(r=454 ohms)
C1 1 2
C1 2 3 R1 2 0
o11 = rtop(s11)
o21 = dB(S21) + (0,1)*PHASE(s21)
```

but do notice that the first fragment is "more readable" due to the keyword statements. Also remember that NDL is independent of order, so the description above could have been "shuffled" into any order and meant the same thing. As with all special character sequences used in AGILE NDL (units, output variables, functions, etc.), the spelling and plurality of the keywords is critical for proper recognition. This means that statements such as:

```
Frequency:
PORT:                    are all illegal
ENVIRONMENTS:
```

will not be recognized and will cause an error to occur.

As previously mentioned, the only keyword statement described thus far with meaning to AGILE is the `NETWORK` keyword statement (and the HARMONIC previously described). The network keyword statement must be used somewhere in the input. AGILE actually uses the network keyword statement for a purpose other than name establishment. The *network keyword statement* defines a "network-type", informing AGILE of what "kind" of network is being defined. AGILE also analyzes "tabulated" multi-port networks which are not defined by components and connection, but rather as a table of numbers which directly represent network parameters (the S-parameters are used for this purpose). This is described in section 3.3, *AGILE TABULATED NETWORKS*. The `GOALS` keyword statement is used in relation to goal definition statements described in chapter 5.

2.18 MULTIPLE NDL LINES

Even though we stated that Agile is record/line oriented, you can place more than one NDL line in each "record" by separating it with a semi-colon (;). The record:

```
#1 1 0 ; #2 2 0
```

for example; it contains two port definition statements. Continuation lines can be used in combination with the multiple line feature. Some care is needed when multiple statements are entered as the semi-colon is also used for AGILE command separation, however, this is discussed in the next chapter.

This feature may be used for more than two NDL statements, and can also be used to contain "end-of-line" comments:

```
R1 = RES(r=45 ohms) ; R1 2 45 ; R1 9 15 ;* R1 is a balance resistor
```

As many NDL statements as desired may be placed on a single record. It is recommended, however, that range variables (section 2.8) be placed on a single line. This will be discussed further in chapter 3.

2.19 PUTTING IT ALL TOGETHER

This section presents and describes a complete NDL example. A minimal complete description of a network would typically contain one or more of: *frequency-definition* statements, *port-definition* statements, *circuit-component* statements, *connection* statements, *output definition* statements, and possibly *environment* statements. A complete example is shown in Figure 2.12. The first line establishes a name for the network. We are using keyword statements for clarity and as good practice. In this example, we are analyzing a two-port network with ports numbered `1` and `2`. We establish an environment called `Ev1` which we use in defining the only component `T1`. The component `T1` is a microstrip transmission line with a 20 mil substrate height, dielectric of 10, physical length of 100 mils, and a width of 12.6 mils. We chose to use a variable, `halfwidth`, in the definition of the transmission line. For outputs, we are assigning the complex-valued variable `o11` to the polar form of S11, the real-valued variable `trans` to the decibel form S21, and the real-valued variable `Tangle` to the phase angle in degrees of S21.

```
Network : Test
 FREQUENCIES:
   10 20 1 GHz
 PORTS:
   #1 11 0
   #2 12 0
 ENVIRONMENT:
   Ev1 : ht = 20 mils, er = 10
 COMPONENTS:
   halfwidth = 6.3
   T1 = MSTRL(ENV=Ev1, PL=100 mils, W=2*HalfWidth mils)
 CONNECTIONS:
   T1 11 12 0
 OUTPUTS:
   o11 = rtop(s11)
   trans = dB(s21)
   Tangle = PHASE(s21)
END
```

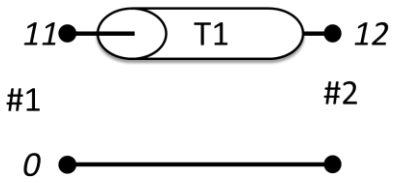


Figure 2.12 A complete NDL example, so far

3 ADVANCED NDL

This chapter describes features and capabilities of AGILE NDL which were not previously discussed in chapter 2. One significant feature of AGILE that has not been described is that of *sub-modeling*, the ability to specify a network and include it in another network. This includes the ability to specify a network as a function of *parameters* and *invoke* it from another network with a specified set of parameters. These topics will be fully described in section 3.1, *SUB-MODELS / NETWORK HIERARCHY*. AGILE also has its own *tabulated network* format as is discussed in section 3.2, *TABULATED NETWORKS*, allowing multiport networks to be represented in S-parameter form.

Another capability of AGILE not discussed in chapter 2, is the ability to analyze networks with *non-default port terminations*. If not otherwise specified, the *system impedance* is assumed as the "termination" at each port, however AGILE allows any network to be used as a port termination, as is discussed in section 3.3, *NON-DEFAULT PORT TERMINATIONS*. One of the new features of AGILE is the ability to have *global* data, either for variables or environments. This allows, for example, that temperature be declared and available in all resident NDL descriptions. This is described in section 3.4, *GLOBAL NETWORKS AND VARIABLES*.

3.1 SUB-MODELS / NETWORK HIERARCHY

In many cases, it is desirable to build large network topologies from several smaller networks. AGILE facilitates this type of inter-network construction through *sub-modeling*. A sub-model may be "fixed" in the sense that values used for component definition are entirely defined within the network itself. However, a sub-model may also "receive" values from the "invoking" network and uses them to define its own components. This is a *parametric* form. Perhaps the best way to introduce these topics is by illustration, using a network such as:

```
Network : RLC
FREQUENCIES:
 1 12 1 GHz ; 0.8 3 .1 Ghz
PORTS:
 #1 10 0 ; #2 20 0
COMPONENTS:
  R1 = RES(R=10 OHMS)
  L1 = IND(L=1 nH)
  C1 = CAP(C=10 pF)
CONNECTIONS:
  R1 10 2
  L1 2 3
  C1 3 4
  R1 4 5
  L1 5 6
  C1 6 20
  R1 4 7
  L1 7 8
  C1 8 0
OUTPUTS:
  o11 = s11
  O2 = dB(S21)
END
```

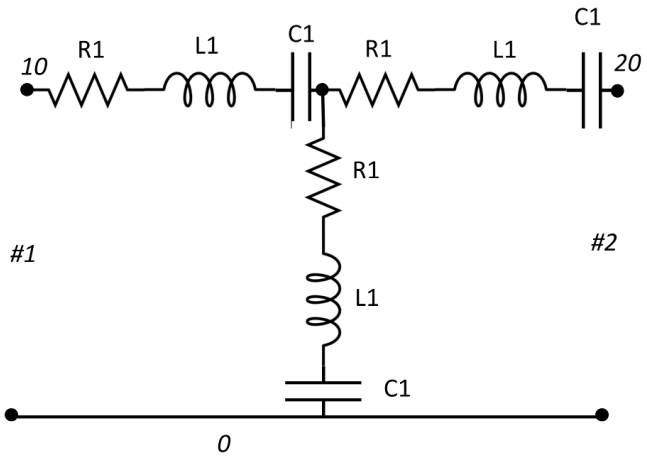


Figure 3.1 An example network.

As is obvious from the schematic, a series resistor, inductor, and capacitor are "repeated" three times. This type of redundancy is well suited for a sub-model implementation. Let's create a network containing only these components. This will be called, `RLCseries`, as shown:

```
Network : RLCseries
COMPONENTS:
  R1 = RES(r=10 ohms)
  L1 = IND(l=1  nH)
  C1 = CAP(c=10 pF)
CONNECTIONS:
  R1 1 2
  L1 2 3
  C1 3 4
END
```

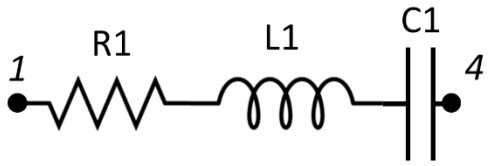


Figure 3.2 The RLCseries network.

Notice that this network has no frequencies, ports, outputs, etc. that other networks used as examples previously had. If analysis is attempted, an error message would be generated:

```
Error - Network Analysis with No Frequencies
```

which happens to be the first thing AGILE detects as being deficient in order to do network analysis. Even if frequencies were added to this network, then the next complaint AGILE would have is: `Error - Network Analysis with No Ports`. However, we are getting off the track here, the original definition of the network `RLCseries` is suitable, even without frequencies, ports, or outputs, for **sub-model connection**. Before defining the generalized syntax for sub-model invocation, an example of usage is presented. The network, `RLCmain`, will be used to "invoke" the sub-model, `RLCseries`, as:

```
Network : RLCmain
FREQUENCIES:
  1 12 1 GHz
  .8 3 .1 GHz
PORTS:
  #1 10 0
  #2 20 0
COMPONENTS:
  RLCsub = :RLCseries() { 1 4 }
CONNECTIONS:
  RLCsub 10 4
  RLCsub 4 20
  RLCsub 4 0
OUTPUTS:
  o11 = s11
  o2  = dB( S21 )
END
```

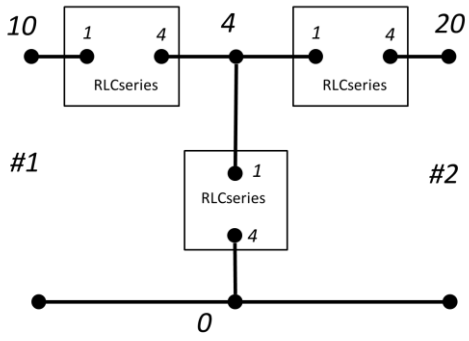


Figure 3.3 The RLCmain network, using RLCseries as a sub-model.

Some terminology is in order; in review of the NDL line:

```
RLCsub = :RLCseries () { 1 4 }
```

the "local" user-named component[6], RLCsub, is assigned to the sub-model, RLCseries. Whereas a built-in component (like TRLs) have "pre-defined" number and order of nodes associated with them, a sub-model invocation must specify which nodes of the sub-model are to be used. The form of a sub-model specification is a colon (:), followed by the name of the sub-model, followed by a "passed parameter" list (empty in the case above), followed by a left curly bracket ({), followed by a node list, and finished with the right curly bracket (}).

In this case, the name of the sub-model is RLCseries, and the node list is 1 and 4. We might call nodes 1 and 4 the *requested node list*, since from the network RLCmain we really do not "know" which nodes even exist inside of RLCseries. Of course, since we created RLCseries, we know what nodes are defined, but if the line:

```
RLCsub = :RLCseries() { 1 5 }      is wrong since node 5 does not exist in RLCseries
```

where used, an error would be generated stating that node 5 does not exist in RLCseries. Let's concentrate on proper sub-model usage before delving to deeply into erroneous situations.

> Tip: All (hopefully) of the networks used in the manual are installed in the Agile Data/Net directory. If you want to try them take a peek at Chapter 9 for a quick start.

If these two networks, RLCmain and RLCseries, (they are in the install) were already stored as files, then: using the 'Open' item in the menu and selecting rlcmain.net will cause both of these to be read in, that is it will "recursively" read in NDL descriptions that are needed as sub nets or port termination networks.

Now suppose the RLCseries network did have frequencies, ports, and the like? Here is a modification of the RLCseries network, called RLCseries1:

```
Network : RLCseries1
FREQUENCIES:
  0.5 6 0.5 GHz
PORTS:
  #1 1 4
CONNECTIONS:
  RES(R=10 ohms)  1 2
  IND(L=1 nH)     2 3
  CAP(C=10 pF)    3 4
OUTPUTS:
  o1 = dB(s11)
END
```

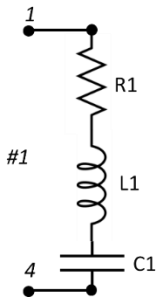


Figure 3.4 The RLCseries1 network.

This network has frequencies, a port definition and output statements, and can be analyzed alone. Besides these additional NDL lines, we also changed the components to "immediate" types. This was arbitrarily done to reduce the total number of lines necessary, but otherwise the networks (RLCseries1 and RLCseries) are the same from a component standpoint (of course, RLCseries1 has a port definition statement which RLCseries does not). Here is a modified version of the network RLCmain, called RLCmain1:

[6] Remember the terminology of Chapter 2.

```
Network : RLCmain1
FREQUENCIES:
  1 12 1 GHz
  0.8 3 .1 GHz
PORTS:
  #1 10 0
  #2 20 0
COMPONENTS:
  RLCsub = :RLCseries1() { 1 4 }
CONNECTIONS:
  RLCsub 10 4
  RLCsub 4 20
  RLCsub 4 0
OUTPUTS:
  o11 = s11
  o2  = dB( S21 )
END
```

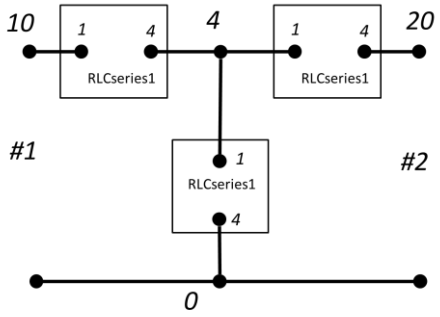


Figure 3.5 The RLCmain1 network, using RLCseries1 as a sub-model.

This network invokes the `RLCseries1` network. The definitions of `RLCmain` and `RLCseries` are identical as for `RLCmain1` and `RLCseries1`, with the exception that `RLCseries1` has a "complete" network definition. This situation is perfectly legal in AGILE. Any network may be used for sub-model connection. However, some questions are pointed up by this example, namely what about the frequencies, port definition and outputs of `RLCseries1`? Notice that the frequencies specified in `RLCseries1` are different than those specified in `RLCmain1`. The rule here is that frequencies are always defined by the "main" network. Any frequency (or harmonics) specifications in submodels are ignored. Of course, in the case of analyzing the network `RLCseries1`, it is then the "main" network so its frequency statements would be used.

Port specifications and output definition statements are also ignored in sub-models, but are allowed as we can see by the `RLCmain1/RLCseries1` example. By allowing these to be specified, one can analyze the network in its own right, but also use it as a sub-model (without changing its definition). An important point that should be re-emphasized here, **sub-models are connected by nodes -- not ports**. Notice that we are requesting particular nodes of the sub-model with the sub-model invocation statement. Ports may be arbitrarily specified in the sub-model but it does not affect the ability of any invoking network to request nodes. In the network `RLCseries1`, the nodes used by `RLCmain1` are coincidently the same pair used for `RLCseries1`'s ports definition.

We have shown examples of a single sub-model inside of a main network. However, the sub-modeling mechanism in AGILE is actually more generalized then the previous examples illustrate. In AGILE, sub-modeling is "hierarchical" meaning that sub-models may contain other sub-models, etc.

We can further improve this example by *passing parameters* to the sub-model. Notice that in the cases of either `RLCseries` or `RLCseries1`, the components are "fixed." Using submodeling, values are "passed into" the sub-model from the invoking network; these values may then be used to define the sub-model's component(s). As an initial example, we might want to generalize the `RLCseries` network by allowing the resistance, inductance, and capacitance values to be parameters. To illustrate this, here is a modification of the `RLCseries` network, called `RLCseries2`:

```
Network : RLCseries2( R1,L1=9,C1 )
CONNECTIONS:
  RES 1 2 (r=R1 ohms)
  IND 2 3 (L=L1 nH)
  CAP 3 4 (C=C1 pF)
END
```

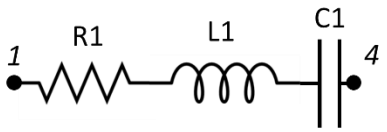


Figure 3.6 The RLCseries2 sub-model network. Same as RLCseries but with component values determined as parameters.

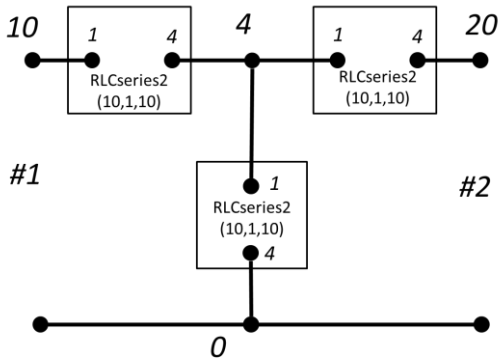
Additional terminology must be introduced, looking at the "Network" keyword NDL specification, a more generalized form than that presented in chapter 2 has been introduced:

```
Network : RLCseries2( R1,L1=9,C1 )
```

We call `R1, L1` and `C1` formal parameters. What is being done is that we are defining the network `RLCseries2` as a function of its formal parameters. We then used these values in immediate connection statements, which defines the component values. Although, in this example, we have used the formal parameters directly, we may actually use them arbitrarily in any expression. Before demonstrating this aspect, we must discuss the "invocation" of the sub-model. The values of the formal parameters will be completely determined by the invoking network using the sub-model invocation statement. As an example, here is a modification of the `RLCmain` network, called `RLCmain2`, we will also re-list the `RLCseries2` network:

```
Network : RLCmain2
FREQUENCIES:
  1 12 1 GHz
  .8 3 .1 GHz
PORTS:
  #1 10 0
  #2 20 0
COMPONENTS:
  RLCsub = :RLCseries2 (r1=10,l1=1,c1=10) { 1 4 }
CONNECTIONS:
  RLCsub 10 4
  RLCsub 4 20
  RLCsub 4 0
OUTPUTS:
  o11 = s11
  o2  = dB( S21 )
END

Network : RLCseries2( R1,L1=9,C1 )
CONNECTIONS:
  RES 1 2 (r=R1 ohms)
  IND 2 3 (L=L1 nH)
  CAP 3 4 (C=C1 pF)
END
```



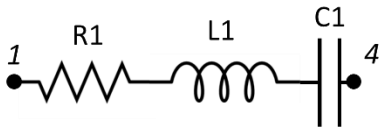


Figure 3.7 The RLCmain2 network, using RLCseries2 as a sub-model.

The sub-model invocating NDL line:

```
RLCsub = :RLCseries2 (r1=10,l1=1,c1=10) { 1 4 }
```

is a generalization of the syntax of sub-model invocation. This statement includes a set of parameters in addition to the sub-model name, described previously as sub-model invocation. The values being passed, in this case the numbers (`10,1,10`), are called the *actual parameters*. These three values are associated, by name, with the three formal parameters of `RLCseries2`: `R1, L1,` and `C1`. Aside: the "`show networks`" command after analysis of `RLCmain2` will show the following (*after* a valid analysis):

```
********************** # 1 **********************
network-name = RLCMAIN2
file-name = C:/Users/…/AppData/Local/Agile Data/Nets/rlcmain2.net
changed vs. file? = no
changed vs. last anal? = no
selected = yes
network-type = Network
Formal Parms : ()

********************** # 2 **********************
network-name = RLCSERIES2
file-name = C:\Users\...\AppData\Local/Agile Data/Nets/rlcseries2.net
changed vs. file? = no
changed vs. last anal? = no
selected = no
network-type = Network
Formal Parms : ((R1, is_traced=0, DEF=none, VAL=10),(L1, is_traced=0, DEF=9, VAL=1),(C1,
is_traced=0, DEF=none, VAL=10))
```

This `RLCseries2` network is now a significant improvement over the fixed `RLCseries` network, it is a sub-model for a generalized series RLC network. It can be used from any invoking network at any values of resistance, inductance, or capacitance. Again, looking at the network line

```
Network : RLCseries2( R1,L1=9,C1 )
```

Note that the parameter `L1` has the `=9` phrase. This phrase defines a *default* value for the parameter `L1` if it is not specified in the invocation. Any or all parameters may have a default value. Note in the case shown Figure 3.7Figure 3.7 The RLCmain2 network, using RLCseries2 as a sub-model., the `L1` default value is not used anyway since it is assigned a value of 1 in the invocation.

It is appropriate at this time to discuss some of the possible error conditions applicable to sub-modeling. If a sub-model includes frequency, port(s), and output statements, it may be analyzed as a "main" network. In this case, default values for all parameters must be specified in order to analyze a parameterized network as a main circuit. For instance, if ports, frequencies, etc. were added to `RLCseries2` and a list outputs command were used, the following error message would result:

```
Error - Main network (RLCSERIES2) Must Have Default Formal Parameter Values
```

Another possible error in the context of sub-modeling is that of "parameter mismatches". This occurs whenever the names of actual parameters are not identical to the names of formal parameters. Suppose we had used the NDL line:

```
RLCsub = :RLCseries2 (R2=10,C1=1) { 1 4 }
```

in the `RLCmain2` network? Here the `R2` actual parameter is illegal. If analysis of this were attempted, this error message would be issued:

```
Error - SubNet Argument (R1) Must Have Value, Does Not Have Default
```

This message tells us that an invocation of `RLCseries2` was attempted from `RLCmain2` where the name of formal parameter specified in `RLCseries2` is not matched by an appropriate actual parameter. Although the actual parameters in the invocation of `RLCseries2` specified in `RLCmain2` are constants, this is not necessary, actual parameters may be any arbitrary, real-valued expression. As an example, here is a modified version of `RLCmain2`, called `RLCmain3` (we will not repeat the definition of `RLCseries2`):

```
Network : RLCmain3
FREQUENCIES:
  1 12 1 GHz
  .8 3 .1 GHz
PORTS:
  #1 10 0
  #2 20 0
COMPONENTS:
  RLCsub = :RLCseries2 (R1=10,L1=lrange,C1=10) { 1 4 }
  lrange = 1 [ 1:5:1 ]
CONNECTIONS:
  RLCsub 10 4
  RLCsub 4 20
  RLCsub 4 0
OUTPUTS:
  o11 = s11
  o2  = dB( S21 )
END
```

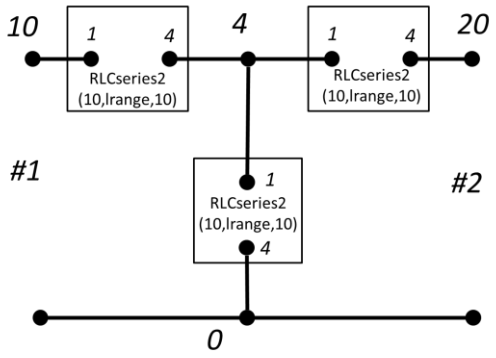
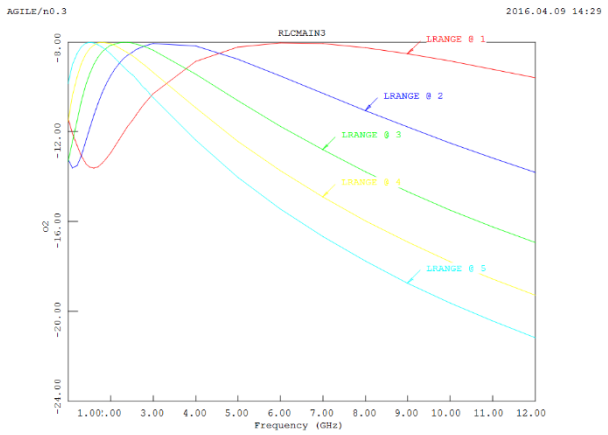


Figure 3.8 An actual parameter is a variable.

We have now specified the second actual parameter being passed to `RLCseries2` as variable, `lrange`, rather than a constant. This evaluation of this variable gets passed to the formal variable `L1` in `RLCseries2`. The variable `lrange` happens to be a range with-step type. Doing a ‘plot vs’ of o2 and lrange will look like this (after some fiddling with the labels and scaling):

Figure 3.9 RLCmain3 plot of o2 vs Lrange



The actual parameter list may actually be any expression which will evaluate as a real value. As an example, this might be an invocation of `RLCseries2`, note that we are using the default value for `L1` in this case:

```
RLCsub = :RLCseries2 (R1=100-SQRT(lrange),C1=4.78*lrange) { 1 4 }
```

As stated, the actual parameters must be real-valued, but you could use two real-valued arguments to represent a complex value if need be. If the definition of `RLCsub` were changed to:

```
RLCsub = :RLCseries2 ( R1=(7,8), L1=3, C1=3 ) { 1 4 }
```

is illegal, will result in the following error message:

```
Error - Argument (R1) has Complex Value
```

Another restriction on actual parameters is that they must not be network dependent. Referring to the above definition of `RLCmain3`, the variable o2 is network dependent, so that if the definition of `RLCsub` where changed to:

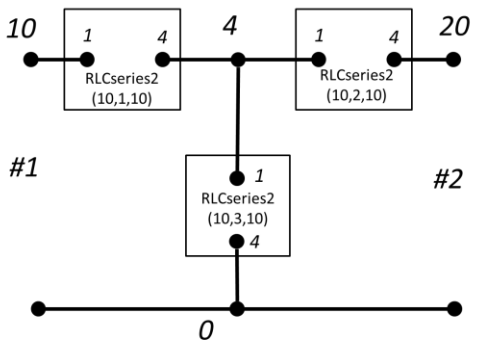
```
RLCsub = :RLCseries2 ( R1=o2, L1=3, C1=3 ){ 1 4 }
```

is illegal, will result in the following error message:

```
Error - Component Argument (R1) is Analysis Dependent
```

Sub-models may be connected using the immediate connection statement, just as with built-in components (remembering section 2.12). If three "instances" of `RLCseries2` where to be connected, it might be convenient to use the immediate connection form. Here is yet another modification to the `RLCmain3` network, called (what else) `RLCmain4`:

```
Network : RLCmain4
FREQUENCIES:
  1 12 1 GHz
  .8 3 .1 GHz
PORTS:
  #1 10 0
  #2 20 0
CONNECTIONS:
  :RLCseries2(R1=10,L1=1,C1=10) { 1 4 } 10 4
```



```
  :RLCseries2(R1=10,L1=2,C1=10) { 1 4 } 4 20
  :RLCseries2(R1=10,L1=3,C1=10) { 1 4 } 4 0
OUTPUTS:
  o11 = s11
  o2  = dB( S21 )
END
```

Figure 3.10 Immediate connection for sub-models.

A key concept here is basically that the sub-model `RLCseries2` has a single definition but it may be used multiple times. The multiple invocations may be at different parameter sets, as is the case in `RLCmain4`, but it is not necessary. For example, refer back to Figure 3.8. Suppose we did not use the `RLCsub` user-named component and instead used the immediate connection form. The NDL fragment for this would look like:

```
:RLCseries2(R1=10,L1=1,C1=10) { 1 4 } 10 4
:RLCseries2(R1=10,L1=1,C1=10) { 1 4 } 4 20
:RLCseries2(R1=10,L1=1,C1=10) { 1 4 } 4 0
```

Notice that all three instances are actually identical. From a sub-model standpoint identical implies the same parameter set and the same nodes. AGILE, however, does not recognize this situation and will analyze them as three "different" instances (correctly, of course).

So far, we have seen examples of a sub-model included inside of another, of a single instance of a sub-model included in a "main" network, and of three instances of a submodel included in a main network. An obvious question is, "Are sub-models allowed to include other sub-models?" The answer is YES. The sub-modeling mechanism in AGILE allows complete freedom in inter-network topology, as long as certain rules are followed.

Certainly, a network may not include itself, or include another which in some form includes itself. A situation of this sort is called "recursive." This is really not a limitation in that this situation cannot physically exist.

An example of "nested" networks is in order. We will use our `RLCseries2` and include it into another, and then include that composite network into a main network. This will effectively illustrate a "two-layer nesting" of networks. First, we would like to create a network which includes one copy of the `RLCseries2` network and a transmission line, we'll call it `RLCandTRL`. When invoking this network, the inductance value will be fixed, but the resistance, capacitance, and the length of a transmission line will be parametric. Here is the definition of this network:

```
Network : RLCandTRL( R1,C1,Tlen )

CONNECTIONS:
    :RLCseries2 2 3 ( R1=R1, L1=1.5, C1=C1 ) { 1 4 }
    TRL 1 2 0 (Z = 45 ohms, Er = 1.3, PL = Tlen mils)
END
```

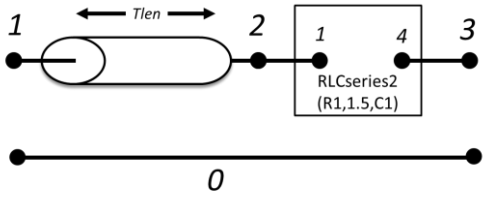


Figure 3.11 The RLCandTRL network, using the RLCseries2 network

Notice that the `R1` and `C1` values are passed into the `RLCandTRL` network and then "passed-thru" into the `RLCseries2` network. Also note the flexibility of placing the nodelist, as with built-in components, the node list may precede the component arguments. We will now define a network which uses two instances of `RLCandTRL`, and call it `RTmain`. However, each instance of `RLCandTRL` will use a different node set and a different parameter set. Here is the definition of `RTmain`:

```
Network : RTmain
FREQUENCIES:
  1  12  0.5  GHz
PORTS:
  #1 1 0
  #2 2 0
CONNECTIONS:
 :RLCandTRL 1 3 0 0 ( R1=10, C1= 10, Tlen=55.5 )  { 1 2 3 0 }
 :RLCandTRL 3 2 0 ( R1= 5, C1=2.5, tlen=37.4 )  { 1 3 0 }
OUTPUTS:
 out1 = s11 out2 = dB( s21 )
 out3 = out2 + (0,1)*PHASE(s21)
END
```

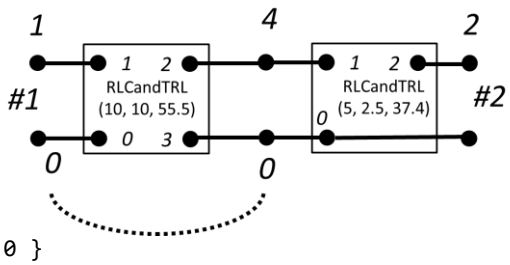


Figure 3.12 The RTmain network, using two RLCandTRL networks.

An important point to note is the connection specification for the first instance of `RLCandTRL`. It specifies that two different nodes of `RLCandTRL` should both be connected to local node `0`. This effectively shorts this component out, the dashed line in the diagram highlights this point. This is perfectly legal however duplicate nodes may not be requested from a sub-model. This error situation will not be demonstrated.

In summary, any hierarchical sub-model or sub-model interconnection is easily and directly represented in AGILE. The sub-modeling capability allows users to "add" models that are not available as built-in components directly in AGILE NDL. Simple examples of RLC networks where used as illustration of this feature, but much more complicated examples, using more extensive algebraic features, are easily achieved. In a sense, the examples shown only "scratch the surface" of what is possible. We demonstrated a "two-layer" deep sub-model interconnection, but AGILE is fully generalized to allow arbitrary nesting of networks. This allows, and even encourages, the design of networks in a functional building block fashion.

3.2 TABULATED NETWORKS

AGILE has a format allowing general multi-port networks to be represented in tabulated S-parameter form. These networks are called (naturally) *tabulated networks*. The format of an AGILE tabulated network is quite simple; the form is: frequency (in MHz) followed by the data. The keyword `TABULATED` must be used in place of the keyword `NETWORK`. Here is an example, using the appropriate keyword phrases:

```
TABULATED : tab2
*
14000.0 .1241  -5.4 .1051   9.2  .1751   9.7  .3167 -15.0
15000.0 .1167 -15.0 .1035   -.8  .1035   -.8  .1085 -24.2
16000.0 .0909 -44.1 .0960 -30.4  .0960 -30.4  .0815 -55.0
*
OUTPUTS:
x11 = rtop(s11)
x21 = rtop(s21)
Ratio2to1 = 10*LOG10(MAG(s21)/MAG(s12))
END
```

Figure 3.13 A two-port Tabulated Network

Notice that the full range of AGILE output computations are available in tabulated networks. `Tab2` is a two-port tabulated network example. Notice that the data is in the order of, S11, S21, S12, S22, this is called **row-major order**. A three-port example is:

```
TABULATED : tab3
*
Freq       S11             S21             S31
           S12             S22             S32
           S13             S23             S33
*
14000.0    .1241  -5.4  .1051  9.2   .1751   9.7  &
           .3167 -15.0  .1035   -.8  .1835   3.8  &
           .1167 -25.0  .4035  -3.8  .7435  15.8
15000.0    .1167 -15.0  .1035   -.8  .1035   -.8  &
           .1085 -24.2  .1014 -10.7  .1014 -10.7  &
           .1000 -34.0  .0985 -20.6  .0985 -20.6
16000.0    .0909 -44.1  .0960 -30.4  .0960 -30.4  &
           .0815 -55.0  .0929 -39.8  .0929 -39.8  &
           .3167 -15.0  .1035   -.8  .1835   3.8
OUTPUTS:
 x11=rtop(s11)
 x21=rtop(s21)
 x31=rtop(s31)
END
```

Figure 3.14 A three-port Tabulated Network

Notice that the data is on one logical NDL line by use of continuation lines. One does not need to use continuation lines, but in the case here it is simpler to visualize. NDL lines containing only data are called data statements. Such data statements should only be used in tabulated networks. AGILE relates the number of ports and the number of real-numbers on each data statement line by use of the expressions:

$$N = \sqrt{\frac{M-1}{2}} \qquad \text{conversely} \qquad M = 1 + 2N^2$$

where M is the number of real-numbers on each line and N is the number of ports being represented. The expressions above, relating the number of ports to the number of real-numbers specified, arise from the fact that there is one frequency and two numbers specified for each combination of ports, a complex number in polar form. Thus a one port network requires three numbers for definition, a two-port requires nine, a three-port needs nineteen, etc.

Although a formal syntax has not been presented for tabulated networks, the following guidelines should generate successful use. The number of real-numbers specified on each NDL line must conform to the equations above, otherwise an error will be generated. Each number must be separated by one or more space characters; remember that frequency is specified first in **mega-hertz (MHz)**. The `TABULATED` keyword must be used in place of the `NETWORK` keyword normally used. Another point previously unmentioned, is that the port numbers are automatically assigned as one through M according to the expressions above. These numbers are used in conjunction with output variables as demonstrated. As for sub-model connection of these networks, node numbers are similarly assigned, as one through M with node 0 (zero) as a reference.

The `save output button in AGILE` will cause AGILE to analyze the current network and place the network analysis results in a file called `name.net` in the above mentioned tabulated form. The network must be linear. There are two useful aspects of such a capability. Firstly, the tabulated output form might be used by external programs for various purposes. Secondly, the tabulated output network might represent the analysis of extremely large fixed networks. In this case, tabulated form can save time by storing the data for such a complex (large number of nodes) network in a file which is then subsequently used as a sub-model in tabulated form.

3.3 NON-DEFAULT PORT TERMINATIONS

All of the examples which have been presented were analyzed using the system impedance as the "port-termination". In this section, we will describe how to specify any sub-model (or sub-model) as a port termination. We will use the `wba` network as part of our examples.

```
Network : wba
FREQUENCIES:
  100 1300 100 MHz
PORTS:
  #1  1  0
  #2  2  0 :dload() { 4 5 }
COMPONENTS:
  Bipolar = :nec() { 1 2 0 }
CONNECTIONS:
  res(r=183 ohms)  1 6
  ind(l=3 nH) 6 3
  Bipolar 1 7 4
  ind(l=1.75 nH)   4 5
  prlc(r=12 c=2.6) 5 0
  ind(l=2.5 nH)    7 3
  cap(c=1.23 pF)   3 0
  ind(l=3.53 nH)   3 2
OUTPUTS:
  NLoutput = Z11
  Loutput1 = Zin1
  Loutput2 = VSWR1
END
```

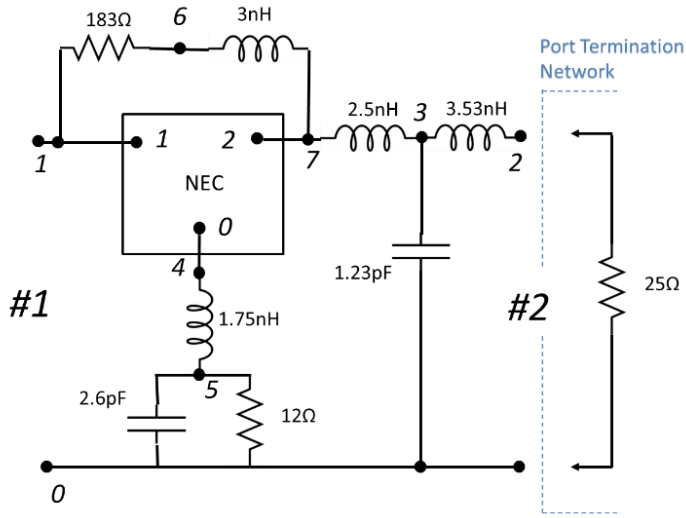
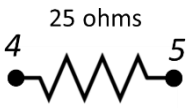


Figure 3.15 The wba network, which uses nec as a sub-model, and terminated with dload.

Suppose we wanted to "load" port 2 of the `wba` network with a 25 Ω resistor not the default system impedance? If both ports of `wba` where to be 25 Ω, we could set the system impendance ont the Set/Show tab to 25, but if only one port is to be changed this doesn't work. So, in the case where the port terminations are not the same, a "port termination" network is required. Here is a simple AGILE network, called `dload`:

```
Network : dload
  RES(r=25) 4 5
```



```
END
```

Figure 3.16 The dload network, just a 25 Ω resistor.

Obviously, this network is just a 25 Ω resistor connected between nodes 4 and 5. As alluded to in chapter 2, there is an extended syntax of the port definition statement. This syntax allows a sub-model to be specified as a network used as the port termination. Since S-parameters are not load dependent[7], the `wba` network uses both `dload` as a port termination network and contains outputs that do depend on port termination.

In examination of the NDL line:

```
#2  2 0 :dload() { 4 5 }
```

The line may be interpreted as: port number 2 of `wba1` is defined as node pair `2` and `0`; the network `dload` is the port termination network, with node `4` of `dload` connected to `wba2` node `2` and node `5` of `dload` connected to `wba` node `0`. Analysis is now done with the `dload` network used as a port termination.

Suppose, we would like to analyze a network using a port termination network which has a transmission line and a resistor? We would like to "control" the length and impedance of the transmission line and the value of the resistance. This will be accomplished using parameter passing. Here is a definition of the network `dload1`, we'll use immediate connection forms to keep the NDL short:

```
Network : dload1 ( Zline, Lline, Rval )
CONNECTIONS:
 TRL(Z=Zline ohms, PL=Lline mils, ER=1 ) 1 2 0
 RES(R=Rval ohms) 2 0
END
```

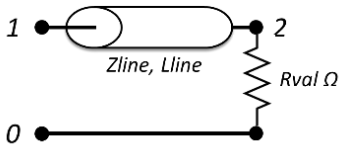
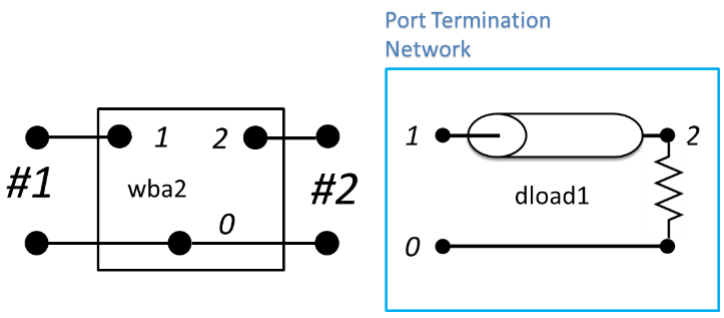


Figure 3.17 The dload1 network.

Nodes `1` and `0` of the `dload` network will be used at the port termination of this next network, called `wba2`:

```
Network : wba2
FREQUENCIES:
  100 1300 100 MHz
PORTS:
  #1 1 0
  #2 2 0 :dload1( zline=Zline, lline=Lline, rval=Rval) { 1 0 }
COMPONENTS:
  Zline = 20 [ 20:50:10 ]
  Lline = 257.0
  Rval = 50
  Bipolar = :nec() { 1 2 0 }
CONNECTIONS:
  res(r=183 ohms) 1 6
```



[7] remember from circuit theory and chapter 2 that S-parameters are NOT DEPENDENT on individual port loads, they will not change regardless of the port terminations, but they DO depend on the system impendence setting (nominally 50 ohms, but settable)

```
  ind(l=3 nH) 6 3
  Bipolar 1 7 4
  ind(l=1.75 nH) 4 5
  prlc(r=12 c=2.6) 5 0
  ind(l=2.5 nH) 7 3
  cap(c=1.23 pF) 3 0
  ind(l=3.53 nH) 3 2
OUTPUTS:
  NLoutput = Z11
  Loutput1 = Zin1
  Loutput2 = VSWR1
END
```

Figure 3.18 The wba2 network, which uses nec as a sub-model, and "terminated" with dload1. Only crucial features are illustrated.

Now, in the case of `wba2`, we have basically re-specified the `wba` network, but have changed the port termination network to `dload1` rather than `dload`. The `dload1` network has three parameters which are passed from `wba2` as shown in the definition in Figure 3.18. The rules for the parameter set of a port termination network are the same as for that of any other sub-model, that is they may be expressions rather than simple variable names. We have now "controlled" values used in the port termination network from the "main" network, we can issue a plot results versus parameters in the load network as plot loutput2 vs zline (*note:* these commands will be fully explained later):

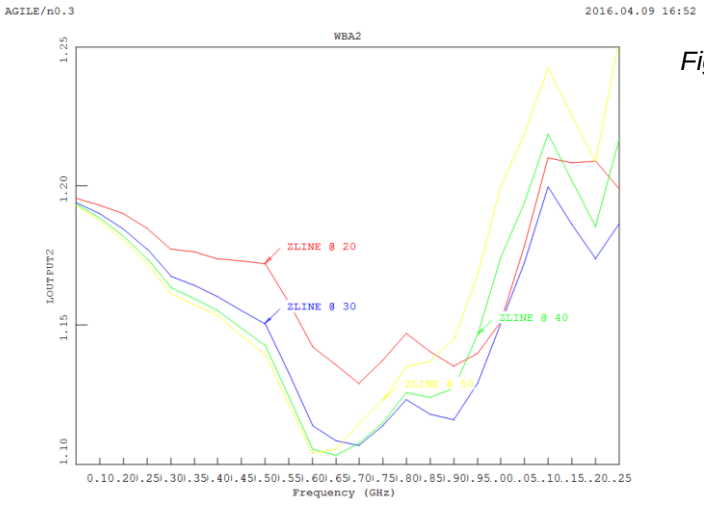


Figure 3.19 Plot of Loutput2 vs Zline in wba2

In summary, the port termination capabilities of AGILE are completely generalized, allowing any two nodes of any network (with or without parameters) to be connected as a port termination. Every port of a main network may be "defaulted" to the system impedance, or any and all of these ports may be terminated with an arbitrarily complex network; even a tabulated network may be used. When using an AGILE network as a port termination, parameter passing sub-modeling style can be used to study the effects of varying the port termination network.

3.4 GLOBAL NETWORKS AND VARIABLES

The networks illustrated thus far have been completely defined with their own *local variables* (and parameters). The variables defined so far may only be used within the context of their own NDL description. AGILE also allows the definition of variables and/ or environments which can be used in any or all networks. All variables defined in a *global network* are available in any other NDL description. An example is:

```
GLOBAL : glob1
pi = 3.14159
j  = (0,1)
ms_env : ht=4 mils, er =12.9, tand = 0.01
END
```

In this case, the global network `glob1` defined a variables `pi, j`, and `ms_env`. Note that the keyword `network` is replaced with `global` when defining a global network. These variables may then be used in other descriptions. For example:

```
Network : w
PORTS:
  #1 1 0;  #2 2 0
FREQUENCIES:
  10 20 1 ghz
COMPONENTS:
  r1 = 47+(1/sqrt(pi-20)) ohms
  c1 = 50 pf
CONNECTIONS:
  MSTRL(env=ms_env,w= 3.3 mils) 1 2 0
  r1 1 2
  c1 2 0
OUTPUTS:
  o11r = s11
  powergain = pgain12
END
```

In this example, the variables `pi` and the environment `ms_env` were not described within the network. In this case AGILE will search resident[8] global networks for this variable. The variables in the global network may be used in as many other networks as is desired. AGILE restricts the types of statements that are useful in global networks, namely: real and complex-valued variables, range variables, variates (described later), and environments. Within any global network, expressions may be used to interrelate variables, but they should not depend upon any other global variable. Here is a summary of rules and restrictions in global variable use:

- Component definitions, frequency statements, port statements, etc. cannot appear in a global network. Only variables (can be frequency dependent but not time dependent), range variables, variates and environments may be used.
- Variables in global networks are evaluated before "regular" (non-global) networks. Any frequency dependency utilizes the frequencies in the main (current) network.

[8] Loading global networks is done with the **Read Single File** menu item.

- As many global networks as needed can be loaded.
- The variables within a global network should only depend upon one another, not variables in other globals. AGILE will not complain about this situation, however, the order in which global networks are evaluated is undefined and miscellaneous errors may result.
- If a local variable has the same name as a global variable, the local one is used. Thus the local name "hides" the global one.
- Within a global network, AGILE will not allow the same variable to have multiple definitions (as it does in any other network). However, if a variable with the same name appears in more than one global, the one that is used is undefined - AGILE will not complain or issue a warning about this either.

3.5 TRACING VALUES

With the ability to write arbitrary expressions and have parametric hierarchies, its useful to be able to "see" the values being used or assigned. The "tracing" feature of AGILE displays the values of assignments, component argument values and actual or formal values. The general method for making a value traceable is by using the question mark instead of the equal sign in its assignment but an example will hopefully clarify this. Consider the following NDL circuits:

```
Network : RLCmain5
FREQUENCIES:
  1 3 1 GHz
PORTS:
  #1 10 0; #2 20 0
COMPONENTS:
  lrange = 1 [ 1:5:1 ]
  Rv ? sqrt(10 + lrange)
  Cv ? freq/1e9
  RLCsub = :RLCseries3( R1?Rv+1,L1=lrange,C1=Cv ) { 1 4 }
CONNECTIONS:
  RLCsub 10 4
  RLCsub 4 20
  RLCsub 4 0
OUTPUTS:
  o11 = s11
  o2  = dB(S21)
END
```

and

```
Network : RLCseries3( R1,L1 ? 9, C1? )
CONNECTIONS:
  RES 1 2 (r?R1 ohms)
  IND 2 3 (L=L1 nH)
  CAP 3 4 (C=Cvv pF)
  Cvv ? C1 + 2
END
```

looking first at the `RLCmain5` network, note that the `Rv` and `Cv` values where assigned using the question mark `?` not the equal sign, =. Thus, both of these become *traced*. The actual argument `R1` in the invocation of sub-network

RLCseries3 is also assigned using the ? instead of an equal sign and thus is also traced. Moving to the network RLCseries3 we see that formal argument L1 uses the ? in assigning a default value and that the argument C1 has a trailing ?, and this makes these traced. The variable Cvv is also traced ala Rv and Cv before. When an analysis is done with the traced checkmark checked, the following output is generated:

```
Traced at Freq=n/a (Symbol -- RV) is 3.31662
Traced at Freq=1 GHz (Symbol -- CV in RLCMAIN5) is 1
Traced at Freq=1 GHz (Component Arg -- R1 of RLCSUB in RLCMAIN5) is 4.31662
Traced at Freq=1 GHz (Formal -- L1 in RLCSERIES3 in RLCMAIN5) is 1
Traced at Freq=1 GHz (Formal -- C1 in RLCSERIES3 in RLCMAIN5) is 1
Traced at Freq=1 GHz (Symbol -- CVV in RLCSERIES3 in RLCMAIN5) is 3
Traced at Freq=1 GHz (Component Arg -- R of line:3 in RLCSERIES3 in RLCMAIN5) is 4.31662
Traced at Freq=1 GHz (Component Arg -- R1 of RLCSUB in RLCMAIN5) is 4.31662
Traced at Freq=1 GHz (Formal -- L1 in RLCSERIES3 in RLCMAIN5) is 1
Traced at Freq=1 GHz (Formal -- C1 in RLCSERIES3 in RLCMAIN5) is 1
Traced at Freq=1 GHz (Symbol -- CVV in RLCSERIES3 in RLCMAIN5) is 3
Traced at Freq=1 GHz (Component Arg -- R of line:3 in RLCSERIES3 in RLCMAIN5) is 4.31662
Traced at Freq=1 GHz (Component Arg -- R1 of RLCSUB in RLCMAIN5) is 4.31662
Traced at Freq=1 GHz (Formal -- L1 in RLCSERIES3 in RLCMAIN5) is 1
Traced at Freq=1 GHz (Formal -- C1 in RLCSERIES3 in RLCMAIN5) is 1
Traced at Freq=1 GHz (Symbol -- CVV in RLCSERIES3 in RLCMAIN5) is 3
Traced at Freq=1 GHz (Component Arg -- R of line:3 in RLCSERIES3 in RLCMAIN5) is 4.31662
Traced at Freq=2 GHz (Symbol -- CV in RLCMAIN5) is 2
Traced at Freq=2 GHz (Component Arg -- R1 of RLCSUB in RLCMAIN5) is 4.31662
Traced at Freq=2 GHz (Formal -- L1 in RLCSERIES3 in RLCMAIN5) is 1
Traced at Freq=2 GHz (Formal -- C1 in RLCSERIES3 in RLCMAIN5) is 2
Traced at Freq=2 GHz (Symbol -- CVV in RLCSERIES3 in RLCMAIN5) is 4
Traced at Freq=2 GHz (Component Arg -- R of line:3 in RLCSERIES3 in RLCMAIN5) is 4.31662
Traced at Freq=2 GHz (Component Arg -- R1 of RLCSUB in RLCMAIN5) is 4.31662
Traced at Freq=2 GHz (Formal -- L1 in RLCSERIES3 in RLCMAIN5) is 1
Traced at Freq=2 GHz (Formal -- C1 in RLCSERIES3 in RLCMAIN5) is 2
Traced at Freq=2 GHz (Symbol -- CVV in RLCSERIES3 in RLCMAIN5) is 4
Traced at Freq=2 GHz (Component Arg -- R of line:3 in RLCSERIES3 in RLCMAIN5) is 4.31662
Traced at Freq=2 GHz (Component Arg -- R1 of RLCSUB in RLCMAIN5) is 4.31662
Traced at Freq=2 GHz (Formal -- L1 in RLCSERIES3 in RLCMAIN5) is 1
Traced at Freq=2 GHz (Formal -- C1 in RLCSERIES3 in RLCMAIN5) is 2
Traced at Freq=2 GHz (Symbol -- CVV in RLCSERIES3 in RLCMAIN5) is 4
Traced at Freq=2 GHz (Component Arg -- R of line:3 in RLCSERIES3 in RLCMAIN5) is 4.31662
Traced at Freq=3 GHz (Symbol -- CV in RLCMAIN5) is 3
Traced at Freq=3 GHz (Component Arg -- R1 of RLCSUB in RLCMAIN5) is 4.31662
Traced at Freq=3 GHz (Formal -- L1 in RLCSERIES3 in RLCMAIN5) is 1
Traced at Freq=3 GHz (Formal -- C1 in RLCSERIES3 in RLCMAIN5) is 3
Traced at Freq=3 GHz (Symbol -- CVV in RLCSERIES3 in RLCMAIN5) is 5
Traced at Freq=3 GHz (Component Arg -- R of line:3 in RLCSERIES3 in RLCMAIN5) is 4.31662
Traced at Freq=3 GHz (Component Arg -- R1 of RLCSUB in RLCMAIN5) is 4.31662
Traced at Freq=3 GHz (Formal -- L1 in RLCSERIES3 in RLCMAIN5) is 1
Traced at Freq=3 GHz (Formal -- C1 in RLCSERIES3 in RLCMAIN5) is 3
Traced at Freq=3 GHz (Symbol -- CVV in RLCSERIES3 in RLCMAIN5) is 5
Traced at Freq=3 GHz (Component Arg -- R of line:3 in RLCSERIES3 in RLCMAIN5) is 4.31662
Traced at Freq=3 GHz (Component Arg -- R1 of RLCSUB in RLCMAIN5) is 4.31662
Traced at Freq=3 GHz (Formal -- L1 in RLCSERIES3 in RLCMAIN5) is 1
```

```
Traced at Freq=3 GHz (Formal -- C1 in RLCSERIES3 in RLCMAIN5) is 3
Traced at Freq=3 GHz (Symbol -- CVV in RLCSERIES3 in RLCMAIN5) is 5
Traced at Freq=3 GHz (Component Arg -- R of line:3 in RLCSERIES3 in RLCMAIN5) is 4.31662
```

You may notice that the non-frequency dependent value `Rv` is only printed (and evaluated) once. All the other values are frequency-dependent so the they are printed as such. Note that all sub-network formal arguments are treated as being frequency-dependent even though they may not be. AGILE will correctly display the hierarchical name so that multiple invocations of a model can be distinguished.

AGILE also allows (syntactically) component assignments to be traced. The syntax looks like:

```
c3 ? cap 3 4 (c=cvv)
```

where again the equal sign is replaced with a `?` between the user reference and the component function. The intent is to use tracing to get "internal" model information, for example, the `Zo` computed for a microstrip TRL (`MSTRL`) could be printed. Unfortunately, the models of current version of AGILE do not provide anything yet – this is a future feature.

3.6 SYMBOLIC (BUILT-IN) VARIABLES

We have already seen one of the symbolic variables allowed in AGILE, namely `FREQ`, which represents the frequency of analysis in Hertz. Although the full use of these variables will not be explained until their appropriate section, they are presented here to complete the NDL discussion. These variable names are *reserved* and they may only be used in the proper context. Here are all of the symbolic variables:

```
FREQ       - frequency (hz) during analysis
NUMFREQ    - number of frequency points
F1BASE     - fundamental (hz) during harmonic analysis
TIME       - time point (sec) during analysis
HARM       - harmonic number during analysis
NUMHARM    - number of harmonics
TEMP       - actual temperature of current circuit
TEMPNOM    - nominal temperature as specified with the SET TEMP command
YIELD      - value of yield as computed from tolerance analysis
```

These variables will be further defined and their use will be demonstrated throughout the remaining chapters and in the examples sections.

4 THE CIRCUIT COMPONENT LIBRARY

There's still more NDL to learn, but before we discuss that let's talk about the models. This chapter will define all of the built-in circuit component functions and their correct parameters. We have seen example usage of these functions in the previous chapters, but as yet have not given a concise definition of their use. In chapter 2, section 9, *CIRCUIT COMPONENT STATEMENTS*, sample NDL syntax for circuit components was introduced. The entire library of built-in models is contained in a separate appendix, but we will discuss here how to read the documentation.

Tip: The models are all defined in a separate document, this chapter defines how to read that document.

4.1 TYPES OF CIRCUIT COMPONENTS

The circuit component library ("library") which is contained in AGILE allows modeling of many microwave/rf components. In order to describe the "currently available" circuit component models, we use a "data sheet" type format. These component data sheets are in alphabetical order by circuit component name as contained in the separate appendix.

4.2 USING COMPONENT DATA SHEETS

Each component sheet contains diagrams which indicate the physical and schematic representation of the components, proper circuit component parameter sets, etc. The physical diagram illustrates the required physical dimensions and attributes of the circuit component. The schematic diagram is used to illustrate the nodal connection order. In order to describe the use of the component sheets, we will use as an example the microwave transmission line. We will split the actual component data sheet into two figures for explanation:

Name: MSTRL

Description: Microstrip transmission line

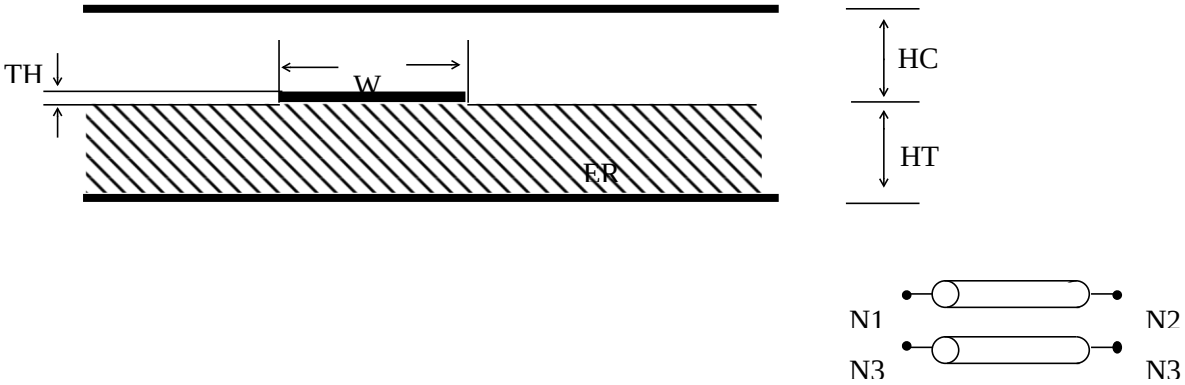


Figure 4.1 A reproduction of part of the component data sheet for the microstrip transmission line.

Figure 3.19 details what the "first part" of a component data sheet contains. In general, this part gives physical and schematic illustrations of the described component, as well as nodal order, etc. In this case, the AGILE name for an microstrip transmission line is given as MSTRL. The order of the nodal connection is shown in the small diagram, while the physical quantities are shown in the drawing.

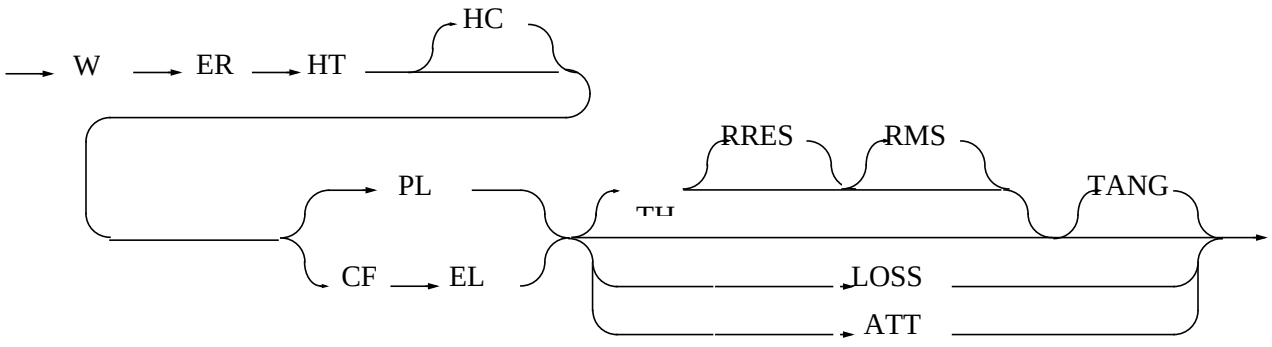
The "second part" of a component sheet details the circuit component parameter set(s) for the component, the microstrip transmission line example is shown in Figure 4.. The nodal connection order is listed as: "*Node Connection Order:*". The ordering is extremely important for the correct connection of any component. In this particular case, the component has symmetry with respect to N1 and N2. In general, this is not the case, because many of the circuit components do not have symmetry. Some examples of the nonsymmetrical components are a voltage controlled generator (`VCG`), or a step in width (STEP). It is very important to keep the order of the node connections exactly as shown in the diagram.

The possible component arguments are listed for the component (note that this generated with the `SHOW MODEL MSTRL` command). Any that are listed as required are just that. However, since argument processing is quite flexible, a component "parameter flow diagram" is shown to exactly define correct arguments.

```
MSTRL   Linear   Microstrip Transmission Line

 Arg Name  Arg-Type  Is-Required   Unit-Type   Unit-Required
 --------  --------  -----------  -----------  -------------
        W  Simple        Yes         DISTANCE     Yes
       HT  Simple        Yes         DISTANCE     Yes
       HC  Simple        No          DISTANCE     Yes
       ER  Simple        Yes             none     n/a
       TH  Simple        No          DISTANCE      No
     LOSS  Simple        No              LOSS      No
      ATT  Simple        No       ATTENUATION      No
     RRES  Simple        No              none     n/a
      RMS  Simple        No          DISTANCE     Yes
     TANG  Simple        No              none     n/a
       CF  Simple        No         FREQUENCY      No
       EL  Simple        No             ANGLE     Yes
       PL  Simple        No          DISTANCE      No
```

Figure 4.2 The remaining part of the microstrip transmission



A parameter flow diagram is used as the method for describing component parameter sets. The interpretation of the flow diagram is simple, just pick the parameters which occur on the path going from the left-hand side to the right-hand side. Any path from the left-hand to right-hand side gives a correct parameter set. Interpreting figure

4.2 specifically: starting at the left and moving in the direction of the arrows toward the right, W, ER, and HT are required; there is no way to proceed other than through these parameters. The description of the symbol and the units for defining the quantities are found in the table of arguments above the flow diagram. To briefly show how these three parameters would appear in NDL:

```
T1 = MSTRL ( W = 5 cm, HT = 10 mils,  ...
```

The values are shown as constants, but may have been written as algebraic expressions or a variable which referred to an algebraic expression (as explained in section 2.9).

Returning to the flow diagram, the next parameter, HC, appears in a branch. The branch indicates two options. If the upper path is taken, HC, the height of the cover is specified. If the lower path is chosen, the microstrip has no cover plate.

The next group of parameters indicates another two options. The upper path uses CF and EL, the center frequency and electrical length. The lower path uses PL, the physical length. Note that although optional, one of these paths must be used.

The last group of parameters contain many options. There are many different combinations of parameters resulting from the interpretation of the paths shown. Here are some of the possibilities:

```
TH, RRES, RMS, TANG
TH, RRES, RMS
TH, RRES, TANG
TH, RRES
TANG
OSS
ATT
TH
```
or nothing.

We have now described how to use the flow diagrams as used throughout the component data sheets. They describe the valid circuit component parameter sets for all components described in the library. However, when defining an "instance" of a circuit component, the order of the parameters is completely arbitrary[9] For example,

```
T1 = MSTRL (W = 5 cm, HT = 10 mils, ER = 10, PL = 10 cm)
```
is the same as
```
T1 = MSTRL (PL = 10 cm, HT = 10 mils, W = 5 cm, ER = 10)
```

We may use the flow diagrams to decide which parameters to use, but we need not use them to decide what order to use. This allows grouping of various parameters into environment statements. Re-iterating section 2.10, environment statements are simply groups of common parameters which can be used in several circuit component instances.

It may be suggested to try to keep the parameter order as similar to the order of the flow diagram as possible. This may aid in finding the missing parameters if AGILE indicates an error of that type exists.

[9] Remember section 2.9.

4.3 ERRORS IN CIRCUIT COMPONENT PARAMETER SETS

AGILE will check to insure that the parameter set is complete and consistent. It also checks that the unit specification is consistent with the parameter type and that all the node connections are specified. Suppose an attempt is made to analyze a circuit with an NDL line with a correct choice of parameters, but with an illegal unit choice on one of them?

for example, the erroneous line:

```
T1 = MSTRL(PL= 10 deg, HT= 10 mils, W= 5 cm, ER= 10)
```

will result in:

```
Error - Wrong Unit for Argument (PL)
```

The unit `deg` for the `PL` is wrong. There are many other errors associated with erroneous circuit component statements. Here are some of them:

4.3.1 SOME ERROR MESSAGES RELATED TO MODELS

```
Error - No Such Model (model-name)
Error - Environment (env-name) is Missing
Error - Index Unexpected for Argument (arg-name) Error - Duplicate Argument (arg-name) Illegal
Error - Index Required for Argument (arg-name)
Error - Index.Index Required for Argument (arg-name)
Error - Unit Required for Argument (arg-name)
Error - Wrong Unit for Argument (arg-name)
Error - Argument (arg-name) Unrecognized for this Component
Error - Argument (arg-name) is Required but Missing
```

It is urged for the reader to refer to the appendix, where these error messages are more fully explained. Next are the set of units that can be used, these are also defined in the Appendix.

4.3.2 TABLE OF UNITS

The following are the units that can be used, their type and the conversion to the MKS system.

```
HZ    -- unit of FREQUENCY   -- to_mks = (1)
KHZ   -- unit of FREQUENCY   -- to_mks = (1000)
MHZ   -- unit of FREQUENCY   -- to_mks = (1e+06)
GHZ   -- unit of FREQUENCY   -- to_mks = (1e+09)
PH    -- unit of INDUCTANCE  -- to_mks = (1e-12)
NH    -- unit of INDUCTANCE  -- to_mks = (1e-09)
UH    -- unit of INDUCTANCE  -- to_mks = (1e-06)
MH    -- unit of INDUCTANCE  -- to_mks = (0.001)
H     -- unit of INDUCTANCE  -- to_mks = (1)
PF    -- unit of CAPACITANCE -- to_mks = (1e-12)
NF    -- unit of CAPACITANCE -- to_mks = (1e-09)
UF    -- unit of CAPACITANCE -- to_mks = (1e-06)
MF    -- unit of CAPACITANCE -- to_mks = (0.001)
F     -- unit of CAPACITANCE -- to_mks = (1)
OHMS  -- unit of RESISTANCE  -- to_mks = (1)
KOHMS -- unit of RESISTANCE  -- to_mks = (1000)
MOHMS -- unit of RESISTANCE  -- to_mks = (1e+06)
UMHOS -- unit of CONDUCTANCE -- to_mks = (1e-06)
```

```
MMHOS -- unit of CONDUCTANCE -- to_mks = (0.001)
MHOS  -- unit of CONDUCTANCE -- to_mks = (1)
DEG   -- unit of ANGLE       -- to_mks = (0.0174533)
RAD   -- unit of ANGLE       -- to_mks = (1)
ASRM  -- unit of DISTANCE    -- to_mks = (1e-10)
UIN   -- unit of DISTANCE    -- to_mks = (2.54e-08)
MILS  -- unit of DISTANCE    -- to_mks = (2.54e-05)
IN    -- unit of DISTANCE    -- to_mks = (0.0254)
FT    -- unit of DISTANCE    -- to_mks = (.3048)
UM    -- unit of DISTANCE    -- to_mks = (1e-06)
MM    -- unit of DISTANCE    -- to_mks = (0.001)
CM    -- unit of DISTANCE    -- to_mks = (0.01)
M     -- unit of DISTANCE    -- to_mks = (1)
NM    -- unit of DISTANCE    -- to_mks = (1e-09)
DB/M  -- unit of LOSS        -- to_mks = (1)
DB/WL -- unit of ATTENUATION -- to_mks = (1)
SEC   -- unit of TIME        -- to_mks = (1)
MSEC  -- unit of TIME        -- to_mks = (0.001)
USEC  -- unit of TIME        -- to_mks = (1e-06)
NSEC  -- unit of TIME        -- to_mks = (1e-09)
PSEC  -- unit of TIME        -- to_mks = (1e-12)
```

5 OPTIMIZATION

> *Tip*: You know enough to try some examples. Jump to section 9 to try Agile quickly. Then come back and learn about the remaining features.

The *objective function value* is a mathematical formulation representing the "quality of performance" of the network as a function of the quantities which are to be varied. Optimization is the varying of parameter values used to define a network in trying to reduce the value of the objective function. This chapter describes how to define the "objectives" (objective function) through goal definition statements and the *fmerit* statement, as well as how to use the *optimize* command in AGILE. Due to the algebraic methods used in goal definition, AGILE is sufficiently generalized to successfully approach non-network optimization problems as will be shown.

This chapter divides the description of this broad topic into six large sections, with a seventh "summary" section. The first section details the definition of "goals", which is the first stage of the definition of the objective function in AGILE. The second describes the definition of the actual objective function using the *fmerit* statement. The third section describes how *target variables*, the values which the optimizer will change, are defined. The fourth section describes the *optimize* command in AGILE, starting one of the optimizers. The fifth section defines the *termination conditions* for the optimizer; that is, after the optimization has begun, what makes it stop? The sixth section (briefly) illustrates using optimization in AGILE, and the seventh provides a summary and concluding remarks.

5.1 GOAL DEFINITION STATEMENTS

AGILE sub-divides the definition of the objective into the definition of one or multiple *goal statements*. Goal statements are part of AGILE NDL and as such may appear anywhere in the network description. The (purposeless) keyword *GOALS*, however, is used in nicely formatted NDL to introduce these statements.

This section first defines the generalized syntax used in goal statement formulation, followed by examples. In this section the goal statement is first defined as an AGILE NDL statement and is then defined in mathematical terms. This is followed by an example which demonstrates the syntax and serves to illustrate additional graphics capabilities of AGILE with respect to the goal definition statement. Getting started, here is the generalized syntax of goal definition statements:

```
goal-variable goal-type expr1  [ : expr2 [ : • • • ] ]
    where goal-type might be one of:
          SUM             SUMNORM
          SUMSP           SUMNORMSP
          MAXIMUM         MAXIMUMNORM
          MAXIMUMSP       MAXIMUMNORMSP
          UPPERLOWER      UPPERLOWERNORM
          UPPERLOWERSP    UPPERLOWERNORMSP
```

Figure 5.1 Goal Statement Syntax

The syntax defined in Figure 5.1 Goal Statement Syntax indicates that a goal statement is a variable name, called the *goal variable*, followed by one of several *goal-types* followed by at least one or possibly several expressions separated by colons. The essence of these statements is to define a "goal" as a mathematical relationship between actual network performance and a desired performance. Each goal defined in a goal statement represents a single

value, called the *goal-error*, and is determined by the *goal-type* language element and the expression(s) used. The *goal-type* determines the proper number of expressions needed for the goal definition. The mathematical definition for each of the twelve *goal-type*s is:

Goal-Type	Number of expr(s)	Definition
SUM	2	$\sum_{points} \lvert expr_1 - expr_2 \rvert$
MAXIMUM	2	$\max_{points} \lvert expr_1 - expr_2 \rvert$
UPPERLOWER	3	$\sum_{points} \begin{cases} \lvert expr_1 - expr_2 \rvert & if\ expr_1 > expr_2 \\ \lvert expr_1 - expr_3 \rvert & if\ expr_1 < expr_2 \\ 0 & if\ expr_2 > expr_1 > expr_3 \end{cases}$
SUMSP	4	$\sum_{points} (expr_3 \times \lvert expr_1 - expr_2 \rvert)^{expr_4}$
MAXIMUMSP	4	$\max_{points} (expr_3 \times \lvert expr_1 - expr_2 \rvert)^{expr_4}$
UPPERLOWERSP	5	$\sum_{points} \begin{cases} (expr_4 \times \lvert expr_1 - expr_2 \rvert)^{expr_5} & if\ expr_1 > expr_2 \\ (expr_4 \times \lvert expr_1 - expr_3 \rvert)^{expr_5} & if\ expr_1 < expr_2 \\ 0 & if\ expr_2 > expr_1 > expr_3 \end{cases}$
SUMNORM		same as SUM but divided by number of points
MAXIMUMNORM		same as MAXIMUM (name provided for convenience)
UPPERLOWERNORM		same as UPPERLOWER but divided by number of points
SUMNORMSP		same as SUMSP but divided by number of points
MAXIMUMNORMSP		same as MAXIMUMSP (name provided for convenience)
UPPERLOWERNORMSP		same as UPPERLOWERSP but divided by number of points

Figure 5.2 Table of Goal-Type definitions [NOTE: number of points either number of frequencies or number of time points, depending on datum.

The "goal error" of the SUM form is interpreted as representing the sum at each frequency of the absolute value of the difference between *expr*1 and e*xpr2*. The goal error of the MAX form is interpreted as representing the maximum absolute difference between *expr1* and *expr2* across all frequencies. And the UPPERLOWER form, although expressed somewhat awkwardly represents the need to place upper- and lower-bounds upon data. Specifically, the data is represented by *expr*1 while the upper-bound is represented by *expr2* and the lower by *expr3*. The NORM-form of each of these goal types is the same except that its value is divided by the number of frequencies. Having mathematically defined the some of the goal-types, the discussion continues with illustration of their use.

```
The network listed in Network : Opt1
*
* Adjust width and length of microstrip transmission line
* for 50 ohms and electric length 150 degs @ 15 GHz
*
FREQUENCIES:
  10 20 1 GHz
PORTS:
  #1 1 0
  #2 2 0
ENVIRONMENT:
  Ev1 : ht = 20 mils, er = 10
COMPONENTS:
  halfwidth = 6.4 [ 0.01 : 100.0 ]
  length = 100 [ 0.01 : 300.0 ]
  T1 = MSTRL(ENV=ev1, PL=length mils, W=2*HalfWidth mils)
CONNECTIONS:
  T1 1 2 0
OUTPUTS:
  trans = dB(s21)
  transferangle = APHASE( PHASE(s21) )
GOALS:
  goal1 SUM trans : 0.0
  goal2 SUM transferangle : APHASE( -10*freq/1e9 )
  Fmerit = goal1 + goal2
END
```

Figure 5.3 called opt1 will provide a starting point for the discussion of goal statements.

```
Network : Opt1
*
* Adjust width and length of microstrip transmission line
* for 50 ohms and electric length 150 degs @ 15 GHz
*
FREQUENCIES:
  10 20 1 GHz
PORTS:
  #1 1 0
  #2 2 0
ENVIRONMENT:
  Ev1 : ht = 20 mils, er = 10
COMPONENTS:
  halfwidth = 6.4 [ 0.01 : 100.0 ]
```

```
  length = 100 [ 0.01 : 300.0 ]
  T1 = MSTRL(ENV=ev1, PL=length mils, W=2*HalfWidth mils)
CONNECTIONS:
  T1 1 2 0
OUTPUTS:
  trans = dB(s21)
  transferangle = APHASE( PHASE(s21) )
GOALS:
  goal1 SUM trans : 0.0
  goal2 SUM transferangle : APHASE( -10*freq/1e9 )
  Fmerit = goal1 + goal2
END
```

Figure 5.3 The opt1 network.

The NDL description of this network is all familiar (right?), with the exception of the two goal statements under the `GOALS:` keyword and Fmerit statement. Notice the *range variables:* `halfwidth` and `length` (refer to section 2.8 if necessary). The range variables will become the values that the optimizer will change, this is further defined in the next section, this section describes the goal statements. We have defined two goal variables using the goal definition statement, named `goal1` and `goal2,` please make note that the general form does not require names that begin with "goal" or even contain the string "goal", they might have also been called `MagnitudeOBJ` and `PhaseOBJ`.

As indicated in the NDL comments, we are trying to use optimization to adjust the width of a 20 mil high microstrip transmission line of dielectric 10 to give an impedance of 50 ohms, and to adjust the physical length to approximate an electric length of 150º at 15 GHz. The approach taken is to use two goal statements, one of which is used to achieve the 50 ohm requirement and the other of which is used to achieve the correct length required. Specifically, goal1 is the SUM at each frequency of the difference between the decibel magnitude of S21 and zero. The reduction of this goal to zero would effectively require that the transmission line be Zsys (default 50 ohms) at all frequencies. We could have also used dB(S21) rather than one thousand times this quantity, which we did in the next example (opt1); we will explain this later. There are many other ways of specifying that the transmission line be 50 ohms; i.e., make the *VSWR* at either port equal to one at all frequencies (which is a different formulation of the desired characteristic). Before exploring other possibilities, the proper and complete understanding of this example is needed.

The goal statement defining `goal2` is used to achieve the desired phase. Specifically, `goal2` represents the `SUM` of the difference between the "absolute phase of *S21*" and the equation for the phase angle of an ideal transmission line of electrical length 150º at 15 GHz, which varies linearly with frequency. As you might already be imagining, there are also alternative goal definitions for this phase requirement.

A powerful use of the graphics capabilities in AGILE is the direct displaying of the goal variables. This provides visualization of goal definition. Specifically, after the creation of the network `opt2` above, if AGILE is asked to plot goal1 and goal3, a graphical representation as in Figure 5.4 will result.

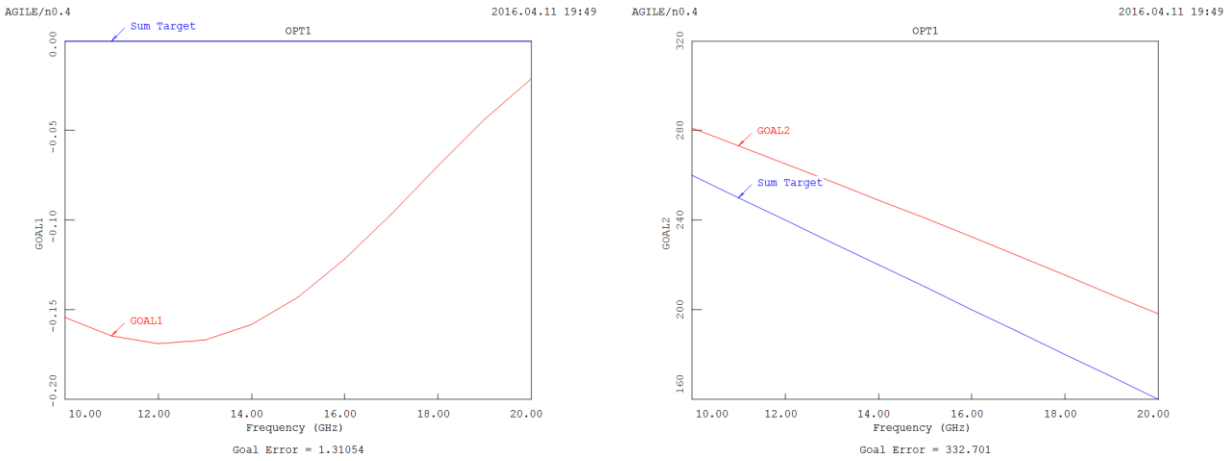


Figure 5.4 The result of the plot goal1 and goal2 command for network opt1, before optimization

The left side of the Figure 5.4 graphic is a rectangular plot of the two expressions used to define goal1 versus frequency. The goal-value, 1.31054 in this case, is printed at the bottom of the graph, and as the table in Figure 5.2 indicates, this value may be obtained by adding (`SUM`) the difference between these two curves at each frequency point. On the right side of the figure, the plot of `goal2` shows the desired and actual S21 phase for the frequency band and the error which is the sum of the difference between these curves. In each case, the curve labelled `SUM TARGET` is the evaluation of the second expression while the other curve is evaluation of the first (labelled "*GOAL*" something). The goal error represented by each goal variable is printed at the bottom of each graph.

How to form the objective function from the goal errors has not yet been discussed nor how to begin optimization. Actually, even what the "optimization variables" are hasn't been covered yet, but you may have (correctly) assumed that the range variables are to be the optimization variables. These final topics are covered in the next two sections. However, since the correct definition of goals is critical to successful use of AGILE optimization, continued illustration of goal definition is beneficial.

To demonstrate the `MAXIMUM` form of goal statement, suppose `goal1` definition in network `opt1` were changed to:

```
goal1  MAXIMUM  trans : 0.0
```

The expressions specified are the same as original `SUM` form previously used, only the goal-type has been changed. In this case the graphic of `goal1`, if requested, would be identical to that in figure 5.1.4 with the exception that the goal error would be `0.1692` (approximately). Notice that this is the value of the maximum difference between the curves across the frequency range in Figure 5.4.

In illustration of the goal-type form, `UPPERLOWER` suppose the network `opt2` was created (as a modification of `opt1`) to state:

```
fg = freq / 1e9   ;* Fg is freq in GHz
fs = 0.15
UpperLimit = 1.2+(1.8-fs*fg)*UNIT(12 - fg) + (fs*fg - 2.7)*UNIT(fg-18)
LowerLimit = 1.0
goal3 UPPERLOWER vswr1 : UpperLimit : LowerLimit
```

The assignment for `upperlimit` uses the `UNIT` function to partition the expression into three parts, this can be done equivalently as:

```
UpperLimit = if( fg < 12, 3.0-fs*fg, if( fg > 18, -1.5+fs*fg, 1.2 ))
```

which uses the "`if`" function to also partition the function into three parts. The figure below illustrates the graphic result of plotting `goal3` (whichever `upperlimit` form is used). The halfwidth was changed to a value of 3 too:

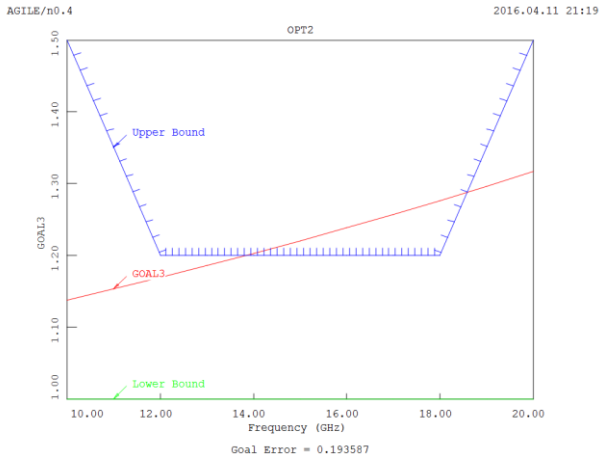


Figure 5.5 The result of the plot goal3 command for network opt2.

An explanation of how the goal error (about 0.193 in this case) is computed in this form is useful. Some of the goal curve is within the upper and lower bounds curves. But some areas are not. These contribute to the goal error in a sum form.

If, in the evaluation of the upper and lower bounds, AGILE discovers that the lower bound is greater than the upper, a warning message, which includes at which frequency point this occurred, is generated reflecting this fact (obviously, this cannot be allowed). If all data points lie between the upper and lower bounds the goal error is zero.

We have now illustrated the three goal-type forms (`SUM`, `MAXIMUM`, `UPPERLOWER`) with examples. These examples illustrate usage with *real-valued* expressions, however, AGILE also allows *complex-valued* expressions to be used in goal statement definitions (depending upon the goal-type, as will be seen). Jumping right to an example, suppose the `goal4` definition in the `opt2` network were added as:

```
elen = 150
cfreq = 15
target = ptor( 1.0 + (0,1)*(-freq*( elen/cfreq )/1e9 ) )
goal4 SUM s21 : target
```

In this NDL fragment, the definition of the complex valued `target` is made which imitates the *S21* parameter of an ideal 50 ohm transmission line with an electrical length of 150° at 15 GHz (in rectangular form). `Goal4` then simply becomes the `SUM` at each frequency of the vector difference between the data (`s21`) and the expression for target. This might be a good time to refer back to the Figure 5.2 table and notice the mathematical definition of the goal-type `SUM`. The definition is still valid for complex data if the absolute value signs are read as "magnitude difference". AGILE maintains the display capabilities of goal variables for complex data, Figure 5.6 below

illustrates plotting `goal4`. Please note that this display was "enhanced" using `display` commands in turning the markers on, only for added clarity.

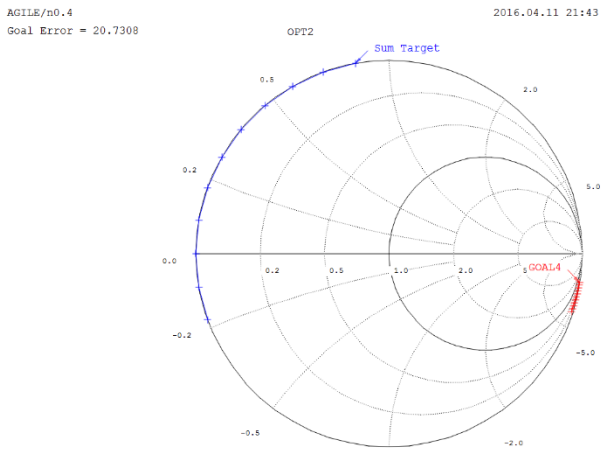


Figure 5.6 The result of the plot goal4 command for network opt2.

This figure shows two curves representing the two expressions defined in the goal formulation. The curve labelled `GOAL4` represents the first expression, in this case `s21` while the second is labelled `SUM TARGET` (marked with crosses) represents evaluation of the second expression. This labelling is exactly the same as with rectangular data. The goal error evaluation is printed on the graph, in this case approximately `20.73`, this is computed as the sum of the vector differences at each frequency. So with these changes, `opt2` is now:

```
Network : Opt2
*
* Adjust width and length of transmission line
* for 50 ohms and electric length 150 degs @ 15 GHz
*
* Approximate Answers: halfwidth = 10.1, length = 122
*
FREQUENCIES:
 10 20 1 GHz
PORTS:
 #1 1 0 ; #2 2 0
ENVIRONMENT:
 Ev1 : ht = 20 mils, er = 10
COMPONENTS:
 halfwidth = 3 [ 0.001 : 50.0 : 10 ]
 length = 10 [ 0.001 : 300.0 : 30 ]
 T1 = MSTRL(ENV=ev1, PL=length mils, W=2*HalfWidth mils)
CONNECTIONS:
 T1 1 2 0
OUTPUTS:
 trans = dB(s21)
 transferangle = APHASE( PHASE(s21) )
GOALS:
 fg = freq/1e9 ;! freq in GHz
 fs = 0.15
```

```
 UpperLimit = if( fg < 12, 3.0-fs*fg, if( fg > 18, -1.5+fs*fg, 1.2 ))
 LowerLimit = 1.0
 goal1 SUM trans : 0.0
 goal2 SUM aphase(transferangle) : aphase(-10*fg)
 goal3 UPPERLOWER vswr1 : UpperLimit : LowerLimit
 elen = 150
 cfreq = 15
 target = ptor( 1.0 + (0,1)*(-freq*( elen/cfreq )/1e9 ) )
 goal4 SUM s21 : target

 FMERIT = 1000*goal1 + goal2 + 1000*goal3
END
```

Figure 5.7 The opt2 network

As you may have already imagined, the `MAXIMUM` formulation is also applicable to complex-valued data. If the definition of `goal4` were changed to:

```
goal4  MAXIMUM  s21 :  target
```

the resulting graphic would be the same as in Figure 5.6, since the goal expressions are the same, except that the goal error would be approximately `2` (which is the largest vector difference at each frequency point between the curves – almost complete across the chart).

The `UPPERLOWER` form does not allow complex data, as may have already been guessed. Since no possible definition for this form for complex valued data has been devised, AGILE will issue appropriate error messages when complex-valued evaluation is used in this form. Both complex and real valued expressions are allowed for the `SUM` and `MAX` forms. AGILE also allows "mixed" expressions (i.e., *expr*1 is complex and *expr*2 is real). In these cases, the real-valued data is converted to complex and the operations are performed.

Any goal definitions defined in sub-networks or sub-models are ignored during optimization, only those specified in the current network are effective.

We have now defined and illustrated several goal-type forms. The syntax used for goal definition is always the same: *variable goal-type* followed by one or several expressions depending upon the goal type used. The three additional goal types are: `SUMNORM, MAXIMUMNORM, UPPWERLOWERNORM`. These goal function formulators perform the same job as their counterparts, except that the goal error is divided by the number of frequencies.

5.2 THE FMERIT STATEMENT

Having defined the required goals, the next step in preparation for optimization is formulation of the *objective function*. In AGILE the objective (or error) function is called `FMERIT` (roughly standing for *figure-of-merit*). This value may be assigned using an assignment statement. It has been present in the above NDL examples but not yet discussed. For example in the original (Figure 5.3) `opt1` case above, the user may want the figure of merit to be the sum of each of the two goal errors. In this case the NDL would look like:

```
FMERIT = goal1 + goal2
```

assigning `FMERIT` the value of the sum. Since the values of `goal1` and `goal2` were approximately `1.31054 and 332.70`, the value for `FMERIT` would be `334.012`. Keeping in the spirit of case insensitivity,

the variable FMERIT, as with any other variable, may appear in any (or mixed) case. If the *maximum* of the two goals were desired as the objective function, a statement such as:

```
Fmerit = goal1 MAX goal2
```

would be used, making use of the MAX binary operator. In this case, the value for FMERIT would be 332.70, the maximum of goal1 and goal2. The expression used to define fmerit need not be restricted to expressions of goal statements. For example, suppose in the opt1 case above it was desired to make the value for halfwidth as small as possible. A fmerit statement reflecting this might look like:

```
Fmerit = goal1 + goal2 + 10*halfwidth
```

This effectively adds a contribution from the halfwidth variable to the objective function. The fmerit expression is completely arbitrary and may be any expression, however it may not be complex-valued or frequency dependent. In the context of the original opt1 network, either of these will result in an error:

```
Fmerit = goal1 + trans                  -- error trans is frequency dependent      or
Fmerit = goal1 + (0,1)*sqrt(length)  -- error - expression is complex-valued
```

The first of which is illegal because trans represents the dB(s21), which is a frequency dependent value -- this is not allowed, fmerit must represent a single real-value. The second case above is illegal since it is complex-valued. Some care must be exercised in expressing the figure of merit. Specifically, since one of the termination criteria in the optimizers is that the value of fmerit is equal to or below zero, the fmerit expression should be formed such that its value is positive and the objective(s) have been met as fmerit → 0.

Notice in opt1 that the goal goal1 is much smaller than goal2. This suggests that perhaps a scaling of the goals is in order, so look at the expression for Fmerit in opt2.

```
FMERIT = 1000*goal1 + goal2 + 1000*goal3
```

Goal1 has been scaled to 1000x its value as has goal3. This would result in a better optimization, assuming that these goals are of about the same importance. More on this subject in the next sections.

5.3 TARGET VARIABLES

The values which must be changed during the optimization are called *target variables*. As alluded to previously, the *range variables* in AGILE are in fact the target variables. The NDL fragment, from Figure 5.7 above:

```
halfwidth = 3 [ 0.001 : 50.0 : 10 ]
length = 10 [ 0.001 : 300.0 : 30 ]
```

shows two range variables, again refer to section 2.7 if necessary. In the context of optimization, these variables are called target variables. Defining some terminology: in the case of the variable halfwidth, the current value is 6.4, the *lower bound* is 0.01, and the *upper bound* is 50.0 (similarly for length).

In the current AGILE implementation, any target variables (range-variables) specified in sub-networks/models are *not part of the optimization process.* These variables are evaluated at their current value. However, optimized variables may be passed as arguments into sub-models as described in section 4.1, thus affecting sub-model performance during optimization. Also: any range variables specified in global networks are also considered target variables.

The "dimension" of the optimization problem is the number of target variables being optimized, in the case of the `opt2` network the dimension is 2. Thus the more target variables (or range variables if you prefer), the higher the problem dimension. The "current point" of the optimizer is referred to as a vector, of length equal to the dimension.

Each vector element represents one of the target variable values. Thus the current point specified above in vector form is: `( 3, 10.0)`. The need for this additional terminology will become apparent later in the chapter.

5.4 PERFORMING OPTIMIZATION

Having established the variables to be optimized and the figure of merit to be minimized, executing the optimization takes place on the `Optimize` tab:

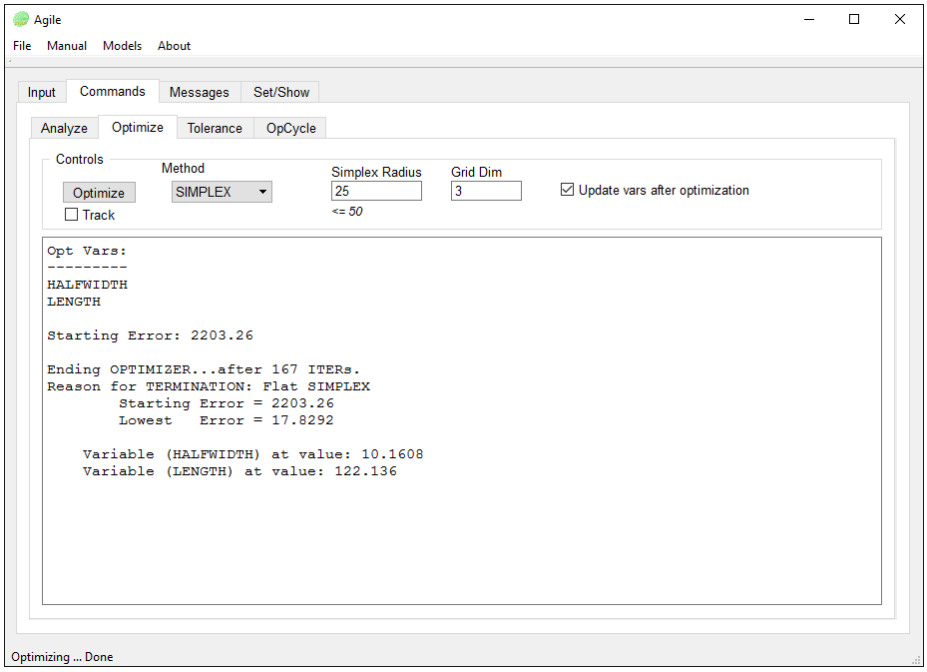


Figure 5.8 Optimization Result. The method is a combobox selection and parameters needed for the Simplex method and Grid are shown in textboxes. Pressing the Optimize button will start the optimization.

This section details the initialization of optimization, while the next section details the *termination reasons* in optimization; i.e., reasons that the optimization process will stop.

As indicated in Figure 5.8, possibilities to choose from include the `SIMPLEX`[10], `DAVIDON`[11], `CONJGRAD`[12], `BFGS`[3] optimizers, as well as `GRID`. Optimizers fall into two broad categories, *direct* and *gradient*. The simplex method is a direct optimizer, while the others (Davidon, Conjgrad, BFGS) are gradient. All optimizers (as referenced below)

[10] J.A. Nelder and R. Mead, "A simplex method for function minimization", Computer Journal, Vol. 7, 1965

[11] W.C. Davidon, "Optimally Conditioned Optimization Algorithms Without Line Searches", Mathematical Programming 9 (1975), pp. 1-30, North-Holland Publ. Co.

[12] ACM TOMS ROUTINE #500, written by Shanno and Phua

implement optimization in the context of objective function definition defined above. Additional discussion of the optimizers is postponed.

The value `"Simplex radius"` specifies "how wide the search" should be initialized if the Simplex method is chosen. The **`simplex`** method uses N+1 vectors (where N is the problem dimension) during optimization. The initial points placed in the simplex are the *current point*, as well as points "plus or minus" a percentage based upon the radius number and the range specified for each variable. If the current value is in the lower half of the range allowed for each variable then the "plus" side is used, otherwise if the value is in the upper half the "minus" is used. The percentage is based upon the range of the variable. For example, here are the three initial values for the original `opt2` problem if a radius of `0.25 (25%)` were used would be (approximately):

```
halfwidth,length => (3,10) (15.5,10) (3,85)
```

The range specified for `halfwidth` is from `0.001` to `50.0`, 25% of this is approximately 12.5, therefore, since the current value of `3` is in the lower half of the range, 3+12.5 = 15.5 is used. Similarly, the range for `length` is approximately `300`, 25% percent of this is 75 and `10+75 = 85` is used as its increment. The other optimizers do not currently use the radius information for initialization.

The Davidon, Conjgrad and BFGS optimization methods are gradient search types. The simplex optimization method is well suited for problems in which the current point is far from the optimum. These methods are better suited for problems in which the current point is "close" to the optimum value. A thorough discussion of this topic is beyond the scope of this document.

The **`GRID`** optimizer is quite different from those described. It divides the search space based upon the 'Grid Dim' on the form. Each of these divisions is then taken in combination with the divisions of each of the other variables to form points to analyze. These points, as well as the current point are analyzed in the "grid optimization". In the context of the original `opt2` network, the result would divide the range for both `halfwidth` and `length` into three divisions. In general,

number of points = 2 + N^M *-- the N to the power M points plus 1 init and 1 conclude point*

where **N** is the integer specified and **M** is the dimension of the problem.

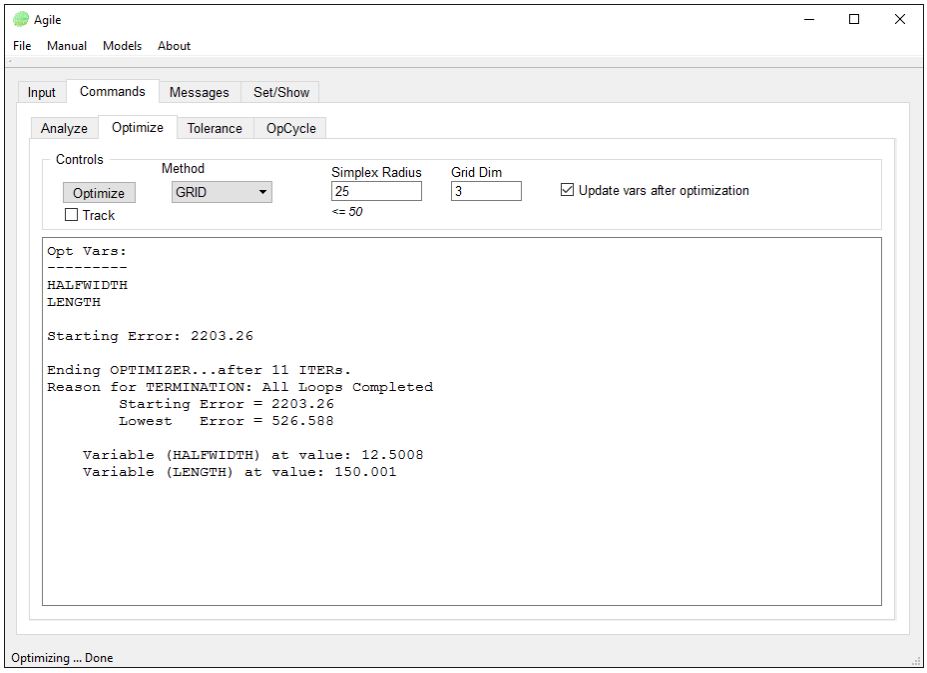


Figure 5.9 Result of 'Grid' optimization on opt2

Due to the combinatorial nature of the `GRID` optimizer, many points may be generated. For example, a problem with 7 variables "gridded" into 5 divisions would generate number of points = $2 + 5^7 = 78127$. Obviously, this is quite extreme. The `GRID` optimizer is recommended in cases where a real "rough cut" at the problem is needed, both the number of divisions and the problem dimensions must be kept low. The other times the `GRID` optimizer is recommended is when computer time is free over weekends, or when you have a super-computer lying around with nothing to do.

The `optimize` command option includes two check marks. The track option will create a printout each time a better point is found. For opt2 and the Simplex method, this is the ending portion:

```
( 152 ) Evaluations...Improved Value: 17.8292
Variable (HALFWIDTH) at value: 10.1607
Variable (LENGTH) at value: 122.137

( 156 ) Evaluations...Improved Value: 17.8292
Variable (HALFWIDTH) at value: 10.1609
Variable (LENGTH) at value: 122.136

( 158 ) Evaluations...Improved Value: 17.8292
Variable (HALFWIDTH) at value: 10.1608
Variable (LENGTH) at value: 122.136


Ending OPTIMIZER...after 168 ITERs.
Reason for TERMINATION: Flat SIMPLEX
Starting Error = 2203.26
Lowest Error = 17.8292
```

```
Variable (HALFWIDTH) at value: 10.1608
Variable (LENGTH) at value: 122.136
```

Showing that the best point found was on iteration 158 with an optimization value (Fmerit's evaluation) of 17.8292.

The other option is '`Update vars after optimization`'. With this checked the best values found for the variates are written back into the NDL file. For example, this would be the result after this optimization:

```
HALFWIDTH = 10.1608 [ 0.001 : 50.0 : 10 ]
LENGTH = 122.136 [ 0.001 : 300.0 : 30 ]
```

You should place optimization variates on a single NDL line, that is don't use the semicolon to have multiple lines on one text line or the update will fail.

In summary, we have defined the operation of the `optimize` command. We leave this section with many questions, like *"What objective function should I use for my problem?"* and *"Which optimizer will work better on my problem, and how should I initialize it?"*. Many of these questions are very application dependent and cannot be answered here; hopefully, further illustration using the `opt2` network, and the last section, *COMMENTS, CAVEATS AND RECOMMENDATIONS* can shed more light on the subject. This section has fully defined how to start the optimizer, while the next section details the reasons that the optimization will stop.

5.5 OPTIMIZATION TERMINATION

Once the optimizer is running, there are several reasons for it to stop. These reasons are called the termination conditions. A list of the possible termination conditions are:

```
Too Many ITERS
Too Many ITERS without Improvement
Error Reduced Near Zero
Flat SIMPLEX
Round-Off - Relax GRADERROR
BREAK DETECTED
Net-Analysis ERROR during OPT
```

Figure 5.10 Optimization termination conditions

The first termination condition, `Too Many ITERS`, stops the optimizer after a fixed number of *iterations*. An iteration in the context of network optimization is an evaluation of the network outputs, goals, and of course the objective function. The second termination reason, `Too Many ITERS without Improvement`, stops the optimizer whenever a fixed number of iterations are performed without finding a better point. These values are set in the config file, see that section. Typical values might be 500 or 1000 for maximum iterations and 100 or so for max iterations without improvement.

The third reason for termination, `Error Reduced Near Zero`, stops the optimizer if the objective function is less than 10^{-5}. Suppose your objective function is *typically* on that order? Remember, with all the algebraic flexibility for `fmerit` definition that the objective error can be "scaled" so that the termination criteria is not limiting. `Fmerit` should always be formed such that it is greater than zero, using the algebraic flexibility in defining the figure of merit, this can always be done.

The next termination criteria, `Flat SIMPLEX`, occurs only when using the simplex method (obviously?). This occurs if the error values associated with the N+1 vectors in the simplex are all within 10^{-5} of each other. This *may* mean that the best point has been found, but the simplex reference should be consulted for further information.

The next termination reason, `Round-Off - Relax GRADERROR` occurs only when using the conjugate gradient or BFGS optimizers. It means that the requested gradient error has not been achieved (see next section). It will generally occur at the minima, if however, this occurs before the minima has been reached, then the GRADERROR term should be relaxed.

The next termination, `BREAK DETECTED`, will only occur if the optimization is interrupted.

> Tip: This version does not currently provide a Break function. It is a single threaded app.

The final possible termination, `Net-Analysis ERROR during OPT`, results when the optimizer attempts analysis and cannot for some reason. Prior to termination of any of the AGILE optimizers, a message will be displayed which is one of the errors normally associated with analysis, all are in the appendix. An example might be, if the value used as a width of a transmission line is allowed to become less than or equal to zero.

Remember, it is the responsibility of the user to establish the limits of the target variables.

5.6 THE OPTIMIZE COMMAND ILLUSTRATED

This section will provide example usage of the optimization capabilities of AGILE, in a more tutorial format than the previous sections. Suppose the previous `opt2` network has been established with the definition in Figure 5.7 as the current network; it is included in the AGILE distribution. After optimization, the variables where updated as:

```
HALFWIDTH = 10.1608 [ 0.001 : 50.0 : 10 ]
LENGTH = 122.136 [ 0.001 : 300.0 : 30 ]
```

Of course, now that this better point has been found, the goal variables may be plotted to see "how we did". Compare Figure 5.4, which is a plot of `goal 1 and goal2` before the above optimization, to Figure 5.11, which is after. You may notice the dispersive effect of the microstrip transmission line compared to the ideal target specified.

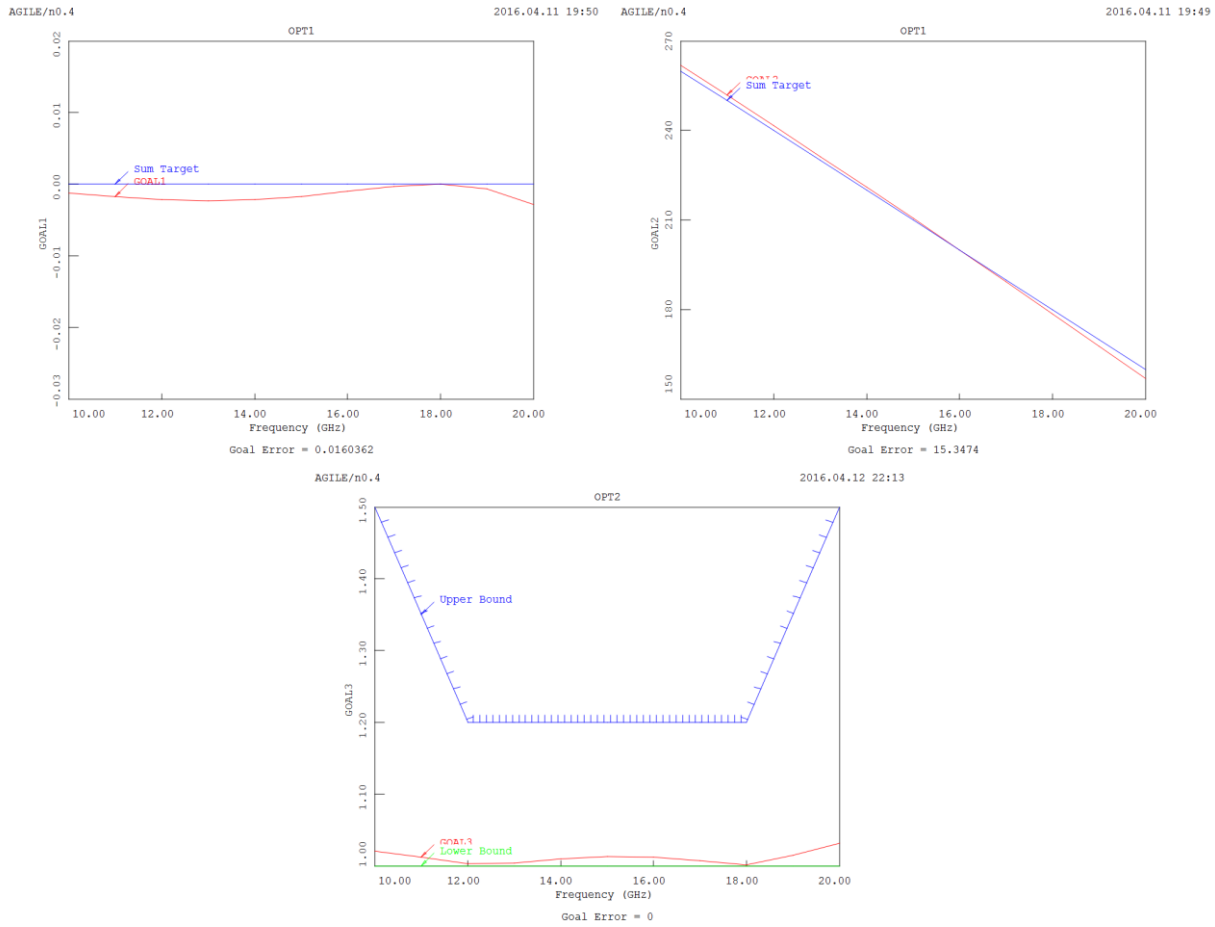


Figure 5.11 The result of the plot goal1 2 and 3 for network opt2 after optimization

These plots show that the goal values have been reduced, since they are contributors to fmerit.

A discussion of the network `opt1` was promised. One problem with it is that goal scaling is not very good. Assuming that each of its goals are of equal import, they should be scaled as such. Specifically `goal2` is based upon the angles in degrees. These differences are on the order of magnitude in the 10's and 100's, while the differences for `goal1` are based upon the decibel scale, perhaps on the order of magnitude of 0.1's. This is why the `opt2` network has `goal1` "weighted" by a factor of one thousand.

Also care must be taken when using angles (phase values) for comparisons. Using the APHASE function for each angle value will make sure that the same relative phase (e.g. in degrees angles might be on a 0 to 360 or a -180 to 180 scale) is used.

As mentioned in the introductory remarks of this chapter, another application of optimization in AGILE is to non-network problems. There has been no mention that goals need be based upon network dependent quantities, and in fact they need not be. Here is an example from the financial domain which uses optimization to "invert" an expression, that is solve for an independent variable for a given result.

```
Network: Math
* Example of the applicability of AGILE to finding solutions
* of transcendental equations which are independent of circuits.
```

```
* Optimization is used.
* Problem: If the cost of producing 'x' items is given by the
* expression for 'CX' below, what is the value of 'x'
* for the given 'CX'.

X = 359.728 [ 50 : 800 ]
CX = 100.5 + 0.0277*X - 1503.6/X + 5.12*EXP( X/413.7 )
DesiredCX = 118.50
Fmerit = abs(cx-desiredcx)

END
```

The calculation of `x` cannot be computed directly to give a particular cost per item, because the expression for `CX` cannot be inverted to give an expression giving `x` as a function of `CX`. The optimizer is used to match the calculated cost, `CX`, to the desired cost, `DesiredCX`. This is done directly by defining `fmerit` to be the absolute value of the difference between the calculated cost and the desired cost. Use of any of the AGILE optimizers in determining `x` is appropriate (the "answer" is `x = 359.7` by the way). Notice that goal statements were not used (but could have been) in solving this problem. Since this problem is not a function of frequency, `fmerit` can be formed directly from the data.

Other possible non-network applications of optimization is in the *curve-fitting* area. The goal statement must be formulated as the difference between the calculated curve and the data points. As an example, suppose a linear approximation to the following data were needed:

X	DATAE(x)
1	2.8
2	2.9
3	3.1
4	3.1
5	3.5
6	3.6

Figure 6.7.2 Some data

The first step is to write an NDL description for DATAE versus *frequency* in GHz, rather than X. This due to the fact that we will want to `SUM` the differences between the actual data and fitted curve across the independent variable (`freq`). The GHz scale is used rather than a Hz scale to allow convenient plotting. This can be done through the proper use of the `DIRAC` function. Specifically:

```
Network : math2
 1 6 1 GHz
 fg = freq/1e9
 Datae =   Dirac(fg-1) * 2.8      &
```

```
        + Dirac(fg-2) * 2.9      &
        + Dirac(fg-3) * 3.1      &
        + Dirac(fg-4) * 3.1      &
        + Dirac(fg-5) * 3.5      &
        + Dirac(fg-6) * 3.6
 Ap = 0 [ -100 : 100 ]
 Bp = 0 [ -100 : 100 ]
 Cdatae = Ap + Bp*fg
 LinearMatch SUM  Cdatae : Datae
 Fmerit = LinearMatch
END
```

Figure 5.12 The MATH2 network, an example of curve-fitting.

The `Datae` variable takes on the actual data point values for each value of frequency, while the `Cdatae` variable respresents the linear curve that we wish to approximate it with. Optimization is used to determine the values of `ap` and `bp` which best satisfy the goal defined by `LinearMatch`. After the optimization, the values: `ap = 2.6` and `bp = 0.16666` are found with an objective function value of *0.3* (the initial objective function value was *19*). The `plot LinearMatch` command after optimization illustrates the results:

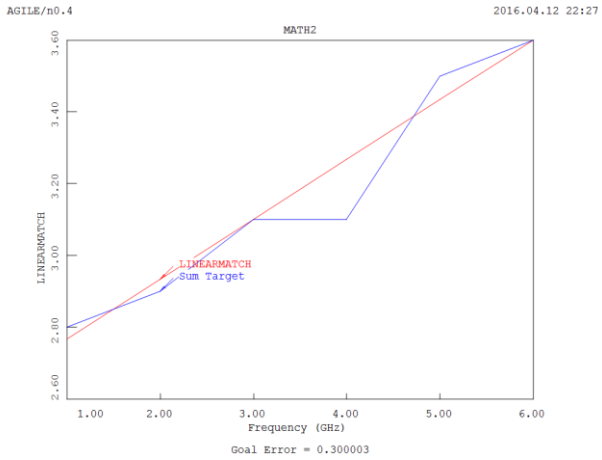


Figure 5.13 The result of the plot LinearMatch command.

The `LinearMatch` goal could be modified to best fit the squares of the difference, rather than the linear difference:

```
BestSquares  SUM  ( Datae - Cdatae ) ** 2 : 0.0
```

which has the same basic intent but a different mathematical meaning. The best fit to this data by a quadratic-curve might be found by changing the appropriate lines in Figure 5.12 to:

```
Ap = 0 [ -100 : 100 ]
Bp = 0 [ -100 : 100 ]
Cp = 0 [ -100 : 100 ]
Cdatae = Ap + Bp*fg + Cp*(fg**2)
QuadraticMatch  SUM  Cdatae : Datae
Fmerit = QuadraticMatch
```

However, the results for neither of these will be demonstrated here. These simple examples can be used as the basis for more exotic possibilities. Notice that goal statements are necessary here since the data to be matched is an array (across frequency). Thus the goal statements may be viewed as *data collectors*. Also note that the NDL description may contain as many as goal statements as desired, while defining `fmerit` as a function of only a few (or none). This means, in the examples above that:

```
Ap = 0 [ -100 : 100 ]
Bp = 0 [ -100 : 100 ]
Cp = 0 [ -100 : 100 ]
Ldatae = Ap + Bp*fg
Cdatae = Ap + Bp*fg + Cp*(fg**2)
LinearMatch     SUM  Ldatae : Datae
QuadraticMatch  SUM  Cdatae : Datae
*Fmerit = LinearMatch
Fmerit = QuadraticMatch
```

all of the lines above may be contained in one NDL description, using the commenting function to switch between which type of fit is desired.

The figure below further demonstrates optimization using a microstrip branch-line power divider as an example.

```
* Optimize branch-line coupler.
*
Network : lay7
Frequencies:
    14.5 15.5 0.2 ghz
Ports:
    #1  1  0
    #2  2  0
    #3  3  0
    #4  4  0
Environment:
    ev1: ht = 20 mils,  er = 10
Components:
   * VARIABLES :
    w50    =  20.0 [ 15 : 25 ]
    w35    =  30.0 [ 15 : 50 ]
    w70    =  10.0 [  5 : 20 ]
    l90w35 = 100.0 [ 50 : 200 ]
    l90w70 = 100.0 [ 50 : 200 ]
    l1w50  = 1
   *
    ptrl50 = mstrl(env=ev1, w= w50 mils, pl= l1w50  mils)
    ptrl35 = mstrl(env=ev1, w= w35 mils, pl= l90w35  mils)
    ptrl70 = mstrl(env=ev1, w= w70 mils, pl= l90w70  mils)
    tjct1 = mstee(env=ev1,w1=w50 mils,w2=w35 mils,w3= w70 mils)
    tjct2 = mstee(env=ev1,w1=w35 mils,w2=w50 mils,w3= w70 mils)
Connections:
    ptrl50  1 10     0
    tjct1  10 11 12  0
    ptrl70 12 15     0
    ptrl35 11 17     0
```

```
    ptrl50  2 13     0
    tjct2  14 13 15  0
    ptrl35 14 20     0
    tjct1  19 20 21  0
    ptrl70 21 18     0
    tjct2  17 16 18  0
    ptrl50  3 16     0
    ptrl50  4 19     0
Outputs:
    Direct   = dB(s31)
    Coupld   = dB(s41)
    Isolated = dB(s21)
    MagDif   = Direct - Coupld
    PhaseDif = APHASE( phase(s31) - phase(s41) )
Goals:
    goal0  SUM  (direct+3.01)**2       : 0.0
    goal1  SUM  (coupld+3.01)**2       : 0.0
    goal2  SUM  (magdif)**2            : 0.0
    goal3  SUM  ((phasedif-90)/10)**2  : 0.0
    FMERIT = goal0 + goal1 + goal2 + goal3
END
```

The `lay7` network defines a single section 3-dB branch-line power divider, with three widths and two physical lengths as target variables. It includes the microstrip "tee" effects. A fixed length transmission line 1 mil long is used as a "lead-in" component to the *tee's*. It is a named component `ptrl50`. The interesting part of this network is the description of the goal statements. After some experimentation with various goal definitions, these seem to generate very good optimization results for the branch-line network. The first three goals defined, `goal0, goal1` and `goal2`, all relate to magnitude performance requirements, while the last goal, `goal3`, represents a phase requirement. Specifically, the goal error from `goal0` is the sum at each frequency point of the square of the difference between 3.01 dB and the dB transfer of the "direct"-arm. Similarly, the goal error from `goal1` represents the sum of the squares difference between the desired 3.01 dB performance and the "coupld"-arm. Additionally, the goal error `goal2` represents the sum of the squares of the dB-difference between of these two quantities. The reduction of error from these three goals all require that each of the arms reach 3.01 dB and be equal. A good question concerns the "redundancy" of the `goal2` statement, *"If each arm of the power divider is forced to 3.01 dB, then isn't the added fact that they be equal implied, making `goal2` unnecessary?"*. This is true, however, any modification of the goals, which ultimately alters the objective function value, may produce different results. The inclusion of `goal2` alters (strengthens) the objective function with respect to meeting magnitude goal requirements.

The `goal3` goal statement is used to achieve the proper 90° phase requirement. Specifically, it is the sum across each frequency of one tenth of the `APHASE` of the phase difference between the direct and coupld arms minus 90 squared. This too has been found to produce good results for phase requirements included in the objective function.

Here are before optimization and after optimization of the key outputs:

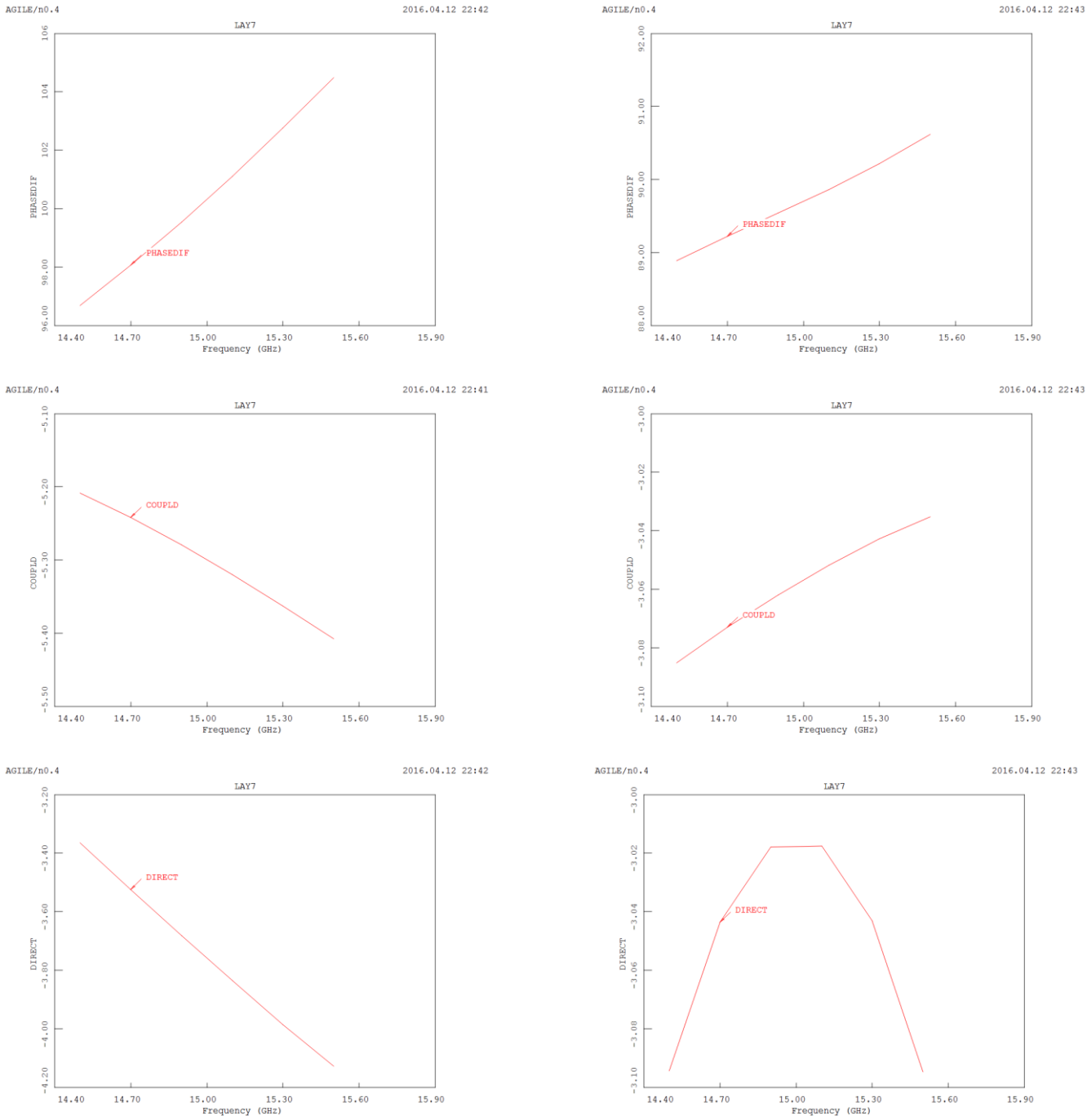


Figure 5.14 Before and after optimization of the lay7 branch-line coupler. Note that scaling is different in each.

The performance of coupler is greatly improved. The solution found with Simplex is

```
Ending OPTIMIZER...after 809 ITERs.
Reason for TERMINATION: Flat SIMPLEX
Starting Error = 56.9048
Lowest Error = 0.0650374
```

```
Variable (L90W35) at value: 50.0001
Variable (L90W70) at value: 82.2482
Variable (W35) at value: 32.0737
Variable (W50) at value: 24.9998
Variable (W70) at value: 19.9999
```

Suppose the definition for `fmerit` were changed to:

```
Fmerit = goal0 MAX goal1 MAX goal2 MAX goal3
```

The evaluation of `fmerit` would then be the maximum of the goals. This is what is typically called a *min-max* optimization. A min-max optimization attempts to minimize the maximum of the goals. The term min-max optimization is in a sense a misnomer. It is not an *optimization technique* but rather an *objective function formulation*. However, it is typically referred to as an optimization technique in other literature. Obviously, this objective function is easily achieved in AGILE as shown above. The type of error function many times used in circuit optimization is called *least-pth*. The 'SP' forms of the goal functions are typically used to formulate this error function, however, this can also be formed directly in the figure of merit expression:

```
Fmerit = goal0**3 MAX goal1**3 MAX goal2**3 MAX goal3**3
```

Returning back to the original definition of `fmerit` (the sum of the goals), here is the result of starting the conjugate-gradient optimizer:

```
Opt Vars:
---------
         L90W35 -> start value = 100
         L90W70 -> start value = 100
            W35 -> start value = 30
            W50 -> start value = 20
            W70 -> start value = 10

Starting Error: 56.9048

Ending OPTIMIZER...after 351 ITERs.
Reason for TERMINATION: Round-Off - Relax
GRADERROR
        Starting Error = 56.9048
        Lowest   Error = 0.16689

    Variable (L90W35) at value: 61.6956
    Variable (L90W70) at value: 82.3074
    Variable (W35) at value: 28.8922
    Variable (W50) at value: 18.7255
    Variable (W70) at value: 17.0321
```

```
Opt Vars:
---------
         L90W35 -> start value = 100
         L90W70 -> start value = 100
            W35 -> start value = 30
            W50 -> start value = 20
            W70 -> start value = 10

Starting Error: 31498.9

Ending OPTIMIZER...after 332 ITERs.
Reason for TERMINATION: Too Many ITERS
without Improvement
        Starting Error = 31498.9
        Lowest   Error = 94.1729

    Variable (L90W35) at value: 76.0296
    Variable (L90W70) at value: 104.001
    Variable (W35) at value: 27.4224
    Variable (W50) at value: 19.9229
    Variable (W70) at value: 10.3538
```

Figure 5.15 Results of CONJGRAD optimization on `lay7` *(left) and* `lay8` *(right)*

In Figure 5.14, the conjugate-gradient optimizer is used for `lay7` (left side) and `lay8` (right side). First, we can compare the results for lay7 (left) to the solution found for this circuit by the Simplex method above; the Simplex

method got the error down to 0.065037 while the conjugate-gradient method reached 0.16689. Both are good, but notice also that the resulting values for the variables are quite different. The lay8 network is trying to achieve the same thing, albeit using a different optimization goal formulation. The Fmerit values aren't comparable to the lay7 network, but we see that rather different variable values are found.

5.7 COMMENTS, CAVEATS AND RECOMMENDATIONS

Optimization is a broad subject, with hundreds of papers written specifically on circuit optimization and certainly thousands on the more general subject of function optimization. Clearly, this subject cannot even be scratched here. However, goal definition in AGILE, as well as the objective function formulation from the resulting goal errors (`FMERIT`) should be well understood. Examples in this chapter have been admittedly scarce, network optimization is so application dependent that it cannot be addressed here. The strength of AGILE optimization is in the algebraic form for goal definition and the visualization of such. Virtually any specification for network characteristic(s) can be expressed in this powerful syntax. As pointed out in the prior section, care must be taken in forming the Fmerit expression, optimizations with similar circuit goals can end with quite different optimization variable settings.

Another point that might be useful to note, notice that there is really no difference between the target and the data specifications. There is absolutely no restriction on making the target data static (fixed) as the optimization progresses. The allowing of the target(s) to change as well as the data during optimization might be called *goal motion*. There are many possibilities for this, for example, fit the upper and lower bound to data which is fixed, rather than move the data to fit the upper and lower bounds. While the implications of goal motion are beyond the scope of this manual, everything needed to explore this topic has been discussed.

Another topic of interest for those more familiar with optimization is the method which AGILE uses to limit the target values to their specified ranges. Specifically, AGILE optimizers use either of two methods for this purpose: *penalty constraints* or *transformation*. Although this is also beyond the scope of this document, penalty constraining (rescale=false) limits the optimizer to only choosing values within the range(s) for each target variable(s) by assigning a very large objective function value whenever evaluation is attempted outside these range(s). This effectively makes the optimizer stay within the bounds required. When tracking the optimization, the user may notice objective function values on the order of 10^8 or 10^9, this means that the penalty is being invoked. In this case, the user *may* want to stop the optimization and readjust the upper and lower bounds for the variable(s) violating the specified constraint.

In general, it is not advised to begin optimization with variables near their "edges", it is recommended that variables be set near their center points, at least to start with. It is also important that reasonable bounds be placed on the variables. For example, don't set an inductor value to a range of 0 to 1000 if you expect that the optimum is actually in a range of 1 to 10 nH.

All of the necessities associated with optimization have been defined, although few examples have been shown. There is no substitute for imagination, the flexibility is there.

6 APPLIED SIGNALS AND TIME DOMAIN OUTPUT

In addition to the *frequency domain* features previously described, AGILE also incorporates very generalized *time domain* features. These features allow the AGILE user to define signals in either the time or frequency domain, apply them to any port(s) with the result being computation of either time domain signals or frequency domain spectra at node(s) of the network. The computations are carried out in the frequency domain with appropriate conversions taking place.

6.1 SIGNAL DEFINITION

The first step in using the time domain capabilities of AGILE is the definition of a signal. A signal in AGILE is defined as either a voltage or a current in either the time or frequency domain. Voltage signals are always defined in units of volts, and current signals are always defined in units of amps. A signal statement is used to define them. The general syntax is:

```
SIGNAL sig-name [VOLTAGE | CURRENT | MW | DBM] [TIME | FREQ] sig-def
   where
          sig-def is either an expression or list of points/values.
```

Lists are curly brace enclosed with the "x" point first followed by a colon, followed by a value expression. As many points as needed may be put in the list.

The `VOLTAGE, CURRENT, MW` or `DBM` part of the phrase determines the type of signal being defined. The word `VOLTAGE` or `CURRENT,` etc. must be spelled out entirely. However, the case of these words is irrelevant (i.e., `Voltage, voltage, VoLtAgE` all mean the same thing). Next comes the domain part, this must be either `TIME` or `FREQ`. As mentioned the signal definition may be either an arbitrary expression or defined in terms of a list of values. Examples are:

```
SIGNAL V1 voltage time 1+1/(time*1e9+1)
signal Acurrent current time {1e-12 : 0.1 }
F_off = 0.5e9     ;* half a gig
SIGNAL v_in voltage freq { F1BASE : ptor(1.2,-30) } {F1BASE+F_OFF : -3 }
SIGNAL i_bias1 DBM freq 0.1*dirac(12e9-freq)
SIGNAL i_bias2 DBM freq { 12e9 : 0.1 } ;* same as above
```

In the examples above, `V1, v_in` are defined as a voltages and `Acurrent` is defined as a current and `i_bias1` and `i_bias2` are defined as a power sources. Also, `V1, Acurrent` are defined in the time domain, while `v_in, i_bias` are defined in the frequency domain. Signals may be complex-valued if defined in the frequency domain, but must be real-valued in the time domain. If the "list format" is used then there is no restriction of the number in the list. Note in the case of `v_in` above that the special variable `F1BASE` is used. Remember from chapter 2, that this is a special variable which represents the base harmonic frequency. Also important to note: the definition of signals does not drive the selection of harmonics or frequencies used in the analysis, the frequency statement(s) or harmonic statement control this.

Note the use of the reserved variable:

```
time
```

above. Just as `freq` is a reserved variable used to represent the current frequency, `time` is a reserved variable which represents a time point. Also, as the symbol `freq` represents frequency in Hertz, regardless of the frequency definition units, `time` represents time in seconds, regardless of time units.

Another consideration in the definition of signals concerns the range of `time` over which the expression is evaluated. Although there exists methods to display information over *any* range of time, the expression is always evaluated over the range: `[0,`*`period`*`]`. This means that the expression used to define the signal should exist over this range. Since we have not defined what the `period` is as yet, we will continue this discussion in section 6.3.

6.2 ATTACHING SIGNALS TO PORTS

Having established one or more signals, the next step in the use of time domain features is the *attachment of a signal to a port*. In general, AGILE signal(s) are applied to one or more ports and the resulting time domain signals or frequency spectra inspected as output. The outputs are described further in section 6.5. The next step is to attach a signal to a port. The *port definition* statement initially defined in chapter 2 section *PORT STATEMENTS*, and then extended in chapter 3 section *NON-DEFAULT PORT TERMINATIONS*, is now further extended. The port statement also allows the signal name to be added after the port nodes. As an example, suppose the 1.2 volt, 500 MHz (at 0 degrees) sine wave defined above were to be attached to port 1 in network `testsig1`:

```
Network : testsig1
  f1  = 500e6                ;* f1 is the frequency in hertz
  SIGNAL sineInput voltage freq {f1 : 1.2}     ;* OK
PORTS:
   #1 1 0 sineInput          ;* attached
       •••
```

In the fragment above, a sine wave input is defined and attached to port 1. Specifically, the phrase, `#1 1 0 sineInput,` performs the attachment. Naturally, it is an error to attach a non-signal to a port, and an appropriate error message would be generated.

Referring back to the correct example for `testsig1`, port 1 has the *system impedance* has its port termination. As described in chapter 3, a non-default termination may also be specified. Signals may be attached to ports with non-default terminations. This is illustrated by:

```
Network : testsig2
  SIGNAL sineInput voltage freq {500e6 : 1.2}
PORTS:
  #1 1 0 sineInput :dload(){ 5 6 } ;* dload term,sineInput attached
      •••
```

In this case, nodes `5` and `6` from the network `dload` are specified as the port termination and the signal `sineInput` is attached to port 1. In the case above, `dload` is a sub-network, signal attachment is still legitimate for sub-models. Thus complete generality is maintained in the signal attachment process.

At this point, it is necessary to know exactly what signal attachment means. AGILE uses a *non-ideal* model to stimulate the network. Figure 6.1 illustrates signal attachment in the case of a voltage signal. Notice that the voltage defined in the signal is not directly attached to the port, unless the port termination network is DIRECT. The voltage defined in the signal is a generator voltage, this voltage is applied through the *portequiv*, the two node reduction of

the specified port termination. If no port termination network is specified, then the *port-equiv*, as shown in the figure, would be the system impedance. In cases where port has a termination network, the equivalent network there is used.

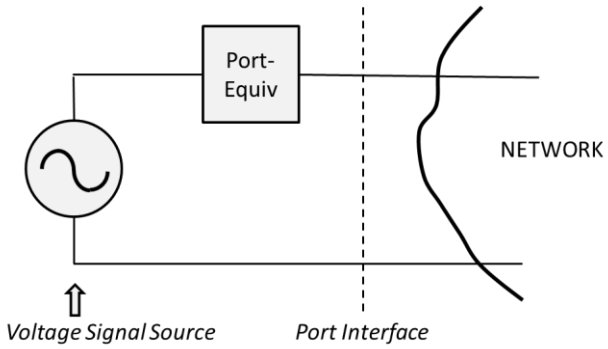


Figure 6.1 Voltage Signal Attachment Model

The attachment of a "`mW`" signal or "`dbm`" signal follows the voltage model above, by conversion to a voltage drive. The `mW` and `dBm` signal specify the *available* power, not the actual power transfered at the port (one could use the `pspectrum` output to compute the power actually delivered at the port. The following expressions are used to convert a milliwatt or dbm specification into a voltage:

$$v = \sqrt{\frac{2((R_Z + R_{SYS})^2 + I_Z^2)\frac{mw}{1000}}{R_Z}}$$

$$v = \sqrt{\frac{2((R_Z + R_{SYS})^2 + I_Z^2)\frac{10^{(dbm/10)}}{1000}}{R_Z}}$$

where

Rz = `real part of port termination impedance`
Iz = `imaginary part of port termination impedance`
Rsys = `system impedance (real)`

Figure 6.2 similarly illustrates the model used in application of a current signal. Notice that this too uses a generator model, the current source is attached in parallel with the *port-equiv* of the port termination. Similar to the voltage generator model, the current generator model uses either the system impedance or a port termination network as the port-equiv, depending upon user specification.

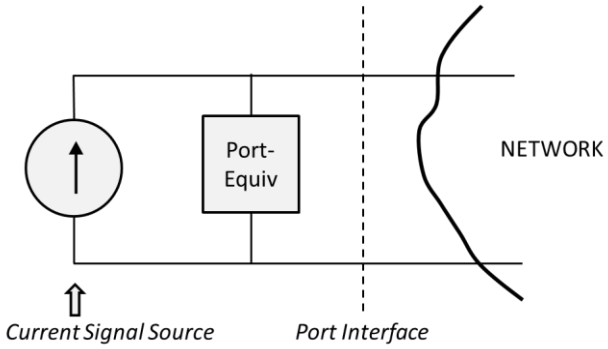


Figure 6.2 Current Signal Attachment Model

The attachment of signals in AGILE is very generalized. One, many, or all ports of a network may have attached signals. Any mixture of attached signals, either voltages or currents, in either domain is allowed. This means that a voltage signal may be applied to one port, while simultaneously a current signal is applied to another port. The results are obtainable through the use of the variables described next. Following this, we explain how the period, etc. is computed.

6.3 SPECTRAL AND SIGNAL OUTPUTS

Several special "output variables" are used to examine output resulting from the application of the above signals. The list in Figure 6.3 summarizes the signal related output variables. Remember, these variables are only meaningful in the context of applied signals. Of course, nonlinear analysis is also only performed in the presence of applied signals.

PSPECTRUM*i* is the frequency domain power in Watts at node *i* and is the multiplication of the voltage and current spectrums.

VSPECTRUM*i* is the frequency domain voltage in volts at node *i*.

ISPECTRUM*i* is the frequency domain current in amps at node *i*.

VSIGNAL*i* is the time domain voltage in volts at node *i*.

ISIGNAL*i* is the time domain current in amps at node *i*.

Figure 6.3 The five signal-related output variables

6.4 DETERMINATION OF THE PERIOD AND RESOLUTION

As described in section 6.1, the time domain simulation in AGILE is periodic. The frequencies used for the analysis, in combination with the two user set factors, determine the period and the resolution of the simulation of the time domain data.

HERE

The two user factors are TPERIODFAC and TSAMPLEFAC, these are set-able using the "set tperiodfac *integer*" or the "set tsamplefac *integer*" commands, respectively. In the case of TPERIODFAC, the integer setting must be >= 1, and for TSAMPLEFAC the setting must be

>=2. The defaults are TPERIODFAC=1 and TSAMPLEFAC=4. The following expressions relate the time domain period and resolution (time-step) to the frequencies of analysis and the factors TPERIODFAC and TSAMPLEFAC:

$$period = \frac{T_{periodfac}}{F_{min}}$$

$$resolution = \frac{1}{2T_{samplefac}F_{max}}$$

Figure 6.4 Expressions relating frequency points to time domain period and resolution

We can see from the expressions in Figure 6.4, that the TPERIODFAC directly factors the number of periods in the time domain, and that TSAMPLEFAC directly factors the resolution in the time domain. Note the general form, the minimum frequency (F_{min}) used in the analysis controls the period, while the maximum frequency (F_{max}) controls the resolution (along with the factors described above). Note that in cases where the minimum frequency is 0 (d.c.) that the next higher frequency is used in the computations. Remember that either the harmonics statement or frequency statements control what the frequencies are. Also note that the frequency steps don't enter into the computation, however, they are needed for two purposes. The frequency steps will obviously control the frequency set, which in turn will determine the "base" frequency used to determine the harmonics. The base frequency is the greatest common divisor for the frequency set (within a tolerance). In case where 0 Hz (DC) is used as part of the analysis, then the next highest frequency would (partially) determine the base frequency.

An example is in order, suppose a circuit is analyzed where the minimum frequency is 10 GHz and the maximum is 20 GHz and that the defaults TPERIODFAC=1 and TSAMPLEFAC=4 are in effect. In this case, the period would be 0.1 nsec and the resolution would be 0.00625 nsec or 6.25 psec. The number of time points would be 0.1 nsec / 6.25 psec = 16 and one is added, so that 17 time points would result.

Remember: the frequency(s) of the analysis are determined independently of the setting of those within the signals, is is your responsibility to achieve meaningful results. For example, if your frequency analysis points are 1, 2, 3, and 4 GHz and signals are applied at 10 GHz, all time domain and spectral plots will be zero! The user must also take care to prevent aliasing, etc.

6.5 TIME DOMAIN EXAMPLE NETWORKS

Having established the entire basis for time domain analysis in AGILE, we proceed with two examples. The first being a simple LC circuit, stimulated with a pulse train input, the second is more complicated, a three winding transformer which uses two input stimuli.

6.5.1 EXAMPLE 1 - PULSE RESPONSE OF AN LC CIRCUIT

In this example, a simple LC circuit is defined. A pulse input is applied to the circuit and the output(s) observed. We would like to apply a pulse train with a magnitude of one, a period of 4 nsec and 50% duty cycle to the input of an LC circuit. The network is called `sig2`:

```
Network : sig2
*SIGNALS:
    Tjump = 2.0
    SIGNAL Ptrain voltage TIME unit(time-Tjump*1e-9)
PORTS:
    250 10000 250 MHz
    #1 1 0  Ptrain
    #2 2 0
COMPONENTS:
    Tr = 20 [ 10:30:5 ]
    T1 = ind(l=Tr nH)
    C1 = cap(c= 5 pF)
CONNECTIONS:
    T1 1 2
    C1 2 0
 OUTPUTS:
    t1volt = Vsignal1
    t1curr = Isignal1
    t2volt = Vsignal2
    t2curr = Isignal2
    f1s    = mag(Vspectrum1)
    f2s    = mag(Vspectrum2)
END
```

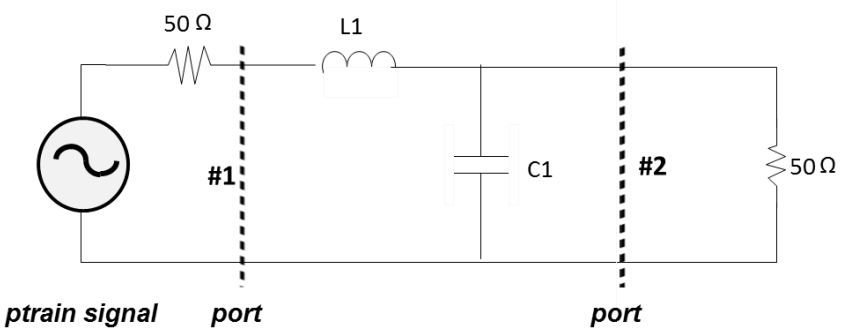


Figure 6.5 Listing and schematic for sig2

The network `sig2` is listed and pictured in Figure 6.5. Notice that the signal generator is applied according to the port termination diagrams previously shown. Since no port termination network or component is specified, it is Zsys which is 50 ohms (the default, but changeable on the settings page) in this case.

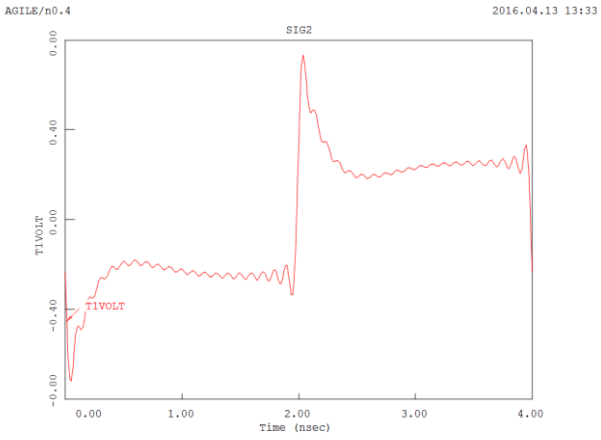


Figure 6.6 The result of plot t1volt

Since the minimum frequency is 250 MHz, the time period for the conversion will be 4 nsec (and since `TPERIODFAC=1`), and the resolution will be 1/(2 * 4 * 10 GHz) = 12.5 psec (since `TSAMPLEFAC=4`).

The graph shown in Figure 6.6 will result when `t1volt` is plotted. This is the voltage at the port, not that of the generator (however, the "`DIRECT`" keyword can be used as the port termination, in which case the signal is directly applied). Notice the "ringing" in the signal, due to the lack of sufficient high frequency data. This can be seen when inspecting Figure 6.8 later.

The output voltage signal is shown Figure 6.7, namely `t2volt`. Similar commands can be used for the other time domain signal outputs, `t1curr` and `t2curr`, but will not be demonstrated here. Also note that the d.c. component in these signals is zero. This is to be expected since 0 was not included in the frequency set. If it were, by simply changing the frequency statement to `"0 10000 250 MHz"`, then there is a d.c. component in both the input and output voltages. This will not be demonstrated.

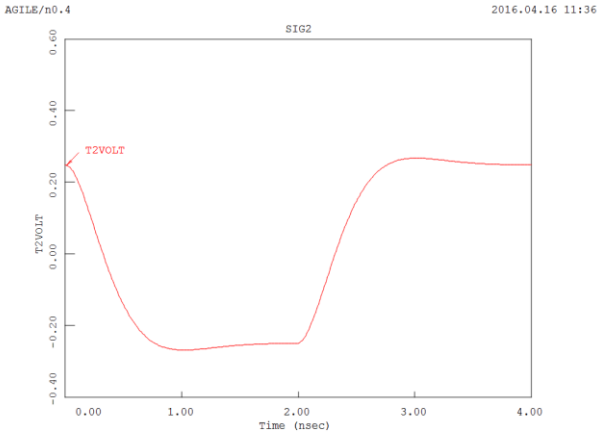


Figure 6.7 The result of plot t2volt

Let's continue by plotting the voltage spectrum based outputs, `f1s` and `f2s`. These are shown in Figure 6.8.

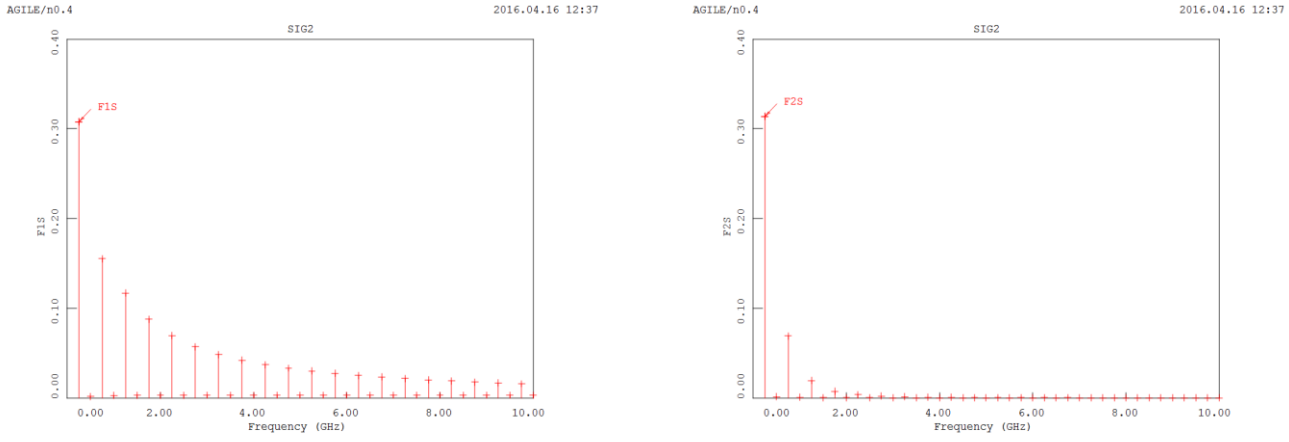


Figure 6.8 Result of plotting f1s and f2s (after adjusting the graph type and markers)

Notice that in both figures that even harmonics are not present. This is because these harmonics were not present in the applied `ptrain` signal, a pulse signal, to begin with. Also note that the harmonic content in `f1s` has not completely diminished, hence the high frequency ringing visible in time domain voltage signal (for node 1). Since the spectra for `f2s` has vanished within the frequency set used, ringing is not present. You can use the TperiodFac and TsampleFac on the Set/Show tab to adjust how time domain signals are displayed – for example setting TperiodFac to 2 will show two periods (cycles) in the time domain, while setting TsampleFac can be used to adjust time domain sampling intervals. However, the fundamental data from which time domain signals are derived is completely determined from the frequency (harmonic) data set.

Tip: Skip to Chapter 9 if you are more interested in non-linear circuit analysis

6.5.2 EXAMPLE 2 - TWO STIMULI TO A THREE WINDING TRANSFORMER

In this example, a network containing a three winding transformer is stimulated with a positive half wave sine waveform on one winding and a negative half wave sine on another winding. We will view the results at all of the ports. The network `multi` below is the basis for discussion here, it is a three-winding transformer network:

```
Network:multi
 #1 1 2  vsig1  res(r=50)
 #2 3 4  vsig2  res(r=50)
 #3 5 6         res(r=200)
 0 500 100 mhz
 tr1=transformer( num=3,coil1=100nh,coil2=100nh,coil3=400nh,       &
                        coef1.3=.99,coef2.3=.99,coef1.2=.99)
 tr1 1 2 3 4 5 6
 SIGNAL vsig1 voltage time if( time < t1, sin(2*pi*f1*time), 0 )
 SIGNAL vsig2 voltage time if( time > t1, sin(2*pi*f1*time), 0 )
 pi = 3.14159
 f1 = 100E6
 t1 = 1/(2*f1)
 vout1 = vsignal1 ;* ref to node 2
 vout2 = vsignal3 ;* ref to node 2
 vout3 = vsignal5 ;* ref to node 2
END
```

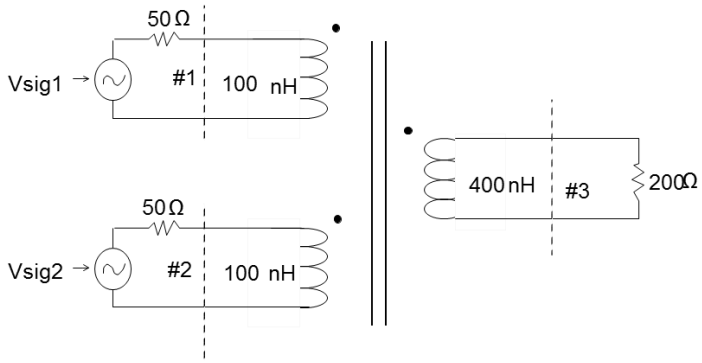


Figure 6.9 NDL and schematic for the `multi` *network*

The configuration used is similar to that used in a "push-pull" amplifier. Note that since the terminations for each of the ports is different, each specifying a different resistive component. The next figure shows plots for each of `vsig1` and `vsig2` which are attached signals for ports 1 and 2 respectively. These make use of the '`if`' operator in NDL to create different portions of a sine wave, as pictured in Figure 6.10.

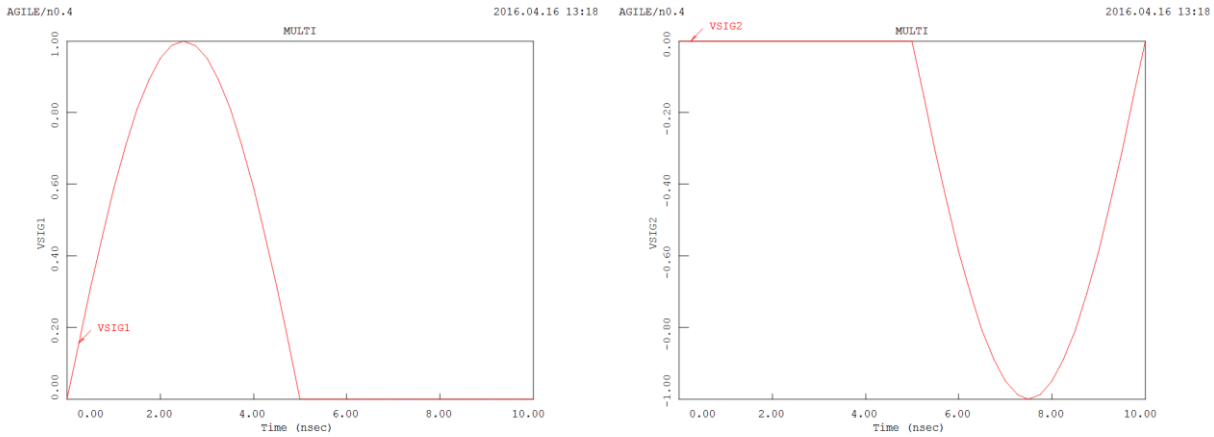


Figure 6.10 Plot of `Vsig1` and `Vsig2` in circuit `multi`

The transformer is specified as having 0.999 coupling coefficients between all three windings of the transformer, see the transformer component sheet for details regarding this definition. These are the signals that are driving the ports, we can now inspect the voltages that are appearing at the port using `Vout1` and `Vout3` (`Vout2` is not shown as it is the same as `Vout1`). This is shown in next figure.

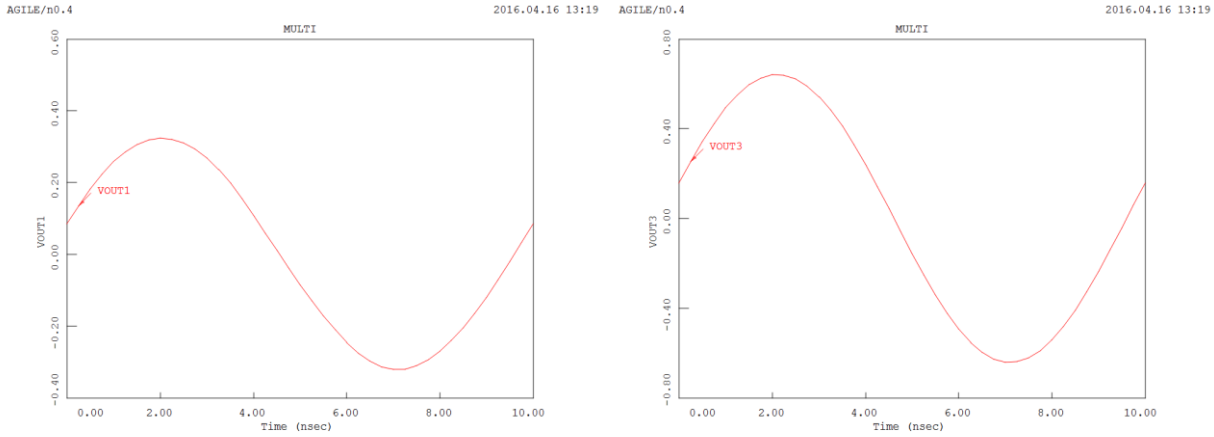


Figure 6.11 Result on plotting `Vout1` and `Vout3` in circuit `multi`

Notice there is great difference between these voltages and the applied generator voltages. `Vsig1` is the signal applied to port 1, but `Vout1` is the signal that appears at the port interface, that is after the 50 ohm port load.

This example effectively illustrates that multiple signals may be applied to any network and emphasizes that the signal that drives the port is not directly connected at the port interface but operator behind the port termination (although one can use a 0 ohm resistor as the port load to directly drive the port). These examples did not show mixtures of voltage and current signal application although this too is perfectly acceptable.

Throughout these two examples, we emphasized the fact that the application signals are periodic. Approximations to non-periodic behavior can be achieved by making the period very long as compared to the "settling" time of the circuit near the points of interest. In many cases such approximations can produce practical results. However,

transient analysis tools must be employed when such approximations are not practical. The discussion continues with time domain optimization.

6.6 OPTIMIZATION IN THE TIME DOMAIN

This section describes how to define optimization goals using time domain signals. As discussed in prior sections, the frequency domain signal based outputs can be used just as the "regular" output variables are in goal statements. Time domain signal outputs, however, can also be used in goal expressions. Some correct examples are:

```
Goal1 SUMNORM Vsignal1, time*1e-9+1
Goal2 MAXIMUM Isignal23., 2
Goal3 UPPERLOWER Vsignal2, (time*1e-9)*2-4, (time*1e-9)*2-3
```

In `Goal1` above, the goal error is the sum *across the period in the time domain* of the absolute value of the difference between `Vsignal1` and the time domain expression `time*1e-9+1`. Remember from chapter 5, the various goal functions (`SUM, MAXIMUM, UPPERLOWER`, etc.), where the functions operated over the frequency domain. `Goal2` represents the maximum deviation across the period between the current at port 23 and 2 (2 in this case is still a time domain signal, its d.c. 2 amps). `Goal3` shows the method used to force the voltage at port 2 to be between two time domain expressions.

An example will illustrate time domain goal usage.

6.6.1 EXAMPLE 3 - OPTIMIZING TO ACHIEVE A TIME RESPONSE

The optimization example presented in this section will not only show time domain optimization goals, but also demonstrate that *both time and frequency domain goals may be used simultaneously*. The example will be a simple microstrip transmission line, we will optimize its width and length such that 1) a minimum reflection is obtained (frequency domain), and 2) a time domain transfer function is achieved. More specifically, the time domain function to be achieved will be a 200 psec delay. The network listing is:

```
Network : FandTgoal
SIGNALS:
    Cutoff = 2              ;* shut off at 2 nsec
    SIGNAL pulse voltage time 2*unit(Cutoff-time*1e9)  ;* pulse
    0 5 0.25 ghz            ;* includes DC
PORTS:
    #1 1 0 pulse
    #2 2 0
COMPONENTS:
     width =  15 [  5 : 50 ]
     length= 800 [100 : 1500 ]
     T1 = MSTRL(ht = 20 mils, er = 10, PL=length mils, W=width mils)
CONNECTIONS:
    T1 1 2 0
OUTPUTS:
    InSpectrum = mag(Vspectrum1)
     InputVolt = Vsignal1
    OutputVolt = Vsignal2
GOALS:
```

```
    Delay = 0.2                     ;* want a 200 psec (0.2 nsec) delay
    GoalinFreq SUMNORM mag(RC1) : 0
    time_ns = time*1e9
    GoalinTime SUMNORM Vsignal2 : &
            if( time_ns > delay AND time_ns <= delay+cutoff, 1, 0 )
    Fmerit = GoalinFreq + GoalinTime
END
```

In the network `FandTgoal`, a transmission line circuit is established. As goals, the magnitude of the input (port 1) reflection coefficient is used in a frequency domain goal statement: `GoalinFreq`. `GoalinFreq` is plotted in next figure, its goal error is 0.0737903. In this goal, the goal function SUMNORM was used, therefore the goal error represents the *average* deviation of the reflection coefficient from zero.

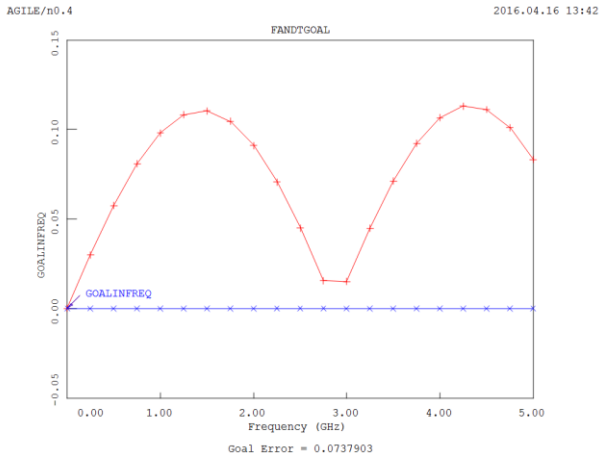


Figure 6.12 Plot of `GoalInFreq` in circuit `FandTgoal`

The time domain goal variable `GoalinTime` is shown plotted in Figure 6.13. Its goal error is 0.036853. Notice that the input waveform is doubled because a perfect input match results in a waveform of ½ of the applied signal at the generator. Remember that the voltage signal generator sits behind 50 ohms. Since we desire the transmission to be lossless (which it isn't), the output waveform should have the magnitude of the input (1 volt) not that of the signal generator (2 volts).

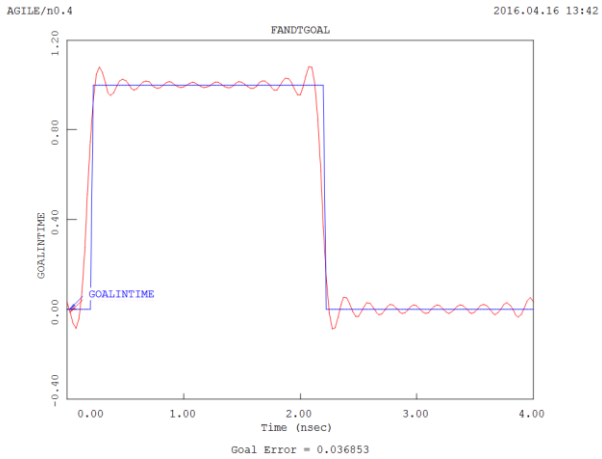


Figure 6.13 Result of plot `GoalInTime` in `FandTgoal` circuit

FMERIT is the sum of these two goal errors which is 0.110643. Since we anticipate that both GoalinFreq and GoalinTime can be significantly reduced, these two goals are not poorly scaled in relation to one another. Therefore, we are ready for optimization.

After use of the simplex optimizer (although any could have been used), the values:

```
WIDTH = 19.1184 [ 5 : 50 ]
LENGTH = 967.635 [100 : 1500 ]
```

were found.

Figure 6.14 illustrates the results for GoalinFreq and GoalInTime after the optimization. The figure of merit was reduced from 0.110643 to 0.0327818. Notice the significant improvement in reflection coefficient. While a bit more difficult to notice, the signal delay as shown in the time domain is better as well. The output is better centered with the goal line.

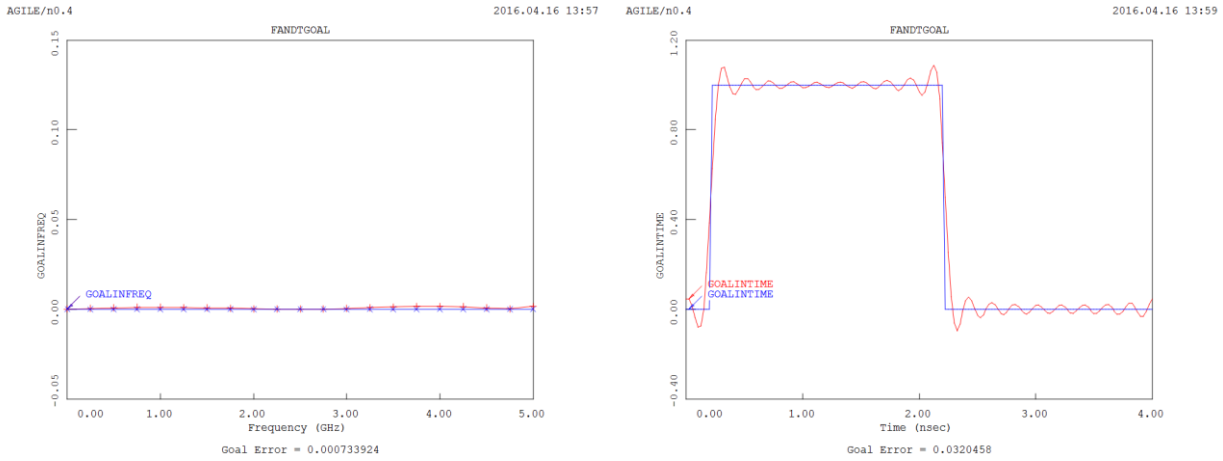


Figure 6.14 GoalInFreq and GoalInTime after optimization

This example demonstrates that goals may be defined simultaneously both time and frequency domains (of course, it is not necessary to use both domains). Use of time domain variables in goal statements is acceptable and necessary in some cases.

However, was this the best way to get this particular result? The group delay output (GDELAY) also reflects the delay of the network (in addition to the time domain approach). So let's try this optimization using a different approach, while still achieving the same idea. Figure 6.15 shows a plot of Gdel = Gdelay12*1e12 (after optimization) which is the group delay in picoseconds for the FandTgoal circuit.

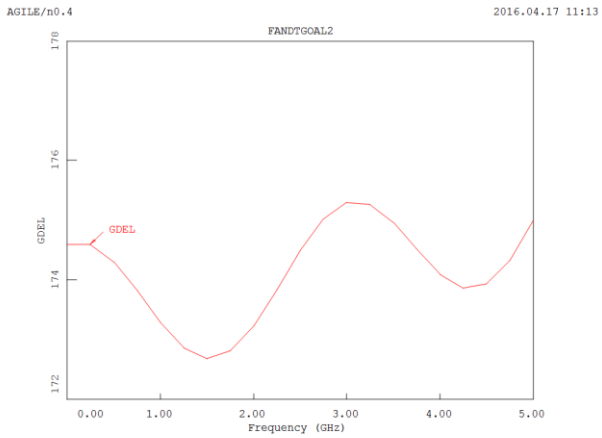


Figure 6.15 Group delay in psecs after optimization of FandTGoal

This shows about 200 psec of delay, averaging around 175. Instead, let's create a new network called FandTgoal2 that instead optimizes the group delay directly in order to get a 200 psec delay. This network is listed here:

```
Network : FandTgoal2
SIGNALS:
    Cutoff = 2               ;* shut off at 2 nsec
    * SIGNAL pulse voltage time 2*unit(Cutoff-time*1e9)  ;* pulse
    0 5 0.25 ghz
PORTS:
    #1 1 0        ;* pulse not needed in this version
    #2 2 0
COMPONENTS:
     width =  15 [  5 : 50 ]
     length= 800 [100 : 1500 ]
     T1 = MSTRL(ht = 20 mils, er = 10, PL=length mils, W=width mils)
CONNECTIONS:
    T1 1 2 0
OUTPUTS:
    InSpectrum = mag(Vspectrum1)
     InputVolt = Vsignal1
    OutputVolt = Vsignal2
GOALS:
    GoalinFreq SUMNORM 1000 * mag(RC1) : 0
    *Delay = 0.2                      ;* want a 200 psec (0.2 nsec) delay
    *time_ns = time*1e9
    *GoalinTime SUMNORM Vsignal2 : &
    *        if( time_ns > delay AND time_ns <= delay+cutoff, 1, 0 )

    Gdel = Gdelay12 * 1e12
    GoalinFreq2 SUMNORM Gdel : 200     ;* want 200 psec delay
    Fmerit = GoalinFreq + GoalinFreq2
END
```

Its mostly the same as FandTgoal, but with items not necessary for the time domain optimization goals commented out and new elements for group delay goals added. Since the group delay goal is looking at differences scaled in the 100's and the reflection coefficient goal is scaled more in the less than one range, the GoalInFreq was scled up by 1000 here. This is the goals and Fmerit before optimization of FandTgoal2:

```
********** Goals **********

GOALINFREQ = 73.7903
GOALINFREQ2 = 25.8724

******* Non-Frequency Dependent Outputs *******

FMERIT = 99.6627
```

If we now optimize this circuit we will get:

```
Ending OPTIMIZER...after 136 ITERs.
Reason for TERMINATION: Flat SIMPLEX
Starting Error = 99.6627
Lowest Error = 1.37574

Variable (LENGTH) at value: 910.833
Variable (WIDTH) at value: 19.1124
```

Which ends with similar variate values but not exactly the same as before. The following figure shows `GoalInFreq2` which is the 200 psec (constant) group delay target and the actual result which is very tight with the 200 psec goal.

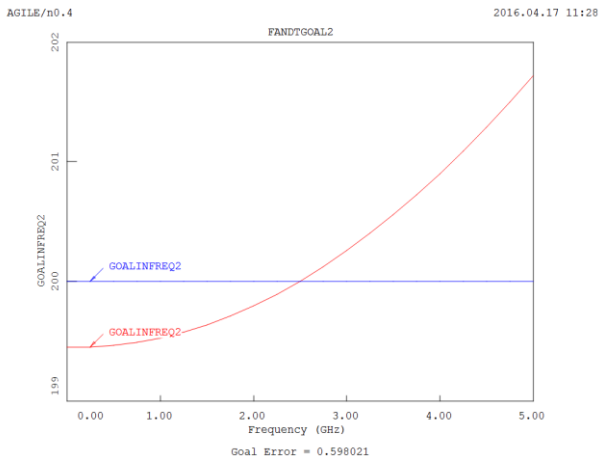


Figure 6.16 `GoalInFreq2` for `FandTgoal2` after optimization

Therefore, in the case here we could have done a time domain or frequency domain optimization process. While in this example it appears that the frequency domain case is better, there are many cases where time domain optimization is useful and practically required, specifically in cases where is very difficult to express the goal(s) in the frequency domain. The example presented in this section, did in fact have a simple frequency domain goal equivalent to the time domain goal used, but this is not always the case.

6.7 DISCUSSION

In this chapter we have reviewed some basic concepts relating to frequency and time domain transformations. We have also introduced AGILE NDL and commands which enable convenient use of time domain features. This included the definition and use of optimization goals which are in the time domain. Some examples were shown to illustrate the many concepts introduced.

7 STATISTICAL FEATURES

In this chapter, we will describe features of AGILE which relate to the statistical definition and analysis of circuits/systems. A very flexible method for describing the statistical properties of a circuit is defined in the first section of this chapter, *7.1 STATISTICAL VARIATE DEFINITION IN NDL*. The section describes how to define a variable which represents a *distribution* rather than a fixed quantity, as well as other things. Section 7.2, *DISTRIBUTION FUNCTIONS*, gives exact definitions for the distribution functions allowed in NDL. Following this, section 7.3, *SENSITIVITY ANALYSIS*, describes how to perform a sensitivity analysis of a circuit and thus begins the description of the `TOLERANCE` command group. The fourth section, *7.4 PASS STATEMENT AND THE COST STATEMENT*, defines the final NDL statements. The PASS statement define acceptable performance such that the yield can be predicted, while the COST statement defines circuit cost. The fifth section of this chapter describes tolerance analysis, it is appropriately entitled *7.5 TOLERANCE ANALYSIS*. Section *7.6 TOLERANCE DESIGN YIELD AND COST OPTIMIZATION* concludes the definition of the `TOLERANCE` command group. Complete NDL examples are used throughout sections 7.3, 7.5 and 7.6 to illustrate the concepts introduced. Some of the newer features in AGILE are meta-modeling along with the design of experiments in support of such.

7.1 STATISTICAL VARIATE DEFINITION IN NDL

In order to statistically analyze circuits, the circuit must be defined in statistical terms. In AGILE, this is done by defining variables as *statistical variates*, or just *variates*. These variates can take on values from a *distribution* of values, rather than be fixed to a specific value. We will use the term *variate* to mean a variable which has a *non-deterministic* value; conversely, all of the variables shown so far are *deterministic*, meaning that they have no statistical variation.

The general syntax defines a "simple" variate variable is:

```
var = real_number { distribution expr }
   where:
   var            - is the variate name being defined
   real_number    - is a real number (not an expression) this number
                    always represents the mean of the distribution
   distribution   - is one of: UNIFORM or PERCENT or NORMAL or LOGNORMAL
   expr           - is any real-valued expression, meaning is dependent upon the distribution used
```

There is another more complicated form (which combines range variables and variates) for variate definition but this will be discussed later.

An example of this form is:

```
variate_var = 0 {NORMAL 1}
```

defines a variate, which happens to be called `variate_var`, with a `NORMAL` distribution having a *mean* of 0 and an expression part which evaluates to 1. Since, for the `NORMAL` distribution, this expression value represents the *standard deviation*, `variate_var` will have a standard deviation of 1. We will completely define what the distributions and argument meaning later, for now let's concentrate on NDL syntax.

As mentioned, the mean must be a real number and cannot be an expression. Therefore:

```
illegal = x+y { UNIFORM 5 }   ;* is illegal and will generate an error
```

However, the parts of the definition contained in the curly braces can be expressions. An error will also be generated if the distribution name is not one of those listed (`UNIFORM`, `PERCENT`, `NORMAL`, and `LOGNORMAL`). Legal variate definitions are:

```
xvar = 5
sdev = 0 {NORMAL 1.2}
var1 = 10 {UNIFORM xvar+1}
var2 = 20 {NORMAL sdev+sqrt(xvar)}
var3 = -9e-4 {LOGNORMAL (xvar-1)*1e-5/4}
```

the variable `xvar` is a simple variable (with a value of 9), the variate `sdev` has a mean of zero and a standard deviation of 1.2. The variate `var1` has a mean of 10 and a `UNIFORM` expression value `xvar+1`, which will evaluate to 6. The variate `var2` has a mean of 20 and a `NORMAL` standard deviation involving `sdev`, which is itself a variate. This perfectly legal in AGILE -- meaning that not only is `var2` a distribution but the *standard deviation* of `var2` is a distribution. The variate `var3` has a mean of -9e-4 and a `LOGNORMAL` distribution with the expression part evaluating to 1e-5.

As mentioned earlier, the range variable syntax (section 2.7) can be combined with variate definition. The following variables illustrate the various types:

```
simple_var  = 1
range_var   = 2 [1:4]
rangeWstep  = 3 [1:4:1]
variate_var = 4 {NORMAL 0.1}
RandV1      = 5 [1:10] {NORMAL 0.2}
RandV2      = 6 [1:10:1] {normal 0.1} ;* NDL is Case Insensitive
```

Only the last two of these have not been previously introduced. The variable `RandV1` is both a range variable and a variate. It has a current value of 5, this is also its mean. During optimization it can take on values from 1 to 10 and during statistical analysis, its distribution is `NORMAL` with a standard deviation of 0.2. The variable `RandV2` is a range with step type variable as well as a variate. It has a mean and current value of 6, under optimization it ranges from 1 to 10, can be parametrically plotted over that range in steps of 1, and has a `NORMAL` distribution with standard deviation of 0.1.

The variable must be defined using the syntax shown above, for example:

```
Wrong = 6 {NORMAL 1} [1:10]  -- WRONG - range part must be first.
```

will generate an error. It is also illegal to make the expression in the distribution part either frequency dependent, time dependent, network analysis dependent, or complex valued. For example, the following variate definitions are all illegal:

```
value = (9,7)                  ;*-- this is OK, its a complex value
variate1 = 5 {NORMAL value+1}  ;*-- WRONG - expr is complex-valued
variate2 = 5 {NORMAL freq/1e9} ;*-- WRONG - expr is freq-dependent
o11 = mag(s11)                 ;*-- this is OK
variate3 = 5 {NORMAL o11*2}    ;*-- WRONG - expr is net. dependent
```

You get the idea. Now that the syntax is defined, it is time to describe the mathematical meaning of the distributions and the effect of the expression value for each distribution type.

7.2 DISTRIBUTION FUNCTIONS

We already stated that there are four distribution functions in AGILE, namely: `UNIFORM`, `PERCENT`, `NORMAL`, and `LOGNORMAL`. For all distribution functions, the current value is the mean.

The `UNIFORM` distribution provides a "level" or "even" distribution. The evaluation of the expression part determines the entire span or *width* of the distribution, which, of course, is centered about the mean. For example, if the NDL line:

```
VAR1 = 10 {UNIFORM 2}
```

were used, the variate `VAR1` would be centered at 10 and its value (when sampled) could range from 9 to 11 (a width of 2, as specified). The distribution is uniform, meaning that the samples will be evenly distributed within the width.

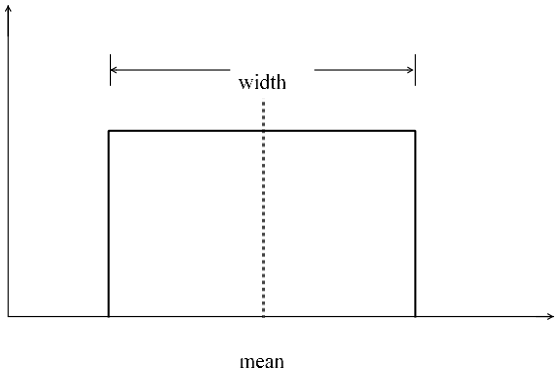


Figure 7.1 Illustration of the UNIFORM distribution.

A graphic illustration of the UNIFORM distribution is shown in Figure 7.1. In this figure, `width` is the parameter of the distribution.

The `PERCENT` distribution is also a uniform, level, or even distribution; however, its width is based upon its current value as well as the parameter. Note that the width of the `UNIFORM` distribution is based only upon the evaluation of its parameter. The parameter of the `PERCENT` distribution defines the distribution percentage over the width. For example:

```
VAR2 = -10 {PERCENT 5}
```

means that the variate `VAR2` has a mean of -10 and ranges ± 5%. In this case, that would mean from -10.5 to -9.5, with an equal probability within that range. The negative mean value was used to emphasize that the mean (for all distributions) need not be positive. This distribution might be used for low-frequency components whose values are nominally specified in terms of percent (*i.e.*, 47 ohm 10% resistor → `R1=47 {PERCENT 10}`). Its shape is the same as that of the `UNIFORM`, but its parameter has a different meaning; a graphic illustration of this distribution is shown in Figure 7.2.

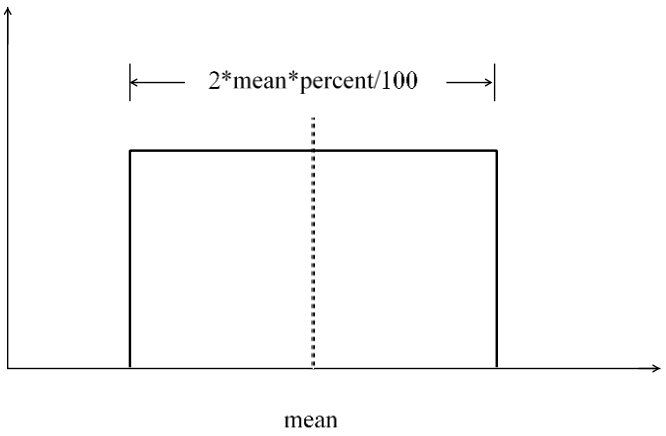


Figure 7.2 Illustration of the PERCENT distribution.

The `NORMAL` distribution (same as a Gaussian distribution) frequently occurs in engineering or other scientific problems. It is characterized by a mean and *standard deviation*. For example:

```
VAR3 = 100 {NORMAL 1}
```

makes the `VAR3` variate normally distributed with a mean of 100 and standard deviation of 1. This distribution is illustrated in Figure 7.3.

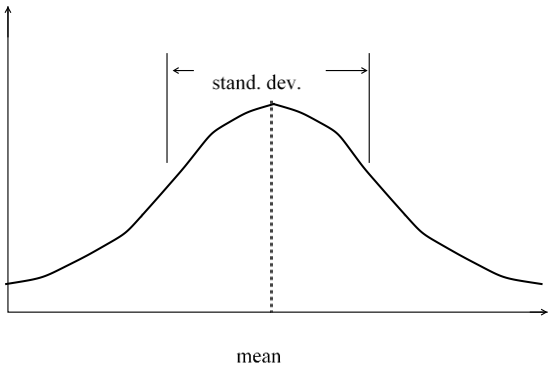


Figure 7.3 Illustration of the NORMAL distribution

The final distribution function in NDL is the `LOGNORMAL`. This also arises frequently in engineering problems, warranting its inclusion in AGILE. It is defined in a similar manner to that of the `NORMAL`, the distribution parameter defines its standard deviation. Of course, its shape is different. The name `lognormal` derives from the fact that the natural logarithm of this distribution is normal. Note that this distribution is "lop-sided", actually non-symmetric. The `LOGNORMAL` distribution is illustrated in Figure 7.4.

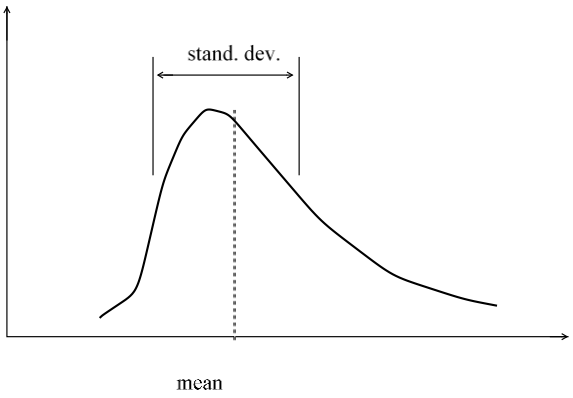


Figure 7.4 Illustration of the LOGNORMAL distribution.

This concludes the definition of the distribution functions in AGILE. With this information, and the material presented in the previous section, the definition of variates in AGILE should be well understood. The table below summaries the definition of the distributions in AGILE:

NAME	PARAMETER MEANING	If Parameter Value = 5
UNIFORM	total width	+ 2.5
PERCENT	percentage of width w/r/t mean	+ 5% of mean
NORMAL	standard deviation	std.dev. = 5
LOGNORMAL	standard deviation	std.dev. = 5

So far, we have described how to define such variates, but have not discussed how they are used. Such usage is actually postponed until two other related sections are presented but finally appears in *8.5 TOLERANCE ANALYSIS*. In the ensuing sections, actual AGILE displays of these distributions, and outputs of course, will be shown. The next topic is sensitivity analysis.

7.3 SENSITIVITY ANALYSIS

This section will be describing the first part of the AGILE tolerance tab, specifically, the `TOLERANCE SENSITIVITY` command. Basically, sensitivity analysis is the computation of *gradients*. In the context of circuit analysis and AGILE, it is the computation of the sensitivity of outputs, goals and the figure of merit, etc. with respect to changes in the values of variables which define component parameters or other values. Perhaps the best way to begin this discussion is with an example. Consider the NDL listing below for the network `tol1`

```
Network : tol1
 FREQUENCIES:
   10 20 1 ghz
  PORTS:
```

```
     #1 1 0; #2 2 0
  ENVIRONMENT:
    Ev1 : ht = 20 mils, er = 10
  COMPONENTS:
    halfwidth = 11 {NORMAL 0.05}
    T1 = MSTRL(ENV=ev1, PL=100 mils, W=2*HalfWidth mils)   CONNECTIONS:
    T1 1 2 0
  OUTPUTS:
   trans = dB(s21)
     Tangle = Aphase(PHASE(s21))
 GOALS:
    AverVSWR SUMNORM  VSWR1 : 0
    MaxVSWR  MAXIMUM  VSWR1 : 0
    Fmerit = AverVSWR
END
```

The only new part of this description is the variate definition for `halfwidth`, otherwise its the same old thing. The concepts of *targets* and *variates* is essential to utilization of all of the tolerance capabilities in AGILE. An example can illustrate these concepts -- the simplest command is to perform a sensitivity analysis. Using the Sensitivity analysis button on the Tolerance tab for this network will result in:

```
                              SENSITIVITY RESULTS
-------------------------------------------------------------------------------
                    NOMINAL| w/r/t:
    NAME              VALUE|       HALFWIDTH
FMERIT                1.045|      0.06449991
AVERVSWR              1.045|      0.06449991
MAXVSWR                 1.1|       0.0964748
```

Each of the names of the *targets* is listed down the left hand side -- in this case: `avervswr`, `maxvswr` and `fmerit`. The *variates* appear as the column headers -- in this case only: `halfwidth`. For each target, its *nominal* value is listed -- this is its value with all variates at their mean. This is the value you would get for these variables in a "regular" analysis. Under each variate column, the gradient (or slope) of each target *with respect to* (w/r/t) each variate is listed. In the case of tol1, the current value for `avervswr` is 1.0454 and its gradient (sensitivity) with respect to `halfwidth` is +0.0641563. This means for increases in the value of `halfwidth` (at the current point!) the value of `avervswr` increases. This increase occurs at the rate of +0.0641563. Of course this also means that decreases in `halfwidth` will bring decreases in `avervswr`. Note that the numbers for `fmerit` are identical to those for `avervswr`, this better be the case since they are defined to be the same thing!

One important question concerning this analysis is *"How were the targets and variates determined?"*. As with many AGILE features, they are automatically determined by default. Specifically, AGILE automatically uses any range variable or variable defined using a distribution as the variates and all goal functions and the figure of merit (`FMERIT`) as the targets.

A valid observation is that the actual distributions of the variates defined are not used in sensitivity analysis. We are only computing gradients, and this can be done with respect to variables which are not distributions (deterministic). In fact, as mentioned above, AGILE uses both variates and range variables as candidate subjects by default. For example, if the line:

```
Length = 100 [ 90 : 120 ]
```

were included in the description, the gradient of the goals and FMERIT with respect to Length would have been computed as well.

It is therefore true that sensitivity analysis could have been discussed in previous chapters. However, since variates are automatically used as gradient variables and the command to execute the sensitivity analysis is TOLERANCE, it was placed in this chapter. The distributions of variates will be used (finally) in section 8.5, but before this we must discuss another NDL statement related to tolerance studies.

7.4 PASS STATEMENT AND THE COST STATEMENT

The final two NDL statements described in this document are the *PASS statement* and the *COST statement*. The PASS statement is used to define acceptable circuit/system performance. In the true spirit of AGILE, this is done completely through an NDL statement and is therefore completely user defined. The syntax for the pass statement will be introduced through an example and then generalized. Consider the NDL line:

```
PASS averVSWR <= 1.05
```

The pass statement starts with the word PASS (case insensitive, of course) and is followed by any expressions. In this case, only a relational operator (which is a binary operator) was used, namely: averVSWR <= 1.05. The evaluation of the PASS statement should be defined such that it results in a "true" or "false" value. In the case above, the condition will evaluate to true if the value of averVSWR is less than or equal to 1.05.

However, multiple condition phrases can be contained in one pass expression. An example might be:

```
PASS averVSWR <= 1.05 AND maxVSWR <= 1.1
```

In the example just above, the pass statement is true if both conditions are true, that is if averVSWR is less than or equal to 1.05 AND maxVSWR is less than or equal to 1.1. Of course the ampersand character can be used to form multiple record NDL lines, so a long pass statement can be written as:

```
PASS  averVSWR <= 1.05 and  &
      maxVSWR <= 1.1  and  &
      averGain >  10
```

thus *and*ing the three condition phrases. Note that the pass statement expression cannot be frequency dependent, time dependent, or complex valued (of course appropriate error messages will result if you try any of these, we'll not bore you with the exact messages). The pass statement can be dependent upon goals and/or fmerit.

AGILE provides relational operators for numeric comparisons and logical operators AND (already shown) and OR? An example might be:

```
PASS  averVSWR <= 1.05 OR ( worstVSWR <= 1.2 AND avergain >=6 )
```

Before moving on to tolerance analysis, the final NDL statement must be described -- it is the *cost statement*. The cost statement defines cost, which will be optimized during tolerance design. Although this is discussed until section 8.6, we will complete the NDL definition by describing the cost statement here. For example:

```
COST = 100 + 100/yield
```

Cost is defined just as a simple assignment statement, it's just that the name COST is special. As you probably have guessed, the cost expression cannot be: frequency dependent, time dependent, or complex valued. It can only

be defined once, just as any other NDL variable. The cost expression can however, be a function of the special variable:

```
yield
```

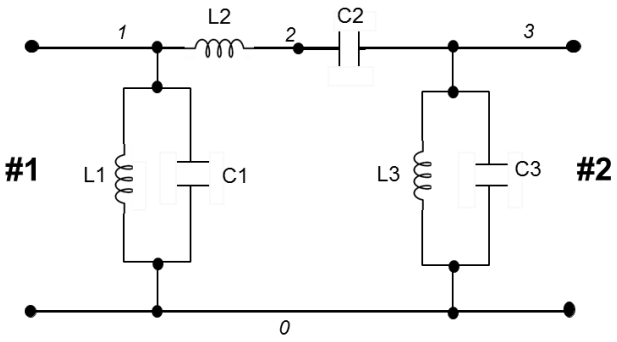
as was the case above. `Yield` is a special reserved variable which cannot be defined in an assignment statement -- its value is internally computed. We will further refine the concept of yield and utilize the cost statement in the next section.

7.5 TOLERANCE ANALYSIS

The entire purpose of pass statement is to enable the determination of yield. Obviously, in any single analysis, the circuit will either pass or fail – based on the evaluation of the PASS statement. However, in a statistical analysis, each variate is varied according to its distribution and *many* analyses are performed. During the course of these analyses, some will result in the pass state and some in the fail state --- the number of passes divided by the total number of analyses is the `yield`. Note that this is a number between zero and one. Since cost expressions involving `yield` may be degenerate when `yield=0` (i.e., `1/yield`), we do not allow `yield` to be less than 0.0001. For any *single* analysis, the circuit will either pass or fail (not pass) which results in a `yield`, respectively, of either of 1 or 0.0001 (100% or 0.01%).

During "Monte-Carlo" analysis, many analyses of the circuit are performed, with each analysis normally called a *sample*. At the beginning of each sample analysis, each variate defined in NDL is assigned a new value *from its distribution*. The definition of `tol2` below will be used as an example:

```
Network: Tol2
* Example of simple bandpass filter *
  FREQUENCIES:
    1  2 .025 ghz   ;* Bandpass Frequencies
*   0.5 3 .05 ghz   ;* Other frequencies for plotting when we're done
  PORTS:
    #1   1 0
    #2   3 0
  CONNECTIONS:
    L1 = 3.857 [ 1 : 10 : 1] {PERCENT 1}
    C1 = 3.284 [ 1 : 10 : 1] {PERCENT 1}
    L2 = 9.131 [ 5 : 15 : 1] {PERCENT 1}
    C2 = 1.387 [0.1:  5 : 1] {PERCENT 1}
    L3 = 3.857 [ 1 : 10 : 1] {PERCENT 1}
    C3 = 3.284 [ 1 : 10 : 1] {PERCENT 1}
    ind(l=L1 nh)   1 0
    cap(c=C1 pf)   1 0
    ind(l=L2 nh)   1 2
    cap(c=C2 pf)   2 3
    ind(l=L3 nh)   3 0
    cap(c=C3 pf)   3 0
  OUTPUTS:
    insertiongain = db(s21)
    inreflcoef = rtop(s11)
  GOALS:
    AverVSWR    SUMNORM          VSWR1 : 0.0
```



```
    AverLOSSdb  SUMNORM  real(PGAIN12) : 0.0
    WorstVSWR   MAXIMUM          VSWR1 : 0.0
    WorstLOSSdb MAXIMUM  real(PGAIN12) : 0.0
    PASS  AverVSWR <= 1.25 AND AverLOSSdb <= 0.1 AND &
         WorstVSWR <= 1.40 AND WorstLOSSdb <= 0.2
END
```

Figure 7.5 Tol2 network NDL and image

This defines a three section "chebychev-response" filter circuit. Note the definition of the variates and pass statement. Many of the components are defined to have 1% distributions. In this case, we are analyzing the circuit only over the pass band, and therefore only concerning ourselves with the pass band performance of this circuit. Of course the generality is there to develop NDL to account for both the pass band performance *and* skirt performance, but we're trying to keep this example short and focused.

The various goal statements determine VSWR and gain factors, we will not describe them, you already know what they are (refer to Chapter 6 if you need a refresher). We have based the pass condition entirely upon these goal functions. The acceptable condition is simply stated: the average VSWR must be below 1.25 and the maximum VSWR must be below 1.4 and the average loss must be less than 0.1 dB and the maximum loss must be less than 0.2 (actually, their all less than *or equal to*). This is across the pass band (1-2 GHz) as defined in the frequency statement.

Anyway, after defining the components as such, a "simple" analysis would appear as:

```
********** Frequency Dependent Outputs **********

F (GHz)| INREFLCOEF      | INSERTIO
  1.00 |  0.151   10.460 |  -0.100
  1.03 |  0.061    0.883 |  -0.016
  1.05 |  0.010  172.230 |  -0.000
<…omitted…>
  1.98 |  0.103   -5.502 |  -0.046
  2.00 |  0.151  -10.473 |  -0.100

********** Goals **********

AVERLOSSDB = 0.0503986
AVERVSWR = 1.21932
WORSTLOSSDB = 0.100145
WORSTVSWR = 1.35566
```

with some of the gory output omitted. We are finally going to get to do a tolerance analysis, by using the Tolerance analysis button with the number of samples set to 400. This will produce the output:

```
Number of Samples: 400

Computed YIELD: 61.25%
```

we will get AGILE to do 400 samples of the circuit, with each of the analyses taking on a different (random) value for each variate from its distribution. The *exact result* you get will depend upon your computer system, as each as a different random number generator. Indeed you will get slightly different results each time to execute the same

command. Issuing this on our system, the result is 61.25% for yield. This means that out of the 400 samples, 245 of them "passed", the remainder did not meet the conditions specified in the pass statement.

As previously stated in the tolerance sensitivity section, the samples for variates, goals, and `fmerit` are automatically saved as part of the analysis. This, too, is done for tolerance analysis.

An important point regarding number of samples: A good question is *"Why was 400 chosen as the number of samples?"*. The number of samples necessary in any particular case is dictated by *confidence tables*. Confidence tables dictate the number of samples necessary based on the expected (true) yield and the accuracy desired. The number of samples needed is not determined by the number of variates, nor their distribution functions. Confidence tables appear in virtually every statistics book, and some tables are produced in the Appendix A for your convenience.

Using confidence tables: The weak link in using confidence tables is that you may not know where the yield lies, especially for "new" problems. The approach here was to try multiple "runs" with a low number of samples (40) to get a "handle" on the yield prediction. From these, a rough estimate of 60% yield was garnered. The confidence table for 95% confidence limit and ± 5% error at an actual yield of 60% reads 368. Rounding this up a bit resulted in 400. Doing three tolerance analyses, each with 400 samples, resulted in yield predictions of: 61.25%, 59.5% and 63.5% all within ± 5% of some center point. Use of the 95% confidence limit tables is typical.

The result of tolerance analysis (as shown above) is not terribly exhilarating -- just a simple printout of yield. AGILE has commands to display histograms of the results. These are provided on the Tolerance tab – all of the usage of the program through the GUI is in Chapter 10. Here is the result of plotting `L1`:

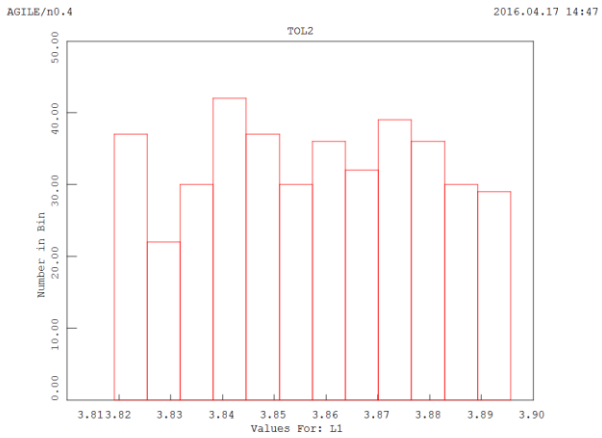


Figure 7.6 Result of plotting `L1` *for the network tol2, after 400 tolerance analysis samples are used.*

As can be seen in Figure 7.6, a histogram plot is produced with the x-axis as the variate in question (`L1`) and the y-axis is the number "in each bin". Of course, some explanation is needed. Binning was automatically done into 12 bins as specified on the Tolerance tab (12 is the default). Note that the center of this graph is approximately 3.857, as was specified as the mean. Remember that we specified that this component should be 1%, which means ± 1%. In this case, one percent of 3.857 is 0.03857, and note that the graph extends approximately ± this number from the center. Of course, another important observation is that the distribution isn't perfect. It is quite level, as it should

be, but it's not perfect -- this is the necessary result of using *random* numbers. Also, the graph scaling was adjusted afterwards to improve the graphic.

Finishing up the `tol2` example, let's look at the result of doing a histogram on the worst VSWR over the band as shown in Figure 7.7. In this figure, we used the HistogramSolid curve type rather than Histogram (empty bars as before) instead:

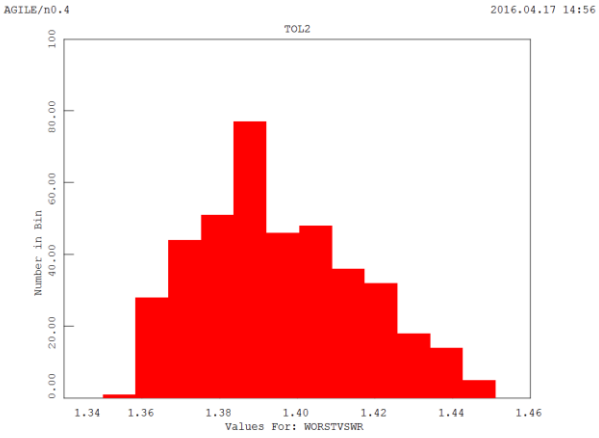


Figure 7.7 Histogram of WorstVSWR in Tol2 network

Note that the binning results are listed in the GUI as well[13], in this case:

```
Binned Data for (WORSTVSWR)
 Bin Low     Bin High     Bin Count
----------  ----------    ---------
      1.35       1.358            1
     1.358       1.367           28
     1.367       1.375           44
     1.375       1.384           51
     1.384       1.392           77
     1.392       1.401           46
     1.401       1.409           48
     1.409       1.417           36
     1.417       1.426           32
     1.426       1.434           18
     1.434       1.443           14
     1.443       1.451            5
```

The command `show tolerance` displays the "status" after a tolerance run. In the case above,

```
Tolerance MAX Number of Variates = 20
Tolerance MAX Number of Targets = 20
Tolerance Meta Mode = None
Tolerance Design of Experiment = Orthogonal Array
Tolerance Recycle Random Sequence = No
```

[13] You can control the number of bins, but you cannot control the binning range in the current GUI, it is always the span of numbers that are being binned.

```
Last TOLERANCE variate(s):
    C1
    C2
    C3
    L1
    L2
    L3
Last TOLERANCE target(s):
    AVERLOSSDB
    AVERVSWR
    WORSTLOSSDB
    WORSTVSWR
Computed YIELD: 61.25%
Number of Samples: 400
```

which shows the names of the variates, the names of the targets, the resultant yield computed and how many samples were taken.[14]

7.6 TOLERANCE DESIGN -- YIELD AND COST OPTIMIZATION

In this section we will discuss how to *optimize* a design including yield effects, this is called *tolerance design* in AGILE. In chapter 6 on optimization, we discussed how to alter variables such that goals are achieved and the figure of merit is improved. In the section above, we discussed how to analyze a circuit in a statistical sense, such that yield is predicted. Tolerance design combines these features, allowing optimization of the *statistical* analysis of the circuit, rather than simple optimization of the *deterministic* analysis. The approach taken is very general, and in being so, certain care must be taken to achieve meaningful, quality results.

The cost statement was discussed back in section 7.4, and as stated may involve expressions containing the special variable `yield`. While tolerance analysis is used to determine yield for a circuit that includes statistically varied elements, tolerance *design* wraps an optimization process around this to optimize statistical designs per the cost expression.

Its detailed explanation is better suited to the context of examples, so without further ado, let's plow into it.

7.6.1 COST OPTIMIZATION OF TOL3

In this first sub-section, a modification of the `tol2` network previously presented will be made to form `tol3`. This modification will really just involve adding some additional NDL, here it is in full:

```
Network: Tol3
* Example of a not so simple bandpass filter
  FREQUENCIES:
    1  2 .025 ghz   ;* Bandpass Frequencies
*   0.5 3 .05 ghz   ;* Other frequencies for plotting when we're done
  PORTS:
    #1   1 0
```

[14] The command line version of AGILE had many options for controlling (adding and deleting) tolerance variates and targets – this current GUI version does not provide these features yet.

```
    #2   3 0
  CONNECTIONS:
    L1 = 3.857 [ 1 : 10] {PERCENT 1}
    C1 = 3.284 [ 1 : 10] {PERCENT 1}
    L2 = 9.131 [ 5 : 15]
    C2 = 1.387 [0.1:  5] {PERCENT 1}
    L3 = 3.857 [ 1 : 10] {PERCENT 1}
    C3 = 3.284 [ 1 : 10]

    ind(l=L1 nh)   1 0
    cap(c=C1 pf)   1 0
    ind(l=L2 nh)   1 2
    cap(c=C2 pf)   2 3
    ind(l=L3 nh)   3 0
    cap(c=C3 pf)   3 0
  OUTPUTS:
    insertiongain = db(s21)
    inreflcoef = rtop(s11)
  GOALS:
    AverVSWR    SUMNORM          VSWR1 : 0.0
    AverLOSSdb  SUMNORM  real(PGAIN12) : 0.0
    WorstVSWR   MAXIMUM          VSWR1 : 0.0
    WorstLOSSdb MAXIMUM  real(PGAIN12) : 0.0
    PASS  AverVSWR <= 1.25 AND  AverLOSSdb <= 0.1  AND &
         WorstVSWR <= 1.40 AND WorstLOSSdb <= 0.2
    cost_per_pf = 5.0
    cost_per_nh = 5.0
    COST = (C1+C2+C3)*cost_per_pf +    &
           (L1+L2+L3)*cost_per_nh +    &
           20*(1/yield-yield)
END
```

Figure 7.8 The tol3 circuit

Everything in this network is the same as in `tol2` except that the COST statement (and the two support variables: `cost_per_pf` and `cost_per_nh`) have been added. Since defining a proper expression for cost is essential to tolerance design, let's investigate its expression in the `tol3` network.

Basically, the cost phrase involves three terms. The first term sums the values of the three capacitors in picofarads and multiples it by `cost_per_pf`. The intention of this term is to increase the cost based upon the total capacitance of the circuit. Since this term adds cost based upon the total capacitance of the circuit, we are stating that we would like the design to have a small value for total capacitance. The term for inductance is exactly equivalent. Of course, if only these two terms were used, the optimizer would just zip the inductance and capacitance values to their lowest points. However, the cost term that involves yield becomes quite large if the yield is small. Obtaining a good yield implies that the circuit is performing as necessary. Thus, in this example, we are trading off yield for use of lower component values. This is important in real designs in which simple yield optimization is not sufficient. Another important observation is that the terms `L1`, `C1`, `C2` and `L3` are variates, and therefore have a non-deterministic value. Currently, the "last" Monte-Carlo run value will be the values used in the computation of the cost expression.

Note that the optimization of yield alone is very easily accommodated by using expressions like: `1/yield` or `1-yield` for cost. Thus, yield optimization is a simple subset of the generality available using the cost statement. We chose the example above to illustrate the more general nature of *cost optimization* versus simple *yield optimization*. For example, yield optimization alone cannot incorporate effects such as component costs. You can see that the flexibility of the cost expression will allow virtually any circuit cost case to be handled.

Returning to our example, `tol3`, the first part of the tolerance design procedure is to get stable tolerance analysis results. We need to determine the correct number of samples to generate confident yield results. We started our process by doing a tolerance analysis with 400 samples that resulted in:

```
# Samples: 400 Computed YIELD: 82.75% Computed COST: $132.785
```

You'll note that at 400 samples, we are likely to incur less than about ± 4% variation in the yield estimate (from 80% yield row in the 95% confidence table). This is probably about the worst stability which is acceptable to tolerance designing (going to 1500 samples per tolerance analysis would help -- there would be ± 2% error), but this would take much longer. Anyway, starting a 400 samples per tolerance analysis design using the Simplex optimizer will result in (this takes over 10 minutes on a year 2015-level PC, its about 400 × 219 = 87,600 circuit analyses):

```
Ending OPTIMIZER...after 219 ITERs.
Reason for TERMINATION: Too Many ITERS without Improvement
        Starting Error = 132.785
        Lowest   Error = 129.139

    Variable (C1) at value: 3.3483
    Variable (C2) at value: 1.37106
    Variable (C3) at value: 3.2828
    Variable (L1) at value: 3.89667
    Variable (L2) at value: 9.13989
    Variable (L3) at value: 4.14106
```

Note that the final cost is $129.139, not much improvement over our initial cost of $132.785. This is because the optimal figure of merit point (deterministic) is the same as the optimal cost point. The next section contains an example in which this is not the case, and of course, this is not the case in general. Of course, any changes to the cost function will (most likely) change the optimal cost point.

Tip: The update checkbox determines if the variables will be updated after optimization automatically

Another very important suggestion in this example is the use of the simplex optimizer. Since the data to be optimized (cost) is *noisy*, gradient methods will not perform very well. This is due to the fact that they will get bad gradient information, which will lead them astray. Therefore, currently, only the simplex method is recommended for use in tolerance design, although we allow any of them in the command. But the simplex method should work best with noisy data.

7.6.2 COST OPTIMIZATION OF TOL4

We are going to get even fancier in this example than in the previous one. In the network `tol3`, we wanted to discover the means of six variables, with three of them variates, such that we optimized cost. In this example, we will discover *how bad the components can be while maintaining yield (performance).* This, too, is a cost

optimization problem which simple yield optimization cannot perform. The essence of this problem is to define a range variable to optimize, with this variable controlling the distribution of a component. We will "keep it simple", by only having one variate and one range variable, although this basic method is easily adaptable to any dimension. Here is a listing of the `tol4` circuit we will use in this section:

```
Network: Tol4
* Example of a not so simple bandpass filter
  FREQUENCIES:
    1  2 .05 ghz   ;* Bandpass Frequencies
*   0.5 3 .05 ghz  ;* Other frequencies for plotting when we're done
  PORTS:
    #1   1 0
    #2   3 0
  CONNECTIONS:
    L1 = 3.857
    C1 = 3.284
    Ltol  = 5.0   [ 0.2 : 20 ]
    l2    = 9.131 [ 5.0 : 15 ] {PERCENT Ltol}
    C2 = 1.387
    L3 = 3.857
    C3 = 3.284

    ind(l=L1 nh)  1 0
    cap(c=C1 pf)  1 0
    ind(l=L2 nh)  1 2
    cap(c=C2 pf)  2 3
    ind(l=L3 nh)  3 0
    cap(c=C3 pf)  3 0
  OUTPUTS:
    insertiongain = db(s21)
    inreflcoef = rtop(s11)
  GOALS:
    AverVSWR    SUMNORM          VSWR1 : 0.0
    AverLOSSdb  SUMNORM  real(PGAIN12) : 0.0
    WorstVSWR   MAXIMUM          VSWR1 : 0.0
    WorstLOSSdb MAXIMUM  real(PGAIN12) : 0.0

    PASS  AverVSWR <= 1.25 and  AverLOSSdb <= 0.1 and &
         WorstVSWR <= 1.40 and WorstLOSSdb <= 0.2

    COST = (10-Ltol/2) + if( yield >= 0.8, 0.0, EXP(8-10*yield)/2.7)
END
```

Figure 7.9 Listing of tol4

Here are the goals for this design. Suppose a 1% tolerance inductor costs $10 while a 20% inductor costs 5 cents, with cost values linear in between this range. Thus the basic cost relationship is approximately given by: *inductor cost* = `10 - Ltol/2`, where `Ltol` is the inductor value. As far as circuit performance goes, we wish to get better than about 80% yield, otherwise we will consider the design unacceptable. We need a means for expressing this requirement along with the cost associated with the inductor's tolerance.

A term of the form: `EXP(8-10*var)` will be a steep curve with value 1 at var = 0.8 and increasing rapidly at lower values of var, it looks like the left side of this figure:

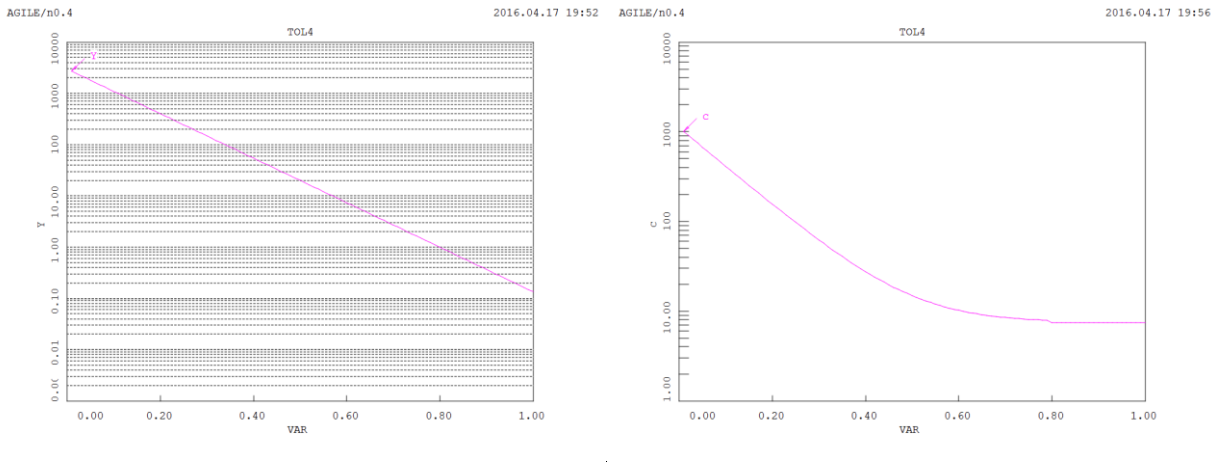


*Figure 7.10 Plot of exp(8-10*var) vs var [left] and*
*c = (10-Ltol/2) + if(yield >= 0.8, 0.0, EXP(8-10*yield)/2.7) vs var [right]*

To formulate the entire cost expression as a function of yield Figure 7.10 shows and evaluation of the expression

```
C = (10-Ltol/2) + if( var >= 0.8, 0.0, EXP(8-10*var)/2.7)
```

For `var` ranges from 0 to 1 and `Ltol` at nominal 5 – with the idea that in the real circuit `var` will be yield (and `Ltol` will be optimized). We can see that the cost c increases rapidly at values of `var` less than 0.8 – for our desired 80% yield minimum. So the expression C is actually the cost function in `tol4` – it captures the effect of making sure we get 80% yield while using the cheapest inductor we can.

After a tolerance design of this circuit we get:

```
Opt Vars:
---------
             L2 -> start value = 9.131
           LTOL -> start value = 5
…
Ending OPTIMIZER...after 129 ITERs.
Reason for TERMINATION: Too Many ITERS without Improvement
        Starting Error = 164.579
        Lowest   Error = 9.40457

    Variable (L2) at value: 14.8093
    Variable (LTOL) at value: 1.19087
```

Which says we should use a 1.2% 14.8 nH inductor for optimum cost.

Note that we optimized both the value of the inductor L2 and its quality in terms of percent value -- a very generalized problem. Note that at the end of a tolerance design, the histogram outputs as in tolerance analysis are valid -- the output being based upon the final tolerance analysis (which may NOT BE THE OPTIMAL).

The tolerance design result is entirely based upon the cost statement we gave. In review, we have basically said that we want to use the "sloppiest" inductor we can while still maintaining a yield of greater than about 80%. Of course, *all components* could have been optimizable and/or statistical variates, but we chose to keep this problem simple for demonstration purposes.

7.7 META-MODELING AND DOE (REMOVE CODE AND SECTION?)

In all cases previously, the optimization and/or tolerance analysis was carried out using actual simulation runs. The idea of meta-modeling is to build a quickly executing "model of the (simulation) model" (i.e. a meta-model) and perform the Monte-Carlo runs on this model rather than on the actual simulator (which is many times quite time consuming). In order to build this model, points of a design-of-experiments (DOE) are used in a fitting procedure to create the meta-model. AGILE (currently) supports two forms of DOEs: central-composite (CC) and orthogonal-arrays (OA) and two basic forms of meta-models: polynomial (linear, quadratic and cubic) and Sacks spatial correlation (SSC) model. AGILE is designed so that any DOE can be used with any of the meta-models (with varying results, though).

As stated, two forms of DOEs are implemented in AGILE (CC and OA). In both cases, the DOE is bounded around $\pm$ 1.5σ, that is the region of points (and hence the region that the fit occurs on) is $\pm$ 1.5σ in each variate. For the CC case, AGILE actually uses an augmented central composite design. The design generated by the method, for the number of variates *n*, requires an exponential number of points:

$$N^{CC}_{doe} = 1 + 2n + 2^{n+1} \quad \text{(Eq 8-1)}$$

The central composite design uses the center point, two points on axis the variate (at $\pm 2\sigma$) and "cube-points" which are those made by taking combinations of variates at $\pm\sigma$.

The points generated by OA DOEs are a bit more complicated to define. Orthogonal arrays of *strength* 2 are implemented in AGILE. The points of an orthogonal array of strength two will completely fill a grid in any two dimensional projection. That is if there are ten variates, if any two variates are chosen, points projected onto the plane of these two variates will completely fill a grid (of size *p*). Although in general OAs of strength 2 grow as n^2, AGILE's implementation adds points (for better fitting and for cubic models) and grow as n^3. Orthogonal arrays are thus suited to much larger problems (over central composite or fractional factorial types). The equation below gives the number of points for OAs of strength 2 as implemented in AGILE:

$$N_{doe}^{OA} = p^2 > 1 + 3n + \frac{3}{}n(n_2 - 15) + \frac{}{6}n(n-1)(n-2)$$

(Eq 8-2)

where p is prime

---

15. Note: the time to fit the Sacks spatial correlation (SSC) model, t_f, is fairly significant. Of course as the electrical circuit grows, it becomes less and less significant.

That is that the number of points for the OA DOE is p^2 for a prime number p such that p^2 is greater than the expression given. As an example, for n=10, the right-hand expression evaluates to 259 and the smallest prime p is 17 such that $17^2 > 259$ -- therefore 289 (17^2) points are generated. In contrast, the CC method would require 2069 points for n=10.

In order to determine the applicability of meta-modeling to any particular problem, the following expressions must be considered:

$$T_{normal} = N_{mc} t_s \qquad \text{(Eq 8-3)}$$

$$T_{meta-model} = N_{doe} t_s + t_f + N_{mc} t_m \qquad \text{(Eq 8-4)}$$

where N_{doe} is either N^{OA}_{doe} or N^{CC}_{doe} depending on which design-of-experiments is chosen, t_s is the time for a single (deterministic) simulation run, t_m is the time to evaluate the meta-model, t_f is the time to fit the meta-model, and N_{mc} is the number of Monte-Carlo runs needed (as determined by confidence tables in Appendix A.9). T_{normal} is the total execution time in the case of not using meta-modeling, as can be seen it's just the number of Monte-Carlo runs times the time for each simulation. $T_{meta-model}$ is the time needed for a meta-modeling based Monte-Carlo analysis. In many cases, $N_{doe}t_s$ is much, much greater than t_f or the $N_{mc}t_m$ product[1] -- therefore the main comparison becomes N_{doe} to N_{mc}. Obviously, whenever N_{doe} is greater than N_{mc}, it makes no sense to use meta-modeling. Since the meta-model is only an approximation to the actual simulation results, the other consideration is the amount of error associated with the metamodel. Of course, this is highly dependent on the number of variates, the choice of the design-ofexperiments, the meta-model and the characteristics of the outputs being modeled. A rule of thumb might be to use meta-modeling only when $N_{doe} < N_{mc}/2$.

Let's consider an example, namely the `tol3` example from section 8.6.1. Since its a derivative of the `tol2` network of section 8.5 and has about 60% yield, for the 95% confidence limit and error of $\pm$ 5% results in about 400 Monte-Carlo samples (N_{mc} above). This problem contains 4 variates (L1, C1, C2, L3) so that the number of DOE samples need would be N_{doe} = 41 for the Central-Composite DOE (Eq 8-1) or N_{doe} = 49 for the OA DOE (Eq 8-2 for p=7). Since both of these are much lower than 400, this is a problem where meta-modeling is appropriate.

Next comes the commands necessary to effectuate meta-modeling. Part of the SET commands control selection of the DOE and meta-model, a synopsis is:

SET TOLDOE [CENT_COMPOSITE | OARRAY]

SET TOLMETA [NONE | LINEAR | QUADRATIC | CUBIC | SACKS]

SET TOLRECYCLE [NO | YES] -- reserved for future use; N/A now

The `SHOW TOLERANCE` (as shown in section 8.5) displays the settings of these. At start up, the settings are: `TOLDOE=OARRAY` and `TOLMETA=NONE`. If meta-modeling is off (whenever `TOLMETA=NONE`), then the setting of `TOLDOE` is irrelevant.

Just as the number of variates are automatically determined, the tolerance command also automatically determines the "outputs" (called targets) as discussed in section 8.5. Ideally, the meta-model would simply be for the `PASS` expression. But meta-modeling techniques are not available for binary values such as the `PASS` expression (i.e.,

each Monte-Carlo run is either "pass", with a value of 1, or "fail" with a value of 0). AGILE therefore automatically meta-models the *parts* of the **PASS** expression. For example, in the case of TOL3, the terms AVERVSWR, AVERLOSSDB, WORSTVSWR, and WORSTLOSSDB are used to compose the PASS expression and these then are the outputs which are metamodeled.

The polynomial models (LINEAR, QUADRATIC and CUBIC) fit polynomials of the designated order to the DOE runs (as selected using SET TOLDOE). Although thorough discussion is beyond the scope of this document, the Sack's spatial correlation (SSC) model basically exploits the expected smoothness of the outputs. While the number and rapidity of variations (i.e., the "shape" of the outputs) are restricted by the order in polynomial metamodels, the SSC method has no such restriction. The SSC model is of the form: $y(\vec{x}) = \gamma(\vec{x}) + Z(\vec{x})$

The γ term captures the nominal features of the output (y) and is often merely a constant or simple linear polynomial (as is the case in AGILE). The Z portion of the model is assumed to be a Gaussian stochastic process with a spatial correlation function such that its covariance conforms as:

$Cov(Z(\vec{x}), Z(\vec{w})) = \sigma^2 R(\vec{x}, \vec{w})$

where

$$Var(Z(\bar{x})) = \sigma^2$$

$$R(\vec{x}, \vec{w}) = \exp(-\sum \theta_j |x_j - v_j|^p)$$

The value of p is fixed at 2 and the θ_j are efficiently estimated using maximal likelihood.

Effectively, the SSC forms an output from statistical measures in the region and the "fixed" linear model.

In the future, AGILE will make use of previous simulations in addition (or in place of) those demanded by the DOEs and the SSC model is well suited towards inclusion of such added samples in areas of rapid fluctuation to create a better model. References [8-1] and [8-2] contain more detailed information in regard to meta-modeling as used in AGILE.

7.7.1 COST OPTIMIZATION OF TOL6 USING META-MODELING

As an example of meta-modeling, consider the following circuit which is very similar to the TOL4 network we had previously:

```
Network: tol6
* Example of a not so simple bandpass filter   FREQUENCIES:
    1  2 .05 ghz   ;* Bandpass Frequencies PORTS:
    #1   1 0
    #2   3 0   CONNECTIONS:
    L1 = 3.857  {normal 0.05}
    Ltol  = 5.0   [ 0.2 : 20 ]
    L2    = 9.131 [ 5.0 : 15 ] {PERCENT Ltol}
    L3 = 3.857  {normal 0.05}
    C1 = 3.284  ;* fixed
    C2 = 1.387  ;* fixed     C3 = 3.284  ;* fixed
```

```
    ind(l=L1 nh)  1 0     cap(c=C1 pf)  1 0     ind(l=L2 nh)  1 2     cap(c=C2 pf)  2 3
ind(l=L3 nh)  3 0     cap(c=C3 pf)  3 0
  OUTPUTS:     insertiongain = db(s21)     inreflcoef = rtop(s11)   GOALS:
    AverVSWR    SUMNORM          VSWR1 : 0.0
    AverLOSSdb  SUMNORM  real(PGAIN12) : 0.0
    WorstVSWR   MAXIMUM          VSWR1 : 0.0
    WorstLOSSdb MAXIMUM  real(PGAIN12) : 0.0
    PASS  AverVSWR <= 1.40 and  AverLOSSdb <= 0.2 and &
         WorstVSWR <= 1.60 and WorstLOSSdb <= 0.4
    COST = (10-Ltol/2) + if( yield >= 0.8, 0.0, EXP(8-10*yield)/2.7) END
```

This circuit is the same as TOL4 except that the two other inductors, `L1` and `L3`, are non-deterministic as well as `L2` and the "specifications" embodied in the `PASS` statement have been relaxed. A non-meta-model tolerance analysis using 864 samples (determined from the confidence tables in Appendix A.9) shows that the nominal yield and cost are 87.7% and $7.50, respectively. The commands:

AGILE Task ? set toldoe cent_composite

AGILE Task ? set tolmeta quad

AGILE Task ? tol design 864 simplex

set AGILE to use a quadratic meta-model and a central composite DOE and performs a tolerance design. Note that the number of DOE runs needed (per Eq 8-1 for n=3) is 23. The time to fit the model and evaluate it (t_f and t_m in Eq 8-4) is indeed very fast compared to model evaluation, thus the meta-modeled version of this tolerance design runs about 35 times faster (=864/23) when compared to the non meta-modeled version. The final solution and results values found are `LTOL=6.67` and `L2=10.39` and the final`COST` is $6.62. Next a quadratic meta-model using an orthogonal array DOE is tried:

```
AGILE Task ? set toldoe oarray
AGILE Task ? set tolmeta quad    --redundant since already done
AGILE Task ? tol design 864 simplex
```

Per Eq 8-2, the number of runs needed to fit the meta-model in this case (n=3) is 25 (p=5; $p^2 > 20$) so this too runs about 35 times faster (864/25) than the non meta-modeled version. The results are then `LTOL=6.03` and `L2=9.205` and the final`COST` is $6.98.

7.8 SUMMARY AND DISCUSSION

In this chapter, we introduced the final NDL statements. These statements describe how to define *variates*, how to define a passing condition for the circuit/system, and how to define circuit cost. The "three-levels" of statistical analysis were introduced: *tolerance sensitivity*, *tolerance analysis*, and *tolerance design*. Each of these analyses is more computationally intense than the previous. Sensitivity analysis computes the gradients of outputs, based upon range and variates being changed. Tolerance analysis computes yield and allows histograms to be displayed of the various variables involved in the analysis. Tolerance design allows *optimization* of yield, and more generally cost as a given expression of yield and other factors. The user has full control of these features using NDL, no programming is necessary to solve very generalized problems.

A general *design procedure* while using AGILE might be:

1. develop a good circuit model,
2. optimize for best figure of merit,
3. incorporate statistical variates into the circuit model and define cost,
4. do a tolerance analysis with only 5-40 samples to verify cost statement, and get a rough idea of the actual yield,
5. from this yield indication, use the confidence tables in the Appendix to determine an appropriate number of samples,
6. perform tolerance design -- if the number of variates are within acceptable bounds (see section 8.7), use meta-modeling first with a much higher number samples than indicated in the confidence tables (perhaps 10x of + 1%) ; shut metamodeling off and redo the problem from this solution -- otherwise just use the nonmeta-model tolerance design, and
7. redo multiple tolerance analysis at newly discovered optimal cost point (without meta-modeling) to check yield prediction validity.

For a non-statistical problem, only steps 1 and 2 are used. This procedure uses the results of deterministic optimization as a basis for beginning statistical design. This is as good a place as any to start tolerance design. Of course optimization (of either fixed or statistical circuits) is not guaranteed to find the absolute best (global optimum), only local minima.

In order to develop confidence in results obtained from the statistical features described, you should perform multiple tolerance analyses to check validity. Another important consideration is proper definition of the variates. In any simulation, proper modeling is essential. If poor or invalid models are used, then the results obtained are garbage. The same holds true for statistical modeling, the (statistical) simulation is valid for the models and distributions specified and nothing more. In fact, obtaining a good statistical model for a circuit is, of course, harder than obtaining a deterministic one. The examples presented in this section are quite idealistic, but the statistical circuit definition features and ensuing analysis types of AGILE are quite powerful and sufficient to handle many problems.

References:

[8-1] Peter J. Moosbrugger, David L. Rhodes, “Case Study Results for Simulation of a Microwave Filter”, *Proc. of 1993 Int Conf.on Simulation in Engineering Education*, January 1993, Vol. 25, No. 3, pp. 12-16, publ. Society for Computer Simulation (SCS), San Diego.

[8-2] J. Sacks, W. J. Welch, T. J. Mitchell, H. P. Wynn, “Design and Analysis of Computer Experiments”, *Statistical Science, 1989*, Vol. 4, pp. 409-435.

8 NOISE ANALYSIS

AGILE incorporates some of the most generalized and simple to use *linear circuit* noise analysis features. AGILE computes the *spot* (narrow band) noise outputs due to stationary, random processes which are of primary interest to microwave and analog circuit designers. It allows noise analysis of any circuit (nodal, as in the rest of the program), without the need for any special NDL statements. All components automatically generate thermal noise during noise analysis. Each sub-network may have its own temperature. The circuit/network may be multi-ported, with noise computations between any pair of ports. Port definitions of ports which do not have a common "ground" are allowed for noise computations as they are for other output computations.

In the generalized spirit of AGILE, noise outputs can be used arbitrarily in: NDL output expressions, goal statements used for optimization, and in PASS statements and thus be involved in yield computations for tolerance analysis. Also, the format for tabulated networks is augmented in this chapter to show how to include noise parameter data, both for two-port and multi-port circuits. Each sub-network can have its own temperature (with may be passed as a parameter), or a single global temperature may be used. The first section describes noise contribution from component models, next we describe the tabulated data formats, followed by the additional output variables associated with noise outputs, and an example.

8.1 TEMPERATURE AND NOISE OF COMPONENTS/SUB-MODELS

All internal components and sub-networks automatically generate thermal noise in accordance with their lossy part. For example, the coupled transmission line model will automatically generate the appropriate amount of thermal noise for both the "self-" and the "coupled-lines". As with all other components, the matrix components, `SMAT`, `YMAT` and `ZMAT`, also generate thermal noise. Ideal capacitors and inductors, of course, generate no noise contribution themselves since they are lossless. Normally, all resistive components (either `RES` or that specified as part of an `SRLC` or `PRLC` component) generate thermal noise. There is a special noise-less resistor (`NLRES`), which generates no thermal noise. One of the uses of this component is for developing special noise models, etc. Use of the `NLRES` component (versus any other resistor) does not affect the non-noise outputs in any way.

The component, `CNCGEN`, is a correlated noise current generator, as described in the component data sheets. This component does not affect the non-noise outputs in any way and is only used to generated (partially) correlated noise currents. This is quite useful for developing custom noise models for devices.

In the absence of the `TEMPERATURE` NDL statement, all networks and components are taken to be at the temperature specified on the settings tab. This is defaulted to be 290° Kelvin. However, each sub-network can be set at a different temperature using the temperature statement as in:

```
TEMPERATURE = 155   ;* this is mighty cold!
```

The temperature is always specified in degrees Kelvin, and the right-hand part of this statement may be an arbitrary, non frequency-dependent expression. This is illustrated in the following NDL fragment, showing that the temperature is dependent upon a passed parameter:

```
Network : fancy_fet( ..., sq_temp, ... )          ...
   Temperature = sqrt( sq_temp )          ...
END
```

Note that the only temperatures of importance are that of the source and the network itself (and its sub-networks). The temperature of the load network does not enter into noise output computations. Also, as with all other NDL statements, AGILE is not sensitive to the case of the statements (`Temperature` and `TEMPERATURE` mean the same thing).

The source network is always taken to be at the temperature specified on the settings tab, even if it has its own NDL temperature statement. This temperature should not be set to exactly zero (0.01 degrees Kelvin is OK), otherwise computational difficulties will arise.

8.2 TABULATED NDL STATEMENTS RELATED TO NOISE ANALYSIS

In addition to the data statements described in section 3.3, *Tabulated Networks*, there are three new NDL statements regarding noise analysis for tabulated networks. Two of the new statements relate to specification of the noise data for the tabulated network, while the third informs AGILE that the network generates no noise.

The `NOISE_DATA` statement specifies noise data using the "typical" noise parameter data associated with two-port (device) circuits. Its format is:

```
NOISE_DATA  freq(Mhz)   NFmin(dB)   |RCnopt|  <RCnopt   RNeqv(ohms/50)
```

specifies noise data for a particular frequency. The frequency point is always specified in Mega-Hertz (just as the tabulated data), then the minimum noise figure is specified in decibels, followed by the complex optimal reflection coefficient in polar form (magnitude and angle degrees) and the equivalent noise resistance, relative to the system impedance. Note that this is very similar to the data format used by other microwave frequency circuit simulators. Of course, this form can only be used two-port circuits. Some example lines of this format are:

```
NOISE_DATA  1000.0     0.712      .25     -160.1      0.38
NOISE_DATA  1200.0     0.822      .28     -162.4      0.41
```

which, for example at 1000 MHz, specifies the noise data as noise-figure minimum of 0.712 dB, optimum reflection coefficient of 0.25 at an angle of -160.1 and an equivalent noise resistance of (0.38 * 50) = 19 ohms (assuming we are working with a system impedance of 50 ohms).

Since the `NOISE_DATA` format only applies to two-port information, AGILE also has a generalized *N*-port format called `NOISE_MATRIX`. This can be used for networks with any number of ports (including 2 if desired). The format is:

```
NOISE_MATRIX  freq(Mhz)   complex-pair ...
```

specifies noise matrix terms (in polar form) for an N-port circuit. There must be N^2 complex terms to be specified, placed in row-major order. Since it is very unlikely that the user would create data in this format, we are not documenting it fully. However, this is the format that AGILE will save using the `SAVE OUTPUTS` command. We may change this in future versions, detecting when its a two-port and then outputting the other (`NOISE_DATA`) form. Also note that AGILE will only save this information if a noise output variable is present in the circuit, the next section describes noise output variables.

The third and final tabulated network NDL statement related to noise analysis is the:

```
NOISE_LESS
```

statement which specifies that the tabulated network is noise-free. Note that the NOISE_LESS statement only applies to tabulated networks, it has no effect on other network types. Otherwise, in the absence of either NOISE_DATA or NOISE_MATRIX statements, the network is assumed to be passive and its equivalent thermal noise is used.

Note that noise data, in either NOISE_DATA or NOISE_MATRIX form, cannot be extrapolated (against frequency), but may be interpolated. The method used is linear in the domain used for the data (i.e., since minimum noise figure is specified in decibels, it is interpolated in the decibels domain, not the linear domain). There are several self-explanatory error messages that may be generated when using tabulated noise parameters. We will not describe them here, but there is some additional description in the appendix which describes all error messages.

8.3 NOISE OUTPUTS

There are five outputs regarding noise analysis, available as NDL output variables. The table of output variables presented in section 2.13, *Output Variables*, is, as was stated in that section, incomplete. The appendix contains a list of all output variables, but Table 8.1 is a summary of the noise outputs, with some description.

Here are some details regarding computation of these outputs. If the circuit is multi-port, each port other than the two involved in the noise output computation are loaded with either the default system impedance or with the specified port termination (see section 3.4, *Non-Default Port Terminations*). If the system impedance is used, it is considered to be noisy and at the temperature specified on the settings page. If a port termination network is specified, it is considered potentially lossy and at the temperature specified in its NDL (or at the default). You can use the NLRES component in a port termination network if you want noiseless external ports.

NFIGij
is the actual noise figure in dB (real-valued) from port i to port j. This is defined as the signal-to-noise ratio at the output (port j) divided by the signal-to-noise ratio at the input (port i). As examples, NFIG12.15 computes the noise figure from port 12 to port 15 and NFIG12 computes the noise figure from port 1 to port 2.

NFIGMINij
is the minimum noise figure in dB (real-valued) from ports i to j.

YNOPTij
is the source admittance (at port i to port j) for optimal noise performance. This is complex-valued, in mhos (obviously 1/YNOPT12 is the optimal impedance).

RCNOPTij
is the reflection coefficient for optimal noise performance. This is complex-valued and is returned in rectangular form. Of course, you can use the rtop function to change forms.

RNEQVij
is the equivalent noise resistance in ohms (real-valued).

Figure 8.1 Noise related outputs

These outputs may be used exactly like the "normal" output variables when placed in NDL output statements (see section 2.14, *Output Statements*). It is useful to note that computation time will increase if noise outputs are present. They may be plotted, listed, etc. just like any other output expression. For example, the following NDL fragment shows one of the ways a goal statement can be used to minimize noise figure:

```
noise_goal MAXIMUM  nfig12 : 0.0
```

or a noise goal can be used in computation of yield:

```
PASS ... AND noise_goal <= 4.0 AND ...
```

Of course, such statements may be freely mixed, and used with the `FMERIT` statement, with output and goal statements of non-noise related outputs. Thus, both noise figure and gain, etc. may be optimized simultaneously.

8.4 EXAMPLE OF NOISE FEATURES

The NDL listing in Table 8.2., shows the listing for device data. Note that the "`NOISE_DATA`" format is used. Also note that the noise data frequency points are not the same as the tabulated S-parameters, this is perfectly acceptable. This noise data will be interpolated when used in the "main" network, as will be seen its description. The tabulated data is for an NEC-700 device, the data for which comes from the NEC data sheets

```
Tabulated : nec700
* linear S-parms for VDS=3V, IDS=30mA
 2000.0  .95  -26  3.57 157  .04 76  .66 -14
 4000.0  .89  -52  3.23 136  .06 62  .63 -26
 6000.0  .83  -71  2.98 121  .08 55  .59 -36
 8000.0  .78  -87  2.58 108  .09 47  .57 -42
10000.0  .73 -102  2.27  96  .10 42  .54 -49
12000.0  .70 -116  2.11  84  .10 39  .48 -56
14000.0  .69 -126  1.79  77  .11 40  .51 -70
16000.0  .67 -130  1.62  68  .12 36  .53 -75
18000.0  .65 -136  1.51  57  .11 40  .54 -77
* NOISE PARAMETERS:
NOISE_DATA   4000.0  0.7  .64   69   .38
NOISE_DATA   8000.0  1.2  .55  115   .20
NOISE_DATA  12000.0  2.0  .48  155   .20
NOISE_DATA  18000.0  2.7  .46  -33   .40
END
```

Figure 8.2 Listing of the nec700 tabulated network

Let’s put this tabulated circuit to use. The listing in shows `nexamp1` which uses `nec700` as a subnet.

```
Network : nexamp1
  4 8 .5 GHz
  #1 1 0
  #2 4 0
  Fcenter=15            ;* center freq (GHz)
  elec1  =   33.733 [ 10 : 170 ]
  TRL(Z= 100 ohms, el= elec1 deg, cf= Fcenter ghz) 1 100 0
  elec2  =   201.734 [ 10 : 350 ]
  TRL(Z= 70 ohms, el= elec2 deg, cf= Fcenter ghz) 1 101 0
```

```
  res(r=0.01 ohms) 101 0
  elec3  =   25.2441 [ 10 : 170 ]
  TRL(Z= 100 ohms, el= elec3 deg, cf= Fcenter ghz) 1 2 0
  :nec700 2 3 0   () { 1 2 0 }
  elec4  =    10.0    ;* fixed
  TRL(Z= 100 ohms, el= elec4 deg, cf= Fcenter ghz) 3 4 0
  elec5  =    143.065 [ 10 : 170 ]
  TRL(Z= 41.5 ohms, el= elec5 deg, cf= Fcenter ghz) 4 102 0
  res(r=0.01 ohms) 102 0
  power_gain = DB(S21)
  input_refl = DB(S11)
  input_s11  = S11
  noise_fig  = NFIG12
  nom_gain   = 10.0
  reflect_goal   SUMNORM    S11 : (0.0,0.0)
  gain_flat_goal UPPERLOWER db(s21) : nom_gain + .5 : nom_gain - .5
  good_noise_fig MAXIMUM    noise_fig : 0.0
  fmerit = reflect_goal + 2*gain_flat_goal + 2*good_noise_fig
END
```

Figure 8.3 The nexampl1 circuit

The main network consists of ideal electrical transmission lines and the device. This network is called "`nexamp1`" and is listed in Figure 8.3. Note that the file is set up for optimization, the variables are set to their values after an optimization. The circuit used is a very simple, single-ended amplifier. Note this topology is not terrific for obtaining great noise figure (it is good for getting flat gain, though), but was chosen for simplicity.

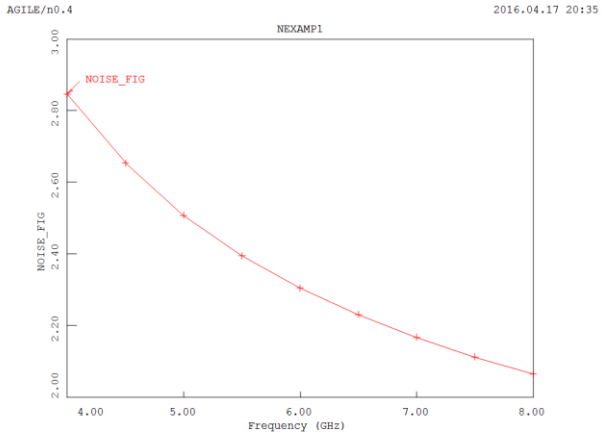


Figure 8.4 Plot of noise_fig from nexamp1 circuit

The optimization goals reflect the following desires for circuit performance: get a good input reflection match (difficult with this circuit), get flat gain near 10 dB and get the best noise figure possible. Note the "weighting" done in the figure of merit (`FMERIT`) expression. The plots in Figure 8.4 and Figure 8.5, show the results obtained from plotting the outputs `noise_fig` and `gain_flat_goal`.

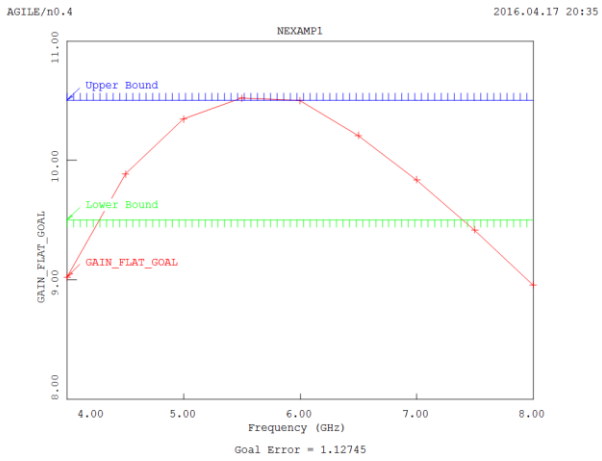


Figure 8.5 Plot of gain_flat_goal from the nexamp1 circuit

The result of applying optimization to this circuit is not shown.

8.5 SUMMARY

The features described in this chapter provide a complete, powerful and easy-to-use noise analysis capability. You can control temperature of each sub-network independently, with temperature possibly being a passed parameter. This allows the user to create temperature-dependent models, including the effect of temperature on noise performance. Noise outputs may be used just as other network output variables, not as some "special" additional feature. They are therefore able to equally participate in optimization and tolerance analysis/design.

9 NON-LINEAR CIRCUITS

Everything described so far, and the examples given, were for linear circuits and networks. However, AGILE supports *harmonic-balance* non-linear modeling as well. There are many references for this technique, but a short overview is given here. AGILE separates nonlinear models and linear models as shown in Figure 9.1. There are several nonlinear models in AGILE, including FETs and diodes, see the component document for details. The separation shown can occur in subnetwork NDL descriptions, that is a non-linear model can be present in a subnetwork, but for analysis it is separated after elaborating the complete description as in the figure.

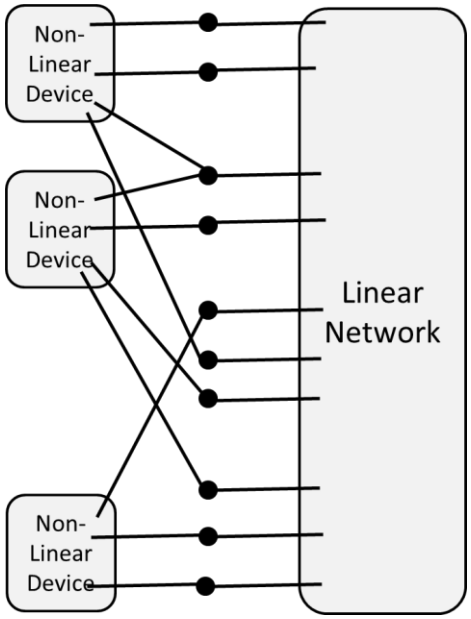


Figure 9.1 Nonlinear models separated for harmonic balance method

A set of nodes results that is the interface between the nonlinear models and the remaining linear network components. At this interface an initial voltage specification is generated at each harmonic (frequency) of the analysis. Note that this is a complex value – can be viewed as magnitude and phase or as a complex number. Given the harmonic voltages at the interface, each non-linear model must then generate its response as a complex current result at each harmonic as illustrated in Figure 9.2.

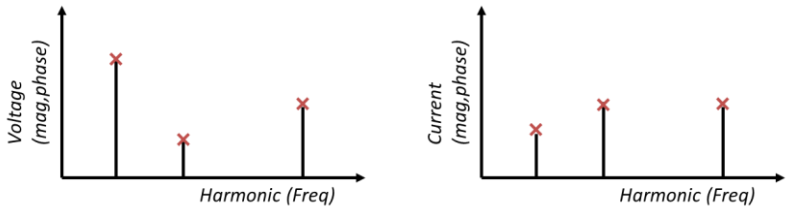


Figure 9.2 Given harmonic voltages, each nonlinear model generates harmonic currents

The harmonic balance method then computes the currents at each node and harmonic that these voltages would generate in the linear network. At each node and harmonic, the difference between the currents generated on the non-linear side and the currents on the linear side is the error that must be reduced. So the method proceeds to adjust

the voltages until these currents agree, hence the term harmonic balance. If this can be achieved the balance method has converged and provides a correct solution according to Kirchhoff's laws and if not then the method ends with an error message. Note that there is no guarantee that a solution found is unique, there could be a multitude of solutions that are valid.

Most of the nonlinear models operate in the time domain, that is the convert the harmonic voltages to a (periodic) time signal and compute the response currents in the time domain. These are then converted back to the frequency domain as the resultant harmonic currents. This process is subject to aliasing and sampling problems, it is the user's responsibility to ensure that significant signal content is captured in the harmonic set used.

Let's move on with a first example. We will introduce some new output variables along the way too. Figure 9.3 shows the NDL for an amplifier circuit with a nonlinear FET, a simple circuit diagram is contained as comments in the listing. It uses two subnetworks, inputt1 and outputt1, which are brought in along with trial1. For output results, it has Vsignal and Isignal terms that we've seen before, but it also has Pspectrum, Vspectrum and Ispectrum outputs as well. These are the power, voltage and current spectrums (frequency domain) at the port. Note that power is in watts, so 1000 times this is the power in milliwatts.

```
Network : trial1
*
* Simple Nonlinear Circuit
* ------ --------- -------
*
*        Consists of nonlinear FET with linear package elements
*        using sub-networks INPUT and OUTPUT
*
*        Frequency: 12 GHZ with 3 Harmonics
*        Input    : 50 ohms, 50 mW nominal ranging 10 -> 120 mW in
*        Output   : 50 ohms
*        Biasing  : fixed as model parameters (but allowed to range)
*
*
*        ______     ____       _____        ___     ___     _______
*    1  |:input| 2 |    |$gate|     |$drain|   |14 |   | 3 |:output|   5
*   ----|1    2|---|SRLC|-----| FET |------|RES|---|IND|---|3     4|-----
*       |______|   |____|     |_____|      |___| | |___|   |_______|
*           |      r=1.019       |       r=1.641 | l=.29       |
*           |      l=.295        |$source        |             |
* Port #1   |                   __|__            _|_            |     Port #2
* (50 ohms) |                  |     |r=1.762   |   |           |     (50 ohms)
* (sig:p_in)|                  |SRLC |l=.01387  |CAP|           |
*           |                  |_____|          |___|           |
*     0     |                    |              |             |
*   ------------------------------------------------------------------------
*
* PORTS:
  #1 1 0 p_in
  #2 5 0
* SIGNALS / FREQ
  p_level = 50 [ 10 : 120 : 20 ]
  fund_freq = 12e9
```

```
  SIGNAL p_in mW freq { fund_freq : p_level }  ;* p_level mW @ 12 ghz

  harmonics ghz 1 12 1 2 3   ;* defines freqs/harms
* COMPONENTS:
  * INPUT CIRCUITRY
  :inputt1 1  2  0   () { 1 2 0 }
  srlc     2 $gate   (r = 1.019 l=0.295 )

  * DEVICE
  fet $gate $drain $source                                          &
           (cgs=0.796 cgd=0.0254 cds=0.15 rin=1.6 rds=400 vbk=7.5    &
            rbk=417 ohms, ravl=1000 vpo=-4 vbi=0.6 gamma=1 beta=.008 &
            tau=7.69 psec, gcur=1.0e-6                               &
            a0=160 a1=66.57 a2=.955 a3=-1.089                        &
            vd=vd_dc vs=.2 vg=vg_dc)   ;* fixed bias

  vd_dc = 4.2 [ 3 : 8 : 1 ]
  vg_dc =  -1 [ -2 : 0 : .5 ]

  * SOURCE CIRCUITRY
  srlc $source 0   (r=1.762 ohms l=0.01387 nh)

  * OUTPUT CIRCUITRY
  ind       14  3     (l=0.29 nh)
  res       $drain 14 (r=1.641 ohms)
  cap       14  0     (c=0.0216 pf)
  :outputt1  3  5 0   () { 3 4 0 }

* OUTPUTS:
  vgate_mag   = abs(vspectrum$gate  ); vgate_ang   = phase(vspectrum$gate  )
  vdrain_mag  = abs(vspectrum$drain ); vdrain_ang  = phase(vspectrum$drain )
  vsource_mag = abs(vspectrum$source); vsource_ang = phase(vspectrum$source)
  vout_mag    = abs(vspectrum5      ); vout_ang    = phase(vspectrum5      )

  gate_c_m   = abs(ispectrum$gate)
  drain_c_m  = abs(ispectrum$drain)

  drain_power= 1000*pspectrum$drain   ;* power in mW
  out_power  = 1000*pspectrum5        ;* power in mW
  fund_power = fselect(fund_freq, out_power )
  fund_gain =  db(fund_power / p_level)

  in_time  = vsignal1
  out_time = vsignal5
  drain_volt = vsignal$drain
  drain_curr = 1000*isignal$drain   ;* mA
end
```

Figure 9.3 trial1 circuit

Let’s look at some plots of the results after analysis.

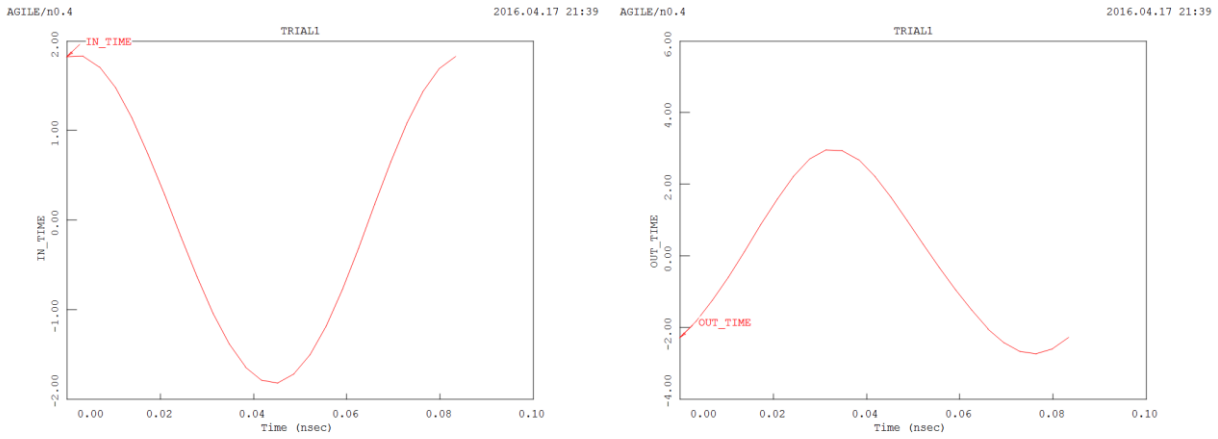


Figure 9.4 In_time and out_time of trial1

We see that p_in is the signal being attached to port #1. This is a nominal 50 mW power signal at the fund_freq which is set to 12e9 (i.e. 12 GHz). We the resultant voltage at port 1 in the figure above. On the right hand side of the figure we see the output voltage at node 5. This is definitely phase shifted from the input and is not quite a clean sinusoid.

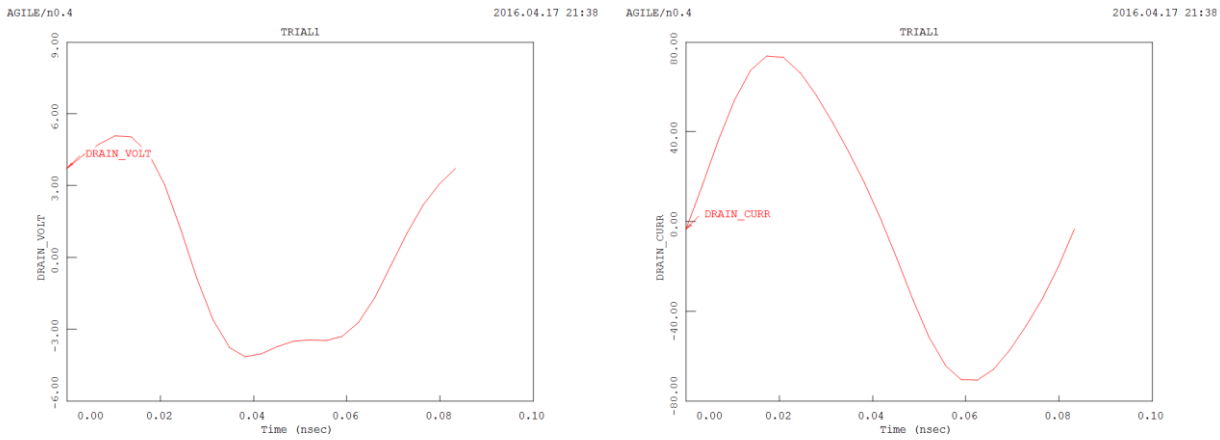


Figure 9.5 Voltage and current at the drain in trial1

Figure 9.5 shows the voltage and current at the drain of the FET device. The voltage is getting subject to some distortion and to a lesser extent so is the current.

So what is a current at a node, such as drain above? Doesn't the current sum to zero at a node according to Kirchoff and as a result of the balance? Why yes, it does, but the current (Isignal) of a node is reveling the current from one side of the linear/non-linear interface. So Isignal at an internal node only makes sense for nodes that are in the non-linear/linear interface. AGILE can also display the op-cycle, which is the current/voltage relationship at a non-linear node across the time period. This can be shown for a single harmonic, or as the sum of all harmonics as shown in Figure 9.6.

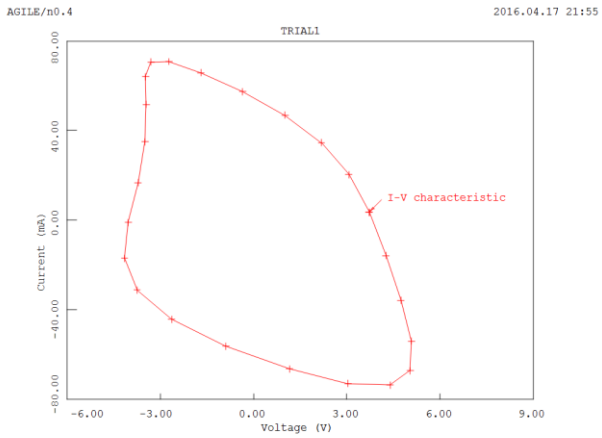


Figure 9.6 Opcycle at the drain for trial1

Finally, let's look at the power at the fundamental frequency as p_level (input power level) increases in terms of power in mW and gain:

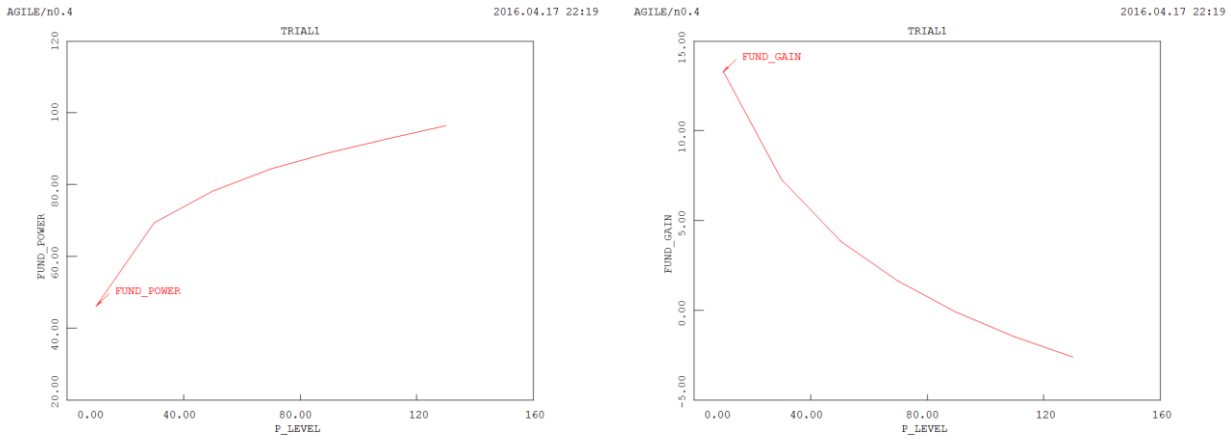


Figure 9.7 Power at the fundamental and gain in db as input power increases

We see that FUND_POWER is increasing as input power increases (figure left), but gain is not linear and is falling off the input power increases (figure right).

9.1.1 TRIAL2.NET

The network trial2.net is very similar to trial1 discussed above, but includes features to control DC biasing of the device. It uses the same two subnets as trial1, a listing is:

```
Network : trial2
*
* Simple Nonlinear Circuit
* ------ --------- -------
*
*        Consists of nonlinear FET with linear package elements
*        using sub-networks INPUT and OUTPUT
```

```
*
*       Frequency: 12 GHZ with 4 Harmonics (incl. DC)
*       Input    : 50 ohms, 50 mW nominal ranging 10 -> 120 mW in
*       Output   : 50 ohms
*       Biasing  : introduced through separate bias ports (bias at package)
*
*
*  PORT #drain_bias (r-0.1,l=100mH)(sig:drain_sig)
*  -------------------------------------------------------
*                                                        |
*  PORT #gate_bias (r=0.1,l=100mH)(sig:gate_sig)         |
*  ---------------                                       |
*                |                                       |
*        _______  |  ____       _____         ___     ___  |  _______
*    1  |:input| 2 |    |$gate|     |$drain|   |14 |   | 3 |:output|   5
*   ----|1    2|---|SRLC|-----| FET |------|RES|---|IND|---|3     4|-----
*       |______|   |____|     |_____|      |___| | |___|   |_______|
*           |      r=1.019       |       r=1.641 | l=.29       |
*           |      l=.295        |$source        |             |
* PORT #1   |                  __|__            _|_            |     PORT #2
* (50 ohms) |                 |     |r=1.762   |   |           |     (50 ohms)
* (sig:p_in)|                 |SRLC |l=.01387  |CAP|           |
*           |                 |_____|          |___|           |
*     0     |                    |               |             |
*   ------------------------------------------------------------------------
*
```

```
* PORTS:

  #1 1 0 p_in
  #2 5 0
  #gate_bias  2 0  gate_sig  srlc(r=0.1 l=100 mh)
  #drain_bias 3 0  drain_sig srlc(r=0.1 l=100 mh)

* SIGNALS / FREQ

  p_level = 50 [ 10 : 120 : 20 ]
  SIGNAL p_in mW freq { F1BASE : p_level }  ;* p_level mW @ fund

  SIGNAL gate_sig  voltage freq { 0 : vg_dc }     ;* gate DC
  SIGNAL drain_sig voltage freq { 0 : vd_dc }     ;* drain DC

  vd_dc = 4.2 [ 3 : 8 : 0.2 ]
  vg_dc =  -1 [ -2 : 0 : .2 ]

  harmonics ghz 1 [8:16:1]  0 1 2 3  ;* must include DC to bias FET

* COMPONENTS:

  * INPUT CIRCUITRY
  :inputt1 1  2  0   () { 1 2 0 }
  srlc     2 $gate   (r = 1.019 l=0.295 )
```

```
  * DEVICE
  fet $gate $drain $source                                          &
           (cgs=0.796 cgd=0.0254 cds=0.15 rin=1.6 rds=400 vbk=7.5     &
            rbk=417 ohms, ravl=1000 vpo=-4 vbi=0.6 gamma=1 beta=.008  &
            tau=7.69 psec, gcur=1.0e-6                               &
            a0=160 a1=66.57 a2=.955 a3=-1.089)

  * SOURCE CIRCUITRY
  srlc $source 0   (r=1.762 ohms l=0.01387 nh)

  * OUTPUT CIRCUITRY
  ind       14  3     (l=0.29 nh)
  res       $drain 14 (r=1.641 ohms)
  cap       14  0     (c=0.0216 pf)
  :outputt1  3 5 0    () { 3 4 0 }

* OUTPUTS:
  vgate_mag   = abs(vspectrum$gate  ); vgate_ang   = phase(vspectrum$gate  )
  vdrain_mag  = abs(vspectrum$drain ); vdrain_ang  = phase(vspectrum$drain )
  vsource_mag = abs(vspectrum$source); vsource_ang = phase(vspectrum$source)
  vout_mag    = abs(vspectrum5      ); vout_ang    = phase(vspectrum5      )

  gate_c_m   = abs(ispectrum$gate)
  drain_c_m  = abs(ispectrum$drain)

  drain_power= 1000*pspectrum$drain   ;* power in mW
  out_power  = 1000*pspectrum5        ;* power in mW
  fund_power = fselect(F1BASE, out_power )

  in_time  = vsignal1
  out_time = vsignal5
  drain_volt = vsignal$drain
  drain_curr = 1000*isignal$drain

  fund_gain = fselect( F1BASE, Pspectrum5 - Pspectrum1 )
  dc_input  = fselect( 0 , Pspectrum2 + Pspectrum3) ;* gate+drain bias DC
  pae = 100*fund_gain/dc_input
end
```

Figure 9.8 Trial2 NDL listing

Notice the signals `gate_sig` and `drain_sig` which are each defined to only have a DC component with value of `vg_dc` and `vd_dc` respectively. We can now analyze as a function of these values which are defined as range variables. A plot of the fundamental output power vs each of these is shown in Figure 9.9.

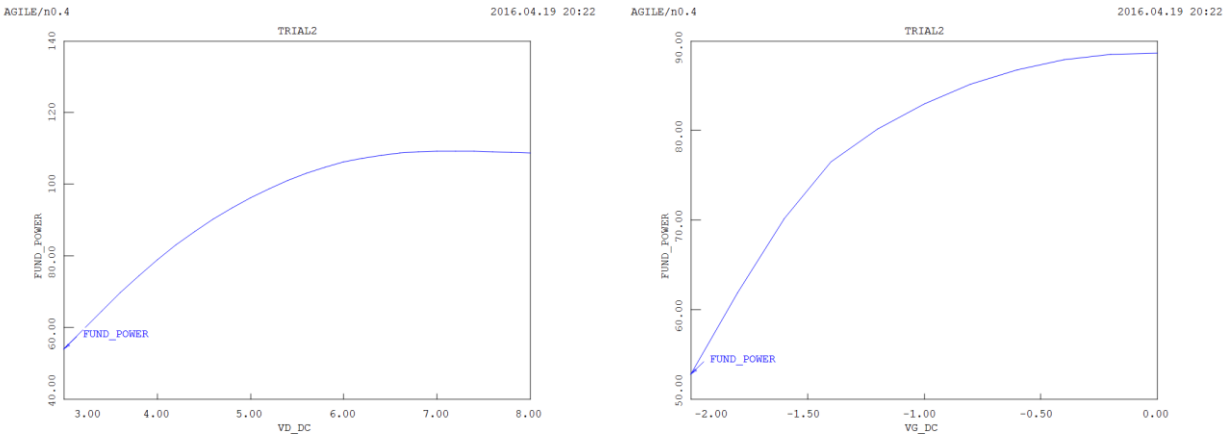


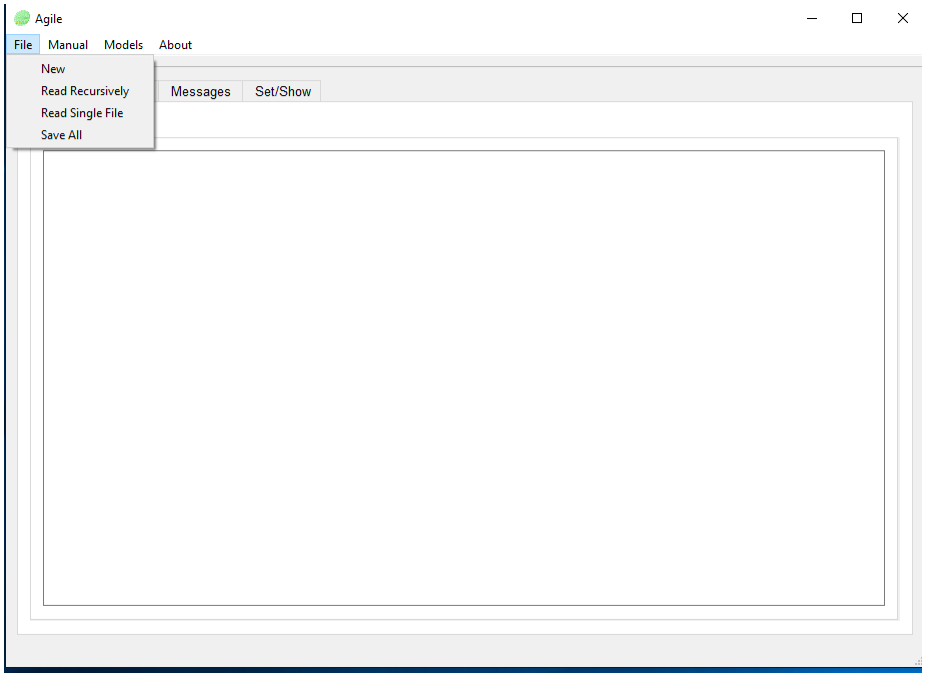
Figure 9.9 Fundamental power as a function of gate and drain bias

Of course, optimization could be used to optimize the circuit, including biases, according to a defined `Fmerit`. There is another provided NDL net called `trial22.net` that mimics `trial2` but wraps some of the linear components surrounding the FET into a submodel. The results should be the same but can vary a bit due to changes in harmonic balance convergence.

10 USING AGILE

All of the previous chapters were devoted to describing Agile's NDL and analysis. Now we finally see how to use it. The version that is described here is the Windows or Linux graphical user interface (GUI) version that leverages Qt (see: http://www.qt.io/).

When started the user will have a GUI that looks like:



Each of the three menu items, Manual, Models and About will cause a dialog or document to be brought up. The first item, File, has options to:

New Create a new network in AGILE. You must associate a file with the new content. Make sure that the name of the file and network name agree. That is if the network name (in the Network statement) is circuit17 then the file name *must* be circuit17.net

Read Recursively Read a NDL description and also read each subnetwork or port termination network referred to. This would be the typical means for reading files. This also removes all NDL descriptions that are currently loaded (following a confirmation dialog).

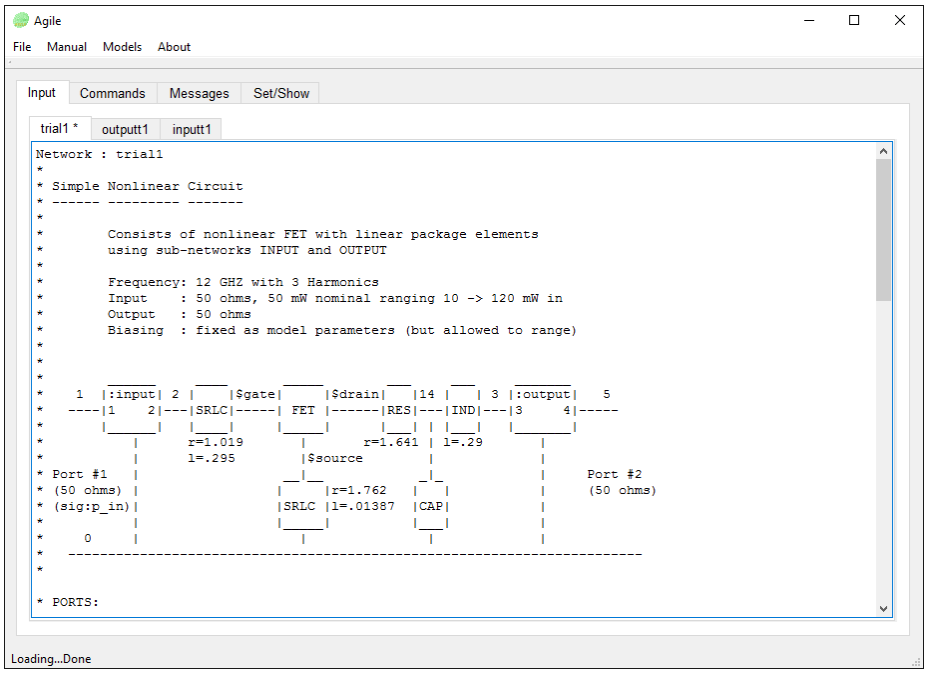
Read Single File Read in a single NDL file. This is needed to read global networks separately from other NDL files.

Save All Write (save) all changed files back to their files.

NDL files are searched in locations as specified in the config.xml file. This will include the location of the networks installed with the application and possibly other locations.

10.1 EDITING FILES – THE INPUT TAB

Once one or more files are read in or created new, each NDL description will be placed into its own tab. Each of these appears under the main Input tab. You can edit the files as expected switching between networks using the tabs:



10.2 THE COMMANDS TAB

The commands tab is where most of the analysis actions are. It is divided into four sub-tabs.

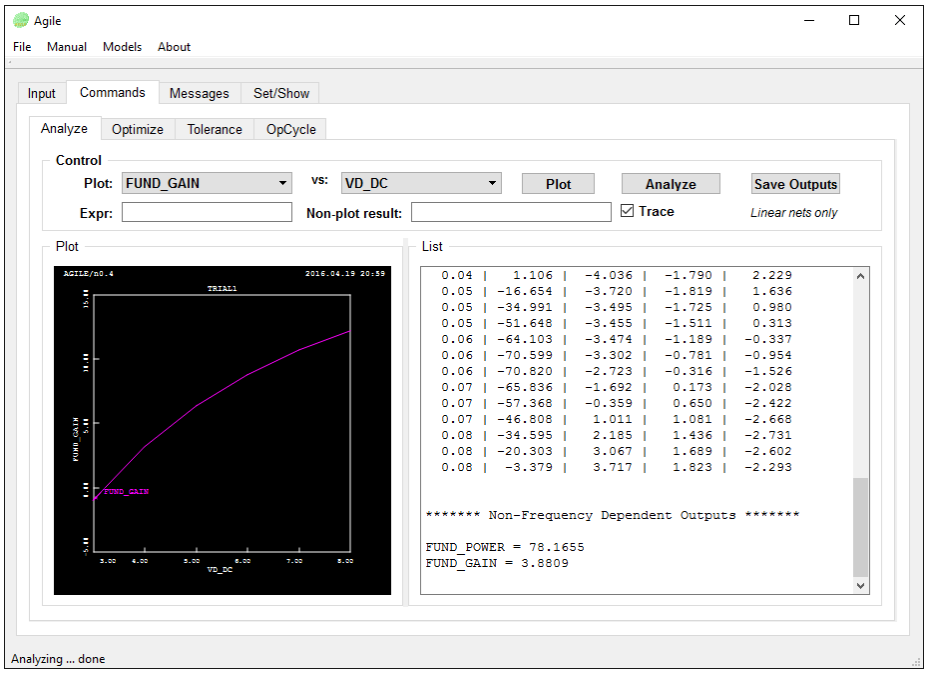
10.2.1 THE ANALYSIS TAB

This is where basic analysis occurs. The Analyze button is used to analyze the network that is selected on the Input tab. Sometimes, no re-analysis will occur if there have not been changes since the last analysis. The Trace checkbox allows tracing of values as described in section 3.5.

After analysis, Agile fills in the plot and ‘vs’ comboboxes. The plot variables are all of the ‘post analysis’ variables, namely the ones that depend on an output. The ‘vs’ combobox is filled in with range variates. You can select any of these and hit the Plot button to generate a plot of the result. We will talk more about how to use the ‘plot widget’ later. Note that there is a slider between the tabulated data on the right and the graphic on the right – use this to change the relative sizes (along with resizing the whole window if you wish). Typing Ctrl-shift-Enter in the textarea will clear the screen (not confirmation); typing ctrl-S wil cause a dialog to come up to save the file.

For results that do not generate a frequency, time or range variate sequence – that is values that are just single numbers – a plot is not generated but the text box (Non-plot result) is filled in to display the result.

There is also a textbox that says ‘Expr:’ – this is used to plot any output variable without having to put it in the NDL description. Note that if there is any text there that ***this takes precedence*** over the Plot combobox – to go back to using the combobox you have to erase the content of the Expr line. You can use the ‘vs’ option for these as well.

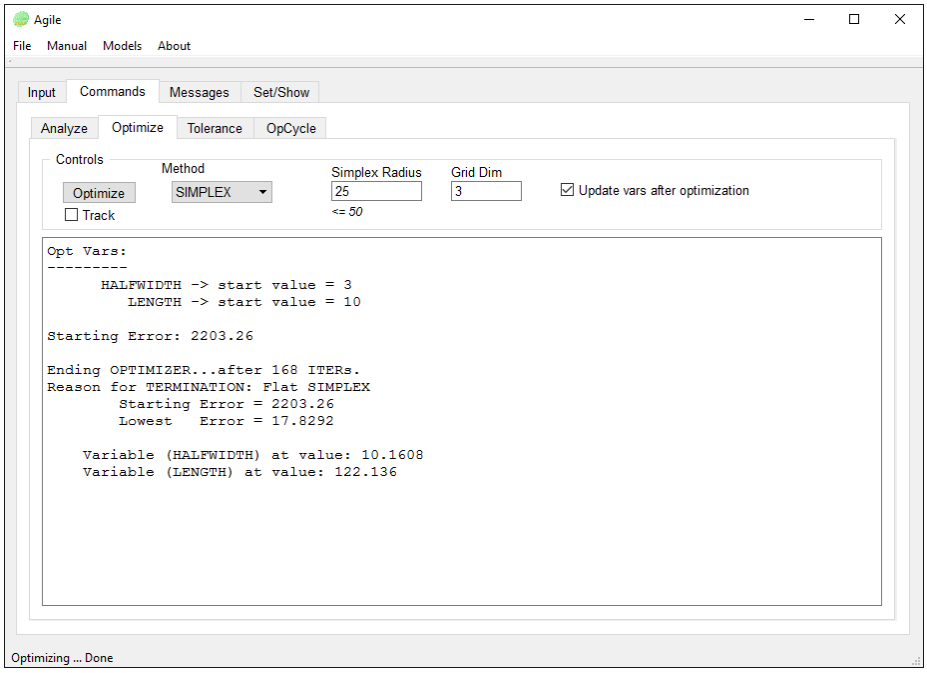


The Save Outputs button will write a linear network analysis result out as an AGILE tabulated network. This can be useful for saving large analysis results, etc. It will not work if there are non-linear models present.

10.3 THE OPTIMIZATION TAB

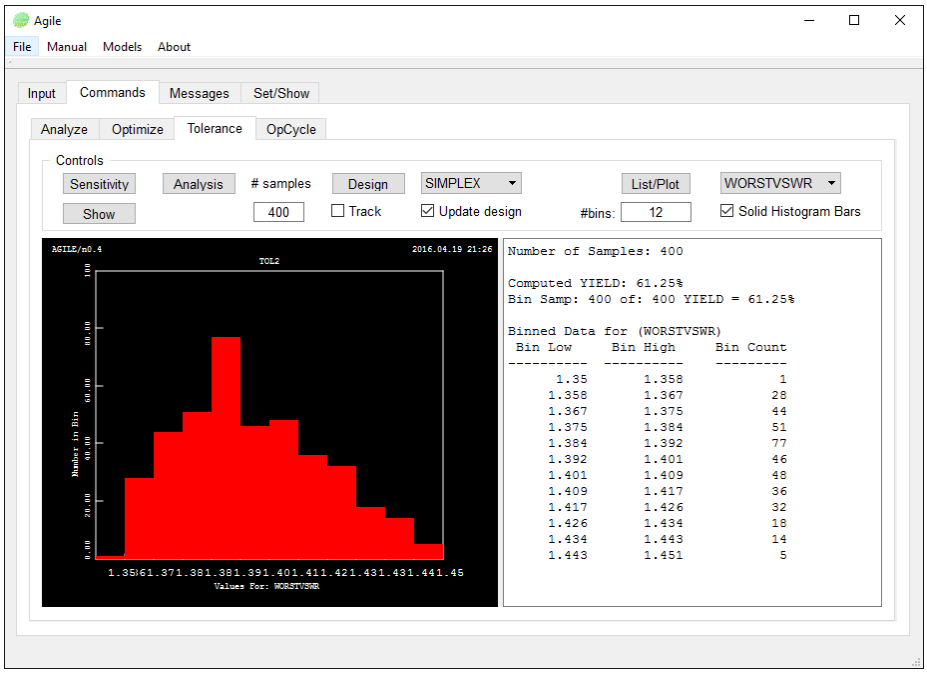
The Optimize tab allows you to launch optimization as discussed in Chapter 6. The tab is quite simple, having a Optimize button to launch the optimization process, which prints information into the text area. Use the combobox to select the optimization method, and the other boxes to set parameters as was described in Chapter 6.

The Track option turns on tracking for the optimization process, printing additional information as it executes. Finally, the *update vars after optimization* checkbox determines if the optimized range variable values are written back into the NDL file. If it is unchecked then the original NDL is not changed, otherwise the best values found so far are written into the NDL description (not back to the file yet, only into the application text description that can be saved elsewise). Please remember that range variables should be placed in descriptions on a line by themselves, that is don’t use continuation lines or multiple lines on one file line for these. Otherwise the variable write back may likely fail.



10.4 THE TOLERANCE TAB

The Tolerance tab is a bit more complex than the Optimization tab, but should be mostly self explanatory. To start, here is a screen shot after a tolerance analysis of the tol2 circuit. This tab has the same structure as the Analyze tab sporting a graphic to the left and text output to the right, with a slider between them.



Here are the buttons and interface features on this tab:

Sensitivity This will launch a sensitivity analysis as described in section 7.3.

Analysis This will launch a tolerance analysis as described in section 7.5. It will use the number of samples specified in the textbox.

Design This will launch a tolerance design as described in section 7.6. It will use the number of samples specified in the textbox for the tolerance analysis portion. It also uses an optimization method as chosen in the combobox. If the Update Design checkbox is checked then the results are written back into the NDL descriptions. The Track option will provide additional output during the process.

List/Plot This will generate histograms of the data requested in the rightmost combobox. The number of bins can be set here but the range of the bins always spans the range of the associated data – although the plot can be rescaled. If the Solid Histogram Bars is checked then you get filled in bars, otherwise empty boxes for the bins.

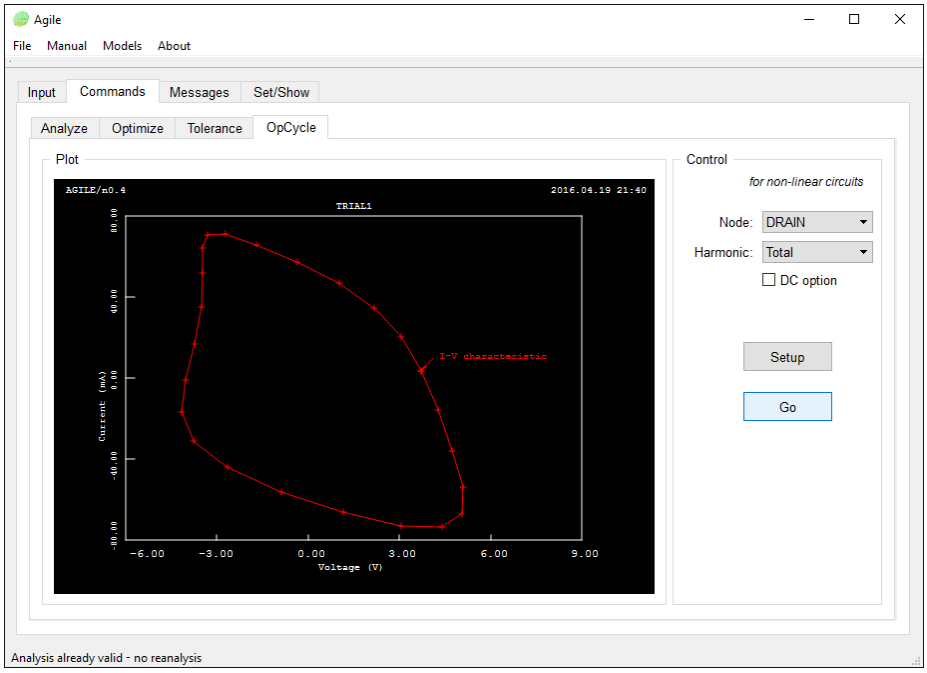
Show This will show information about the last tolerance command. For example:

```
Tolerance MAX Number of Variates = 20
Tolerance MAX Number of Targets = 20
Tolerance Meta Mode = None
Tolerance Design of Experiment = Orthogonal Array
Tolerance Recycle Random Sequence = No
Last TOLERANCE variate(s):
       C1
       C2
       C3
       L1
       L2
       L3
Last TOLERANCE target(s):
       AVERLOSSDB
       AVERVSWR
       WORSTLOSSDB
       WORSTVSWR
Computed YIELD: 61.25%
Number of Samples: 400
```

Tip: The number of tolerance variates and target outputs are limited to 20 right now. The design of experiments and meta mode material listed above doesn't apply to this version either.

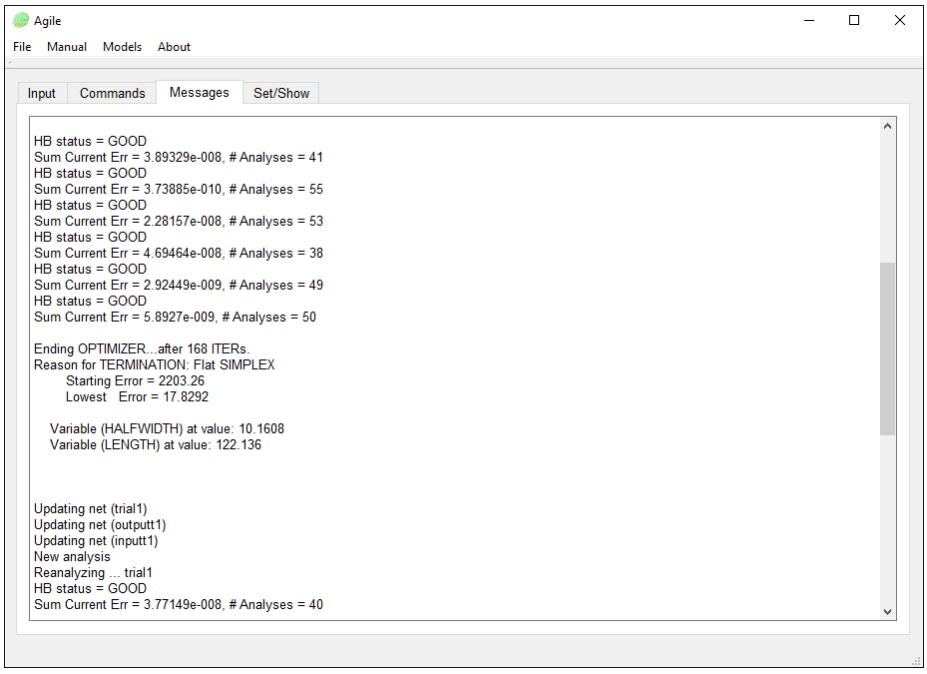
10.5 THE OPCYCLE TAB

The OpCycle tab should be simple to understand. Choose a node and harmonic, use the Setup button and then the Go button to generate the result. The DC Option specifies if DC offsets are included in the result.



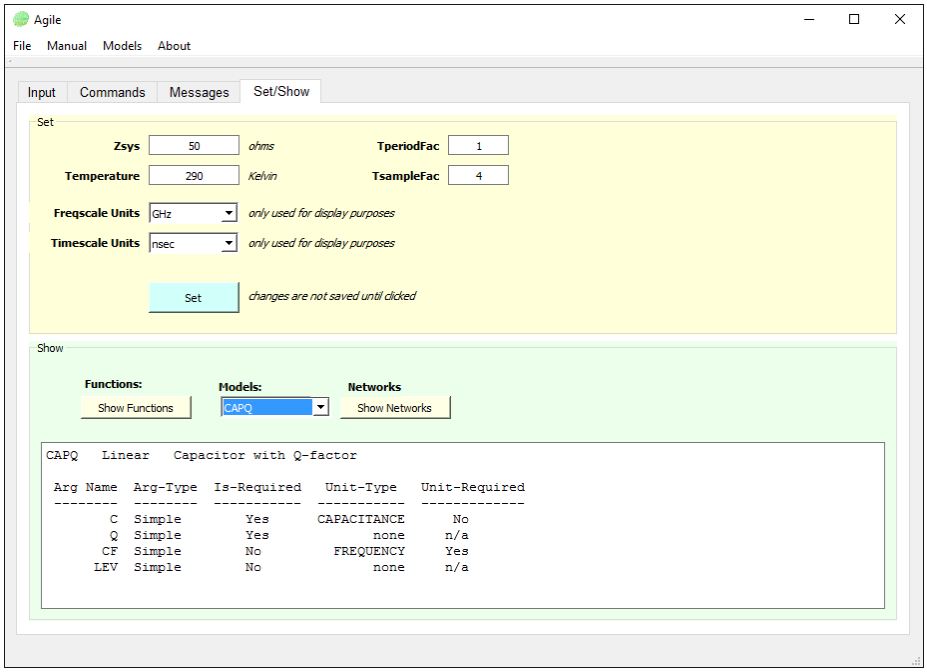
10.6 MESSAGES TAB

The Messages tab displays information as any command is executing. Always check for errors here and other information. An example



10.7 THE SET/SHOW TAB

Several settings can be changed on this page, as well as showing some useful information. An example:



Most of this should be self-explanotory. The units shown there don't change the NDL meanings of reserved names (e.g. FREQ which is always in Hz) and are only used during display of data. The TperiodFac and TsampleFac alter the number of periods and the time sampling intervals for displays as well.

Please note that you must hit the Set button for changes to take effect.

> Tip: There are other settings that are changeable in the config.xml file

In the Show section, you can display information about the NDL functions available, about the models and the network that was last analyzed.

10.8 USING THE PLOT WIDGET

The plot widget appears in multiple places in the GUI, on the Analysis tab, the Tolerance tab and the OpCycle tab for example. It works the same in each place.

> Tip: remember that complex-valued data must be plotted in complex form. You might use the RTOP function to convert a complex number to polar form, such as S11, for the listing, but it has to be in complex form for plotting.

Once a graph is generated, a three item menu will appear whenever you hover the mouse over the widget. The top item is the Save, the middle one is Invert and the last one will bring up a dialog for adjusting the parameters of the graph such as colors and scaling. It looks like this:

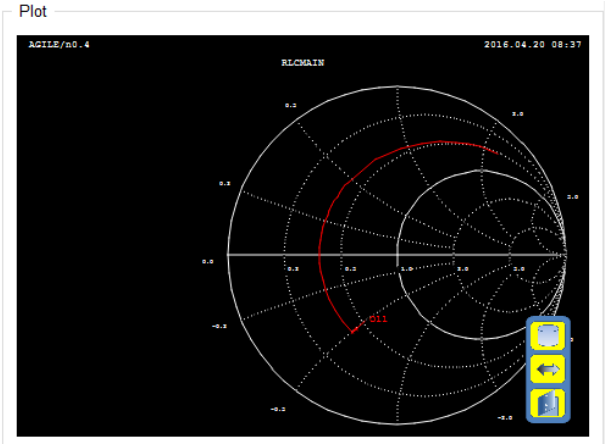


Figure 10.1 The plot widget showing the popup menu.

The save button will bring up a dialog for saving the graphic. This is how all of the graphs in this document were generated. Note that this graphic is saved at a resolution of 1024x768 regardless of the current screen size for the graph. Actually this size is set in the config.xml file and can be changed there for future runs (the config.xml file is only read at startup).

The invert button will reverse the black and white colors for the graph.

The menu button at the bottom will bring up the graph edit dialog, for example:

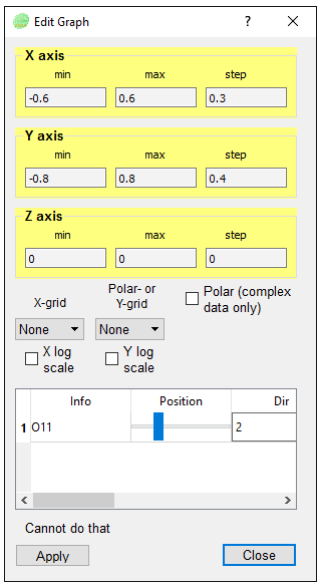


Figure 10.2 The graph edit dialog

Related to the scales: Each axis can be rescaled according to a min/max/step format. For X or Y scales that set as 'log scale' these numbers should be the exponent for the power of ten on the scale. For example, if you want a logarithmic scale from 10^{-3} (0.001) to 10^{2} (100) use a min/max of -3 and 2 respectively. The default Smith chart for complex data can be changed to polar form with the displayed checkbox. The subsequent polar chart can be scaled

using the Y-axis values, the default is scaling to a radius of 1 (as for the Smith chart). Different gridding styles for each axis can also be set.

At the bottom of this dialog is a listing of each curve (dataset) that is on the graph. Here you can adjust (in this order, not all options are visible in the figure above): the label for the curve, its position (index point on the x-axis), the label direction (1,2,3,4 point to NE, NW, SE, SW compass directions), marker style (0,1,2,3,4), line style (a combobox) and color (0 to 8).

10.9 INSTALLING

For Windows a typical installer is used. You can install and uninstall as any other program. The user that installs the application will also get files installed to their local application area with a link placed on the desktop. These contain configuration data, example networks (all those in the manual) and user manual. A link for the executable will be placed on the desktop too.

For Linux a 64-bit executable file is provided along with the same user data. This user data should be placed on the User's desktop.

11 APPENDIX

11.1 FAQS / HINTS

When plotting complex data the graph looks like vertical lines or is otherwise corrupted

Complex valued data must be plotted in complex format, perhaps the data plotted is in polar form – for example using RTOP?

11.2 CONFIG.XML

The file config.xml is read at startup to define various parameters.

11.3 ERROR SUMMARY

NEEDS UPDATE

Here is a summary of all the error messages that AGILE generates. They are in alphabetic order. Most errors, warnings and disasters occur at a particular time, currently either at "insertion" time (denoted as insert-time in the following) or at "analysis" time (denoted as analysis-time). Note that "insertion-time" occurs whenever NDL is changed, for example when using the iline or even the cx commands. "Analysis-time" occurs whenever an analysis must be done, for example when issuing the list outputs or plot var-name commands. Other "times" are indicated in a self explanatory way.

All disasters are listed first, followed by the errors, and ending with the warnings. Disasters are usually not the fault of the user but generally occur when AGILE does not have sufficient "configuration-sizes". All parameters which might lead to these disasters are configurable by the AGILE installer, assuming the "source" version of AGILE is at hand. Errors are real errors in that something is wrong with the user specified NDL. Of the warnings, some are simply messages to "remind" you of something or to emphasize a point, others indicate something that you should truly be wary of. Since these vary in severity please consult the actual warning message listed below to determine your level of concern.

Note: any errors not listed, like those which are totally unexpected, are intended to be self explanatory.

This is an extensive list and is not really intended to be "read" but rather referred to after an error occurs. Depending upon your level of experience with network analysis programs, it might be instructive to "browse" through this, but take a deep breath first...

Disaster - Couldn’t malloc: *object*

Dynamically (run-time) allocated memory is used for various data storage operations. Usually, AGILE will quit whenever it cannot obtain the memory it needs for operation.This should not happen, its occurrence depends on your system configuration (amount of memory, swap space, etc.)

Disaster - Input NDL line too long - MAX_CHAR_PER_NDL

A line was attempted to be inserted (or read from a file) which is longer than that currently allowed. Use the `SET MAXIMUM NDL_READ` command to change the size allowed.

Disaster - Too Many Bins Specified - MaxBins

This will only happen when you override the automatic binning for histograms, you must specify fewer bins.

Disaster - Too ManyNodes in Network (*name*) - MAX_NODEINS

A network has more nodes in it than is currently allowed. Use the `SET MAXIMUM NODES` command to change the size allowed.

Disaster - Too Many Nonlinear Components - NLCOMPS

A network has more nonlinear components in it than is currently allowed. Use the `SET MAXIMUM NLCOMP` command to change the size allowed.

Disaster - Too Many Optimization Vars - MAXOPT

AGILE is configured at compile time to allow for a maximum number of target variables in optimization. This might typically be 51, which is quite a few variables for any optimizer to handle. This parameter can be reconfigured by your AGILE installer.

Disaster - Too Many String Tokens - MAX_STR_TOKENS

Too many variable/function names are on the line you are trying to insert.

Splitting it into separate lines (if possible) will correct the problem.

Disaster - Too Many Tokens on this line

Too many "items" (tokens) are on the line you are trying to insert. Splitting it into separate lines (if possible) will correct the problem.

Disaster - Too Many TOL Targets - MaxTolTar

Only a maximum number of target variables may be specified for a tolerance command, which was exceeded. This parameter can be reconfigured by your AGILE installer.

Disaster - Too Many TOL Vars - MaxTolVar

Only a maximum number of variates may be specified for a tolerance command, which was exceeded. This parameter can be reconfigured by your AGILE installer.

Disaster - Un-Invertable Matrix

This occurs when AGILE is trying to convert between various network parameter formats (S-, Y-, Z-parameters). Are you sure your network is connected properly, if so are you trying to analyze "very idealized circuits"?

Error - , Expected...Could Not be Found

A comma was expected as part of the input but could not be found.

Error - : Expected...Could Not be Found

A colon was expected as part of the input but could not be found.

Error - } Expected...Could Not be Found

An ending curly brace was expected as part of the input but could not be found.

Error - (*name*) is Complex-Valued

Error - (*name*) is Frequency Dependent

The named variable (on the line which is displayed) is not allowed to be complex-valued or frequency dependent in the context shown.

Error - (*name*) is not in Tolerance Data

The `TOLERANCE LIST/PLOT` command was used with the name of a variable that is not in the sampled set. Use the `SHOW TOLERANCE` to see valid names.

Error - (*name*) is not Start,Stop,Step

When issuing the `LIST/PLOT` *`var1`* `VS` *`var2`* command, the variable named in the "VS" part must be defined as a range variable.

Error - (*name*) is READ-ONLY...you cannot save it

The `SAVE` command was used for a file that was read as a "read-only" (using the `-RO` switch on the `RR` command).

Error - (*name*) is Time-Dependent

The named variable (on the line which is displayed) is not allowed to be time dependent in the context shown.

Error -Added TAR targets must be non-FD/TD

The `TOLERANCE +TAR list` command option was used where the target being added was frequency or time dependent, this is not allowed. See chpater 8 for more information.

Error -All TURN values must be defined

The transformer model requires all turn ratios of 1 - NUM to be defined.

Error -All Vars/Goals Cannot Be Resolved in Net (*name*)

This message informs you of "circular" variable definition. Consider the two NDL lines: X = T AND Z = T, which will generate this message as well as other, more complicated situations.

Error -All Vars/Goals Cannot Be Resolved in Net (*name*)

This message informs you of "circular" variable definition. Consider the two NDL lines: X = T AND Z = T, which will generate this message as well as other, more complicated situations.

Error -Argument (*arg-name*) has Complex Value

Error -Argument (*arg-name*) has Illegal Value

Error -Argument (*arg-name*) is Required but Missing

Error -Argument (*arg-name*) Unrecognized for this Component

These are the various errors related to component arguments, they are self explainatory.

Error -Argument Specified Illegal

One of the arguments for the nonlinear element listed is wrong or illegal.

Error -Argument Value < 0 for Function (SQRT) What more can we say?

Error -At Least Two(2) Nodes Must be Requested from Sub-Nets

The node list of a sub-model invocation must be at least two nodes, otherwise it can have no effect.

Error -At Least Two(2) Nodes Must be Specified for Connection

The connection statement requires that at least 2 nodes from any component be connected, otherwise it can have no effect.

Error -ATT Requires CF/EL Specification

Use of the ATT component argument requires that the component be defined in terms of center freq and electrical length, rather than physical length.

Error -Both CF and EL Required

In components, if either CF or EL is used, both must be specified.

Error -Both W1/W2 Must be Specified for *comp-name*

The component *comp-name* requires that both widths be specified.

Error - (*symbol-name*) Cannot be Defined, Only Used in Expressions

The special variable *symbol-name* is reserved, it cannot be defined as the left hand part of an expression.

Error - (*symbol-name*) cannot be Displayed VS (*symbol-name*)

The first symbol/goal cannot be listed/plotted VS the second.

Error -Cannot Extrapolate Lower/Upper Freq in (*net-name*)

Tabulated sub-models are only valid within the frequency range defined, the main network is specifying a frequency outside these limits.

Error -Cannot Find Node (*node-name*) in (*net-name*)

In the instancing of *net-name*, the node *node-name* was requested, it is not defined in the sub-model.

Error -Cannot Have NOISE_DATA and NOISE_MATRIX

A tabulated network can only use either the two-port NOISE_DATA format or the n-port NOISE_MATRIX format to specify noise data, both in the same network is illegal.

Error -Cannot Mix TURN and COIL/COEF parameters

The transformer model uses either turns ratios or inductances and coupling coefficients, but they cannot be mixed in the same component.

Error -Cannot Specify *component-parameter combinations* The combination of component parameters is illegal.

Error -Cannot use Linear Outputs for Nonlinear circuit

Only voltage and current outputs (`VSPECTRUM, ISPECTRUM, VSIGNAL, ISIGNAL`) outputs are legal for nonlinear circuits.

Error -Cannot Use OPCYCLE command for linear networks

The OPCYCLE command is only valid (an useful) for nonlinear circuits.

Error -CF Must be Specified for LEV=#

The SRLC/PRLC component requires CF for level # definition.

Error -Colon Expected

A syntax error occurred where a colon (:) was expected but not found.

Error -Color *color-name* is unknown

The color specified is not one of those known. See DISPLAY CURVE # COLOR command for details.

Error - Complex Argument Required for Function (*func-name*) The function name requires a complex argument.

Error -Complex Value Illegal as TOLERANCE Variate

The +VAR phrase was used in a tolerance command, where a variable being added was complex valued, this is not legal.

Error -Complex-Valued Argument(s) for Function *func-name* The function named does not accept complex arguments.

Error -Component Argument (*comp-arg-name*) has Illegal Expr

Error -Component Argument (*comp-arg-name*) is Analysis Dependent

Error -Component Argument (*comp-arg-name*) is Dependent on Full Analysis

Error -Component Argument (*comp-arg-name*) is Time Dependent

All of these errors relate to the value or expression used to define the named component argument. Basically arguments to a component cannot be analysis- or time-dependent or dependent on a goal.

Error -Component 'Env' List Misformed

The environment must be a reference to a single environment, a phrase such as: `... , env = x+1, ...` will cause this.

Error -Component Parameter List Misformed

The syntax of the component parameter list is not understood, a phrase such as: `..., W= , ...` will cause this.

Error -COST must be specified for TOL DESIGN

The Cost statement must appear in order to do tolerance design.

Error - Curve # *integer* Does not exist in Selected Display

You specified an display operation (using display curve #) on a curve number *integer*. However this curve does not exist in the selected display.

Error - Curve LineStyle *integer* is illegal

You specified a curve line-style index which was outside of the correct range of 1-8.

Error - Curve MarkerStyle *marker-style* is unknown

You specified an un-recognized marker style. Please refer to the display command summary for the correct names of marker styles.

Error - Data Format Line: *number* - Bad Form or Wrong Number Entries Tabulated data was entered that is not understood, please check it.

Error - Data Statement Without Any Data Points

Tabulated data was entered that is not understood, please check it.

Error - DBM/MW Signal Requires Real-Valued Data

A signal was defined in the DBM mode and the expression or list evaluated to a complex value, this is not allowed.

Error - Defined Symbol is both Time AND Frequency Dependent

Variables may not be both time and frequency domain, an NDL line like:

`X1=time*freq` will cause this.

Error - DIRECT token must be last

The port-termination specification DIRECT must be last in the port statement.

Error - Direction *direction* is unknown

You tried to display curve integer label dir command and specified an unrecognized direction. Please refer to the display command reference for correct directions.

Error - Display # *integer* does not exist

You used a display select integer command to attempt selecting a non-existant display. Use the display show command to print the indices of displays which do exist.

Error - Display Type *display-type* is UnKnown

You tried to change display types of the selected display and the type specified was un-recognized. Please refer to the display command reference for correct types.

Error - Displays of different types cannot be merged

Display merging is only allowed between displays of compatible data types. For instance merging complex data with real data versus frequency is not allowed.

Error - Distribution Expr is Complex-Valued

The expression in the distribution part of a variate evaluates to a complex value, this is not allowed. Simple example:
`X1 = 5 {NORMAL 3*(3,4)}`

Error - Distribution (*dist-name*) is Unrecognized

The named *dist-name* is one of those known (NORMAL, UNIFORM, PERCENT, LOGNORMAL).

Error - Duplicate Argument (*arg-name*) Illegal

Error - Duplicate Argument Specified (*arg-name*)

The argument *arg-name* in the component argument list is not allowed more than once.

Error - Duplicate Definition for Data at Freq: *number*

A Tabulated network contains data specified at the same frequency twice.

Error - Duplicate Definition for Goal (*goal-name*)

Error - Duplicate Definition for Port: #*port-name*

Goals may not be defined with the same names as other goals or variables or components, etc. Port names must be unique.

Error - Duplicate Formal Parameter

The names of the parameters of a sub-model must be unique.

Error - Duplicate Naming Statements

Only one network statement (or TABULATED, GLOBAL, etc.) is allowed per network.

Error - Duplicate Temperature Statements

Error - Duplicate PASS Statements

Only one temperature/PASS statement is allowed per network.

Error - Either CF/EL or PL Required

The component requires either set of these parameters.

Error - Either TURN or COIL/COEF Form Required

The component requires either set of these parameters.

Error - Environment (*env-name*) is Missing

The named environment was used but not defined.

Error - ER Cannot be Specified with CF/EL What more can I say?

Error - Expected ’EXP’ as Fourth Token in Frequency Statement

If the exp term of the frequency statement is misspelled this will occur.

Error - Expected Parenthesis Not Found

The syntax parser expected a parenthesis and none was found.

Error - Expected Sub-Net or Component

The port statement allows either a sub-model or component as a port termination, this message will result if it is incorrect.

Error - F1SWEEP Requires Harmonics F1-range

Use of the list/plot var VS F1SWEEP requires that the harmonic statement be defined in the F1 sweep format.

Error - FET model drain bias too low

The FET model is operated outside of its valid range.

Error - First Arg of (IF) must be Real-Valued

The first argument of the three argument IF() function must be real-valued.

Error - File OPEN Error!

A "generic" (unexpected) file access error has occurred. Check system/ directory privileges, etc.

Error - First Arg of (IF) must be real-valued

The IF function requires that the condition part be real-valued.

Error - F-Points in Harmonics Must be >= 0

Negative harmonic integer numbers are not allowed.

Error - Freq-Domain Signal (*sig-name*) Cannot be Time Dep.

If the signal is defined as being in the frequency domain then the expression or list cannot be time dependent.

Error - Frequencies Not Defined

At least one frequency (harmonic) statement is needed in any main network.

Error - Frequency Part of S-List Expression Must be Real-Valued

The first part in the signal list of points represents the frequency (or time) point for the value, this must be real-valued.

Error - (FSELECT) First Arg Must be Real-Valued

The first argument of the function FSELECT selects the frequency point, this must be real-valued.

Error - Gate Bias (VG) required unless DC balancing

The gate bias for the FET model may be established either by the VG argument or gotten from the DC component of the harmonic balance. If the VG argument is not present and the "0th" (DC) harmonic is not present, this error will occur.

Error - Goal (*goal-name*) cannot be Displayed VS (*var-name*)

The LIST/PLOT goal command cannot be done with the "VS" part.

Error - Goal Upper (2nd) Value Less than Lower (3rd)

When inserting a range variable, the upper point (second expression) was less than the lower frequency point (third expression).

Error - Harmonic (#) Not Avaiable

Various commands (i.e., OPCYCLE/CONTOUR) have a harmonic number as an argument, if this is outside of the range of the last analysis then this error will result.

Error - Harmonic Requires # F-Points Second

The second token (after the HARMONIC keyword, i.e., HARMONIC GHZ *number*) must be a positive integer.

Error - Harmonic Requires Frequency Unit First

The first token (after the HARMONIC keyword) must be a unit of frequency.

Error - Harmonic Statement Incomplete

A basic syntax error exists in the harmonic statement, check it.

Error - Harmonic Statement May Only Appear Once

Unlike the frequency statements, of which multiple ones are allowed, the harmonic statement is only allowed once per network.

Error - Harmonics Statement Generates No In-Band Tones

The "multi-tone" form of the harmonic statement is used (# of freqs > 1) and the limits placed on the band (last two #'s) is set such that none of the tones falls in the band. This is not allowed.

Error - Illegal # of Data Points

The data statement requires a frequency, followed by N^2 real number pairs.

This message results if that criteria is not met.

Error - Illegal CompDef in Signal/Port Term. Position

The syntax of the port statement is incorrect, the signal attachment should precede the port termination, which is either a sub-model instance or a twonode component.

Error - Illegal Display chosen for Merge

One of the integer arguments which choses the display indices to merge is illegal (i.e., less than one or greater than the maximum allowed).

Error - Illegal Display Type Conversion

The attempted conversion on the selected display is illegal. For instance, converting complex data to real data versus frequency is not allowed.

Error - Illegal END Statement

The END token must appear alone.

Error - Illegal Format for Signal

The syntax of the SIGNAL statement is incorrect, check it.

Error - Illegal Formal Parameter Default Value

Error - Illegal Formal Parameter Format

The format for the formal parameters is wrong, the format is the formal-name, optionally followed by an "=*number*" or "?*number*", with each parameter separated by commas.

Error - Illegal Item in Data Statement

Only numbers are allowed on data statement lines.

Error - Illegal Item in F-Points of Harmonic Statement

Error - Illegal Item in I-Points of Harmonic Statement

Error - Illegal Item in Order/BandLow/BandHigh part of Harmonic Statement

The frequency point (F-Points) part of the harmonic statement allows for only non-negative real numbers, and the integer points (I-Points) for the single tone form of the allows only non-negative integers. If the multitone form is used the order must be a positive integer and the band low and band high values must be positive real numbers.

Error - Illegal KEYWORD Statement

The statement in question has been partially recognized as a keyword statement but it has incorrect syntax..Check chapters 2 and 3 for correct keyword definition syntax.

Error - Illegal Negative Node in Port Def

The negative node of the port definition statement is wrong, it must be a node name (i.e., $node_name) or a non-negative integer.

Error - Illegal Positive Node in Port Def

The positive node of the port definition statement is wrong, it must be a node name (i.e., $node_name) or a non-negative integer.

Error - Illegal Token in Signal/Port Term. Position

The item found after the negative node of a port statement must be either a signal reference (simple name) or a sub-model or component instance.

Error - In graphics initialization...Aborting plot

This only occurs when AGILE cannot initialize the console graphics. Possible causes are: you are not runnning on the console, there are not enough computer resources available (i.e., number of open windows, memory, etc.)

Error - Index Required for Argument (*arg-name*)

Error - Index Unexpected for Argument (*arg-name*)

Error - Index.Index Required for Argument (*arg-name*)

The named argument, *arg-name*, has bad or missing index(s) associated with it. For example, the "W" argument in a TRL should not have an index on it (ala W3), in this case the second error will result.

Error - I-Points in Harmonics Must be >= 0

The integers used as the harmonic integer points must be non-negative.

Error - Lexical Analysis char =*char* OR (hex=*hex-number*) at char. position *integer*

A character is part of the NDL line that AGILE does not recognize as part of its character set. For example, the vertical bar (|) or any "control-characters" are not part of the characters that AGILE uses. They will cause this error, check your input for extraneous characters, *integer* tells you where to look.

Error - LOG axes cannot contain data<=0

In order to convert to logarithmic axis, the data on that axis cannot contain zero of less than zero.

Error - Lower > Upper Value for Symbol (*var-name*) at Freq = #

The evaluation of the lower expression exceeds the value of the upper expression for the range variable *var-name*.

Error - Magnitude of Correlation Coef(*#,#*) too large in CNCGEN Correlation coefficients must have a magnitude less than one.

Error - Main Network (*net-name*) Must Have Default Formal Parameter Values If your current network, which is always the one your analyzing, has formal parameters they must have default values in order to analyze.

Error - MisFormed Frequency Statement

The frequency statement has several three forms: `# f-unit,` `# # # f-unit,` and `# # # EXP f-unit`. Other forms will cause this error.

Error - Mismatched {} Brackets

Error - Mismatched Parentheses

The syntax of NDL is such that curly brackets or parentheses () must always appear in pairs, please check your syntax.

Error - Missing Right Curly Bracket

Error - Missing Right Parentheses

Either a curly bracket "{" or parenthesis "(" was expected but not found, check the syntax of the line.

Error - Multiple Outer {} Brackets

Error - Multiple Outer Parentheses

The syntax of component definition uses both parentheses for component argument definition and curly brackets for sub-model node lists. The syntax is flexible enough to place these in several places on an NDL line, but only one set should appear.

Error - Must have 4-ports or less for SAVE OUTPUT TOUCHSTONE

The `save output touchstone` command was used on a network with greater than 4 ports. See section 4.9.1 for details.

Error - Must Select A Port for CONTOUR

Error - Must Select A Variable for CONTOUR

Error - Must Select A Variable for OPCYCLE

The CONTOUR/OPCYCLE command was in error.

Error - Must Select First

The NDL Editor of the X Windows was used improperly, a network must be present before it can be read into the NDL Editor.

Error - Must Set '*object*' >= *number*

The `SET MAXIMUM ...` command was used to run-time adjust the maximum size of *object*, for each *object* type, however, there are minimums that must be adhered to.

Error - Name (*name*) Already Defined as a *object*

NDL names for: variables, goals, environments, components, etc. must be unique, this message informs you that this is not the case.

Error - Negative Data Range not allowed on LOG Scale

This error occurs when specifying upper and lower bounds for displays that are logarithmically scaled. you cannot specify either as negative numbers.

Error - Net (*net-name*) has no Connection Statements

Error - Net (*net-name*) has no Frequencies

Error - Net (*net-name*) has no Nodes

Error - Net (*net-name*) has no Ports

What more can we say? If a network is defined as a main network, than it must have these four things. The exception to this is when *none* of these are present, which is allowed, to provide for curve fitting, etc. applications.

Error - Network Analysis with No Resident Networks

A command which directs AGILE to analyze (list outputs, plot var-name, etc.) was used before an NDL network was read in or defined.

Error - Network (*net-name*) is Wrong Type of Network in Analysis Hierarchy

A network was used as a sub-model reference which is loaded but is of the wrong kind (like a GLOBAL).

Error - Network (*net-name*) Not Resident

The named network was used as a sub-model, but it has not been loaded in AGILE.

Error - No Circuit Loaded for *command* command

The *command* was used before any NDL networks were read in or defined.

Error - No Current Circuit - Cannot OVERWRITE

You cannot use the OVERWRITE command unless a network has already been restored or defined.

Error - No File *name* with recognized extensions

You said: rr *name* and there is no cooresponding system file name. Perhaps you forgot to set path? Some of the extensions are ".net", ".s", ".s2p", etc., see chapter 3 for more information.

Error - No FMERIT for Optimization

The optimize command was used when the current network does not have a FMERIT definition. Optimization requires the definition of FMERIT to be used as the objective, see chapter 6.

Error - No Harmonics Single F-Point Statement

The Harmonic statement has fewer tokens in it than is legal, check the syntax.

Error - No Name Specified! Use NETWORK: name

You cannot use the save command unless the network has a name. Use a network definition statement to name it (chapter 2).

Error - No Network to Initialize

When using any tolerance command, a network must be loaded first.

Error - No Network to Save!

The save output command was given, but there isn't any current network to save the outputs of.

Error - No PASS statement for Tolerancer

When using either the tolerance analysis or tolerance design commands, a PASS statement must be included so that AGILE can compute yield.

Error - No Such Model (*name*)

Name was used as a built in model reference, however, there is no model of that name. Use the `SHOW MODEL` command for more information.

Error - No Possible Nodes for OPCYCLE

Error - No Possible Variables for CONTOUR

The command was used in a case where there is no possible correct selection.

Error - No Such Model (*model-name*)

Model-name was used as an internal model, but there isn't such a model loaded in AGILE.

Error - No Such Network Indexed: *number*

The `SWITCH` *`number`* command was used when there aren't that many networks loaded.

Error - No Such Port : *port-name*

The command issued refers to *port-name* as a port, but there isn't one of this name in the current network.

Error - No Such Variable/Goal/Signal: *name*

The command issued refers to *name* as a variable or goal or signal, but there isn't one of this name in the current network.

Error - No Targets to Tolerance

If there are no goals or fmerit and the +TAR option has not been used, the use of tolerance commands do not make any sense.

Error - No Variables to Optimize

The `optimize` command was used but there are no target variables (range variables) in the current network. Optimization requires something to be varied, see chapter 6.

Error - No Variables to Tolerance

The `tolerance` command was used but there are no variates in the current network. Tolerancing requires variates to be defined, see chapter 8.

Error - Node List Unconsecutive

The node list for component connection may appear after the component name (or :sub-model-name) or after the parameters, but it must appear together.

Error - Node (*node-name*) Not Available For OPCYCLE

Only nodes at port interfaces and at nonlinear models are available for OPCYCLE in the current version of AGILE.

Error - Node (*node-name*) Requested More Than Once in Subnet (*net-name*)

In using a sub-network, each node requested from it must be unique, you can however connect one or more of them to the same node in the current network.

Error - NOISE_DATA Format for two-port subnets only

The NOISE_DATA form was used in a non two-port circuit, this is illegal.

Error - NOISE_LESS Keyword must appear alone This keyword appears alone on the line.

Error - NOISE_MATRIX Must Have #ports-squared terms

The correct number of complex-valued NOISE_MATRIX elements is the number of ports squared.

Error - Non-Frequency Unit in Data Statement Illegal

AGILE allows tabulated data to specify a frequency unit in the data statement, but the wrong kind of unit was used. Check your NDL, in the circuit and line given.

Error - Nonlinear Component Argument (*arg-name*) is Freq-Dep

AGILE allows *linear* component arguements to take on frequency dependent values, but the current version does not allow teh arguments of nonlinear components to be frequency dependent.

Error - Nonlinear Component(s) not Allowed for Noise Calculations

Error - Nonlinear Component(s) not Allowed in Port Terminations

Noise dependent outputs (i.e., NFIG) cannot be computed for nonlinear circuits, likewise, nonlinear circuits cannot be port termination networks.

Error - NULL string as node name

Node names must be be at least one character, the input: `RES 1 $ (r=9)` will cause this to happen.

Error - NUM Must be >= #

Several components use NUM to represent a number (like # coupled lines) this must be greater than that specified.

Error - Number Expected as Frequency Specification

The data statement must start with a real-valued number.

Error - Number of Data Items Wrong for TAB data

The number of data items on tabulated data NDL lines determines the number of ports that this data represents, it must conform to the expression: $2*N^2+1$ (see section on tabulated data).

Error - Number of EXP steps in Freq Statement <= 0

AGILE must have a step value greater than 0 for EXP type frequency statements.

Error - Number of F-Points Must be > 0

The third part of the harmonics statement specifies the number of fundamental tones, this must be > 0.

Error - Number of grid points must be >= 2

The command optimize grid *integer* requires that the integer be greater than or equal to 2, you tried with less.

Error - Number of Nodes for Component Does not match Connection

The number of connections for the component used on the NDL line that will be displayed does not match that in the connection.

Error - Number of Nodes for Port Termination Must be 2

The number of connections for the component used as the port termination must be 2.

Error - OFF Must be < difference of Widths W1/W2

Physical constraints demand this restriction for the MSSTEP component.

Error - One W to large in (*component*). Will gen higher order modes

Based on the width of the lines and frequencies of analysis, a pure TEM mode is not assured for this component.

Error - Only W1/W2/... can be Specified for *component*

The *component* requires that widths 1-N be defined, others are illegal.

Error -Output (*name*) Requires Noise Analysis

Requests to LIST or PLOT a noise-related output dynamically in the LIST/ PLOT command requires that noise analysis has already been done. At least one noise-related output should be placed in the network to trigger AGILE to do noise analysis.

Error - Output (*name*) Requires One Node Definition

Error - Output (*name*) Requires One Port Definition Error - Output (*name*) Requires Two Different Ports

Error - Output (*name*) Requires Two Port Definition

The output variable *name* was used improperly. Depending on which output it is either one or two port/node descriptors are required.

Error - Overflow in Expression Parsing - YYMAXDEPTH

The current version of AGILE provides for a fixed (but large) "depth" of expression parsing, this was exceeded. Please break the expression into smaller pieces using additional variables.

Error - PASS Expr Cannot be Complex-valued

The evaluation of the PASS statement resulted in a complex value, this is not allowed.

Error - PL Cannot be Specified with CF/EL

The component defined can only use either CF/EL or PL but not both as arguments.

Error - PL Required With ER Argument

The component defined requires PL if ER is used.

Error - Point # *integer* does not exist for curve # *integer*

The index of the desired point on the curve for labelling does not exist. Either *integer* is less than one or greater than the number of points for the number curve in the selected display.

Error - Port Node (*node-name*) Unconnected to Network in Net (*net-name*)

In the named network, the nodes used for port definition are not connected using any connection statements. AGILE enforces that the nodes used for port definition be mentioned as part of the connection(s), otherwise the port would be completely un-connected to the network, rendering that port disconnected.

Error - Positive Port Nodes Not Unique in Net (*net-name*)

AGILE enforces that all "positive" nodes for each port be unique. Please check chapter 2, PORT DEFINITION STATEMENTS, for more extensive information.

Error - Q must be >= 0

The Q factor for SRLC/PRLC components must be greater or equal to zero.

Error - Real Argument Required for Function (*func-name*)

The named function requires a real-valued argument, the current argument is complex.

Error - Recursive NDL Description Between nets (*net-name*) and (*net-name*)

The two networks name contain either direct or indirect references to each other as sub-models, this cannot exist.

Error - Referenced NDL Connection (*comp-name*) in Net (*net-name*) Undefined The component *comp-name* was used in as a referenced connection but was not defined in network *net-name*.

Error - Requested Node (*node-name*) Not In Network (*net-name*)

The node *node-name* was requested in the sub-model reference, but there is no such node in that network of that name.

Error - S Index Must be 0 thru NUM - 1

Error - SEP Index Must be 0 thru NUM - 1

At least one SEP index of the named component wrong. Check chapter 4 for correct component specifications.

Error - Separation S*number* is <= 0

Error - Separation SEP*number* is <= 0

The evaluation for the separation resulted in a negative value, this is not allowed.

Error - Separation S*number* is Missing

Error - Separation SEP*number* is Missing

The named separation term is missing from the component parameter list (remember that SEP0 implies all).

Error - SET TUNER ?*command*? Unknown

The *command* portion of the SET TUNER command is unrecognized.

Error - Signal Domain is not TIME or FREQ

As part of the signal definition, the domain must be specified as either TIME or FREQ, this was not the case.

Error - Signal Name Cannot be (DIRECT)

This is an odd one, since AGILE uses the word DIRECT to represent something internally in the program, the name DIRECT is not allowed, please change it to another name.

Error - Signal (*name*) Applied to Port *#port-name* is not defined

The signal *name* is used as a signal reference in the port statement, but signal has not been defined.

Error - Signal (*name*) has Unbindable Expression

The signal *name*'s expression uses variables which are not defined.

Error - Signal (*name*) is Analysis Dependent

The named signal uses either outputs or variables dependent on outputs in its expression, this is not allowed.

Error - Signal Type Not VOLTAGE, CURRENT, MW, or DBM

The third token in the signal statement defines the type of signal, this must be one of the above.

Error - (SL) Arg Must be 1 or 2 in (TUNER) model

The Slug-Length (SL) argument must be either 1 or 2.

Error - Size for # of (*object*) must be >0, size not changed

The `SET MAXIMUM ...` command was used to run-time adjust the maximum size of *object*, for each *object* type, however, there are minimums that must be adhered to.

Error - Step Expression is Complex-Valued

The expression in the step part of a range variable evaluates to a complex value, this is not allowed. Simple example:
`X1 = 5 [ 0 : 10 : (3,4) ]`

Error - Source Bias (VS) required unless DC balancing

The source bias for the FET model may be established either by the VS argument or gotten from the DC component of the harmonic balance. If the VS argument is not present and the "0th" (DC) harmonic is not present, this error will occur.

Error - SubNet Argument (*arg-name*) Must Have Value, Does Not Have Default The sub-model argument named does not have a default value and its value was not specified in its invocation.

Error - Syntax Error in Expression

The general error (i.e., mismatched parentheses, missing operators) in expression parsing has occurred. The line is displayed.

Error - TABULATED Network (*net-name*) with no Data

The network *net-name* was declared as TABULATED, but no data statements are within.

Error - Target (*var-name*) cannot be Freq-Dep

Error - Target (*var-name*) cannot be Time-Dep

The variable *var-name* must be non frequency or time dependent in the contect used (i.e., CONTOUR command).

Error - Temperature Expression Complex-Valued in Net (*net-name*)

The value of temperature must be real-valued, check your variable types in the temperature statement.

Error - TH larger than HT in (*component*)

The value evaluated for the TH parameter is greater than the HT parameter in the named circuit component function. This situation is physically unrealizable and therefore cannot be analyzed.

Error - TH must be Specified when RRES is Used

Metallization thickness must be specified when relative resistivity is.

Error - The View Point Chosen is Illegal (mag>1)

For viewing three-dimensional graphics, AGILE allows changing the point from which the data is viewed. However, this point must have a "polar-radius" greater or equal to one. See chapter 3 for more information.

Error - Time Domain Signal (*name*) Cannot be Freq. Dep.

Signals which are defined in the time domain cannot have frequency dependence, either directly or indirectly through the specical variable `FREQ`.

Error - Time Domain Signals Must be VOLTAGE or CURRENT

Signals defined as time domain must be either voltages or current types, for example, you cannot specify MW over time!

Error - Time Part of S-List Expression Must be Real-Valued

In a signal definition for time domain lists, the time (first part) point must evaluate to a real-value.

Error - Time-Dependent (*var-name*) cannot be Displayed VS (*var-name*)

The current version of AGILE does not allow LISTs/PLOTs versus a range variable.

Error - Time-Domain Signal (*sig-name*) Cannot be Freq. Dep.

Signals which are declared as time domain cannot use expressions which are frequency dependent.

Error - Tolerance Data Unavailable

Remember, you can only do the tolerance plot/list commands *after* either a tolerance analysis or tolerance design (not tolerance sensitivity).

Error - Too Many Arguments for Function

The function has more arguments than allowed, please check function reference for correct function usage.

Error - Too Many Expressions in Function - MAX_ARGS_PER_FUNC

The function has more arguments than allowed, please check function reference for correct function usage.

Error - TPERIODFAC Must be >= 1

Error - TSAMPLEFAC Must be >= 2

The factors related to the frequency to time conversion must be set appropriately.

Error - TUNE 'Write...' Button Invalid Now

The X Windows version of AGILE has a write button to write the adjusted value back to the NDL description, this button is not valid unless the circuit, var name, etc. are set.

Error - TURN values must be > 0.0

The turn numbers for the transformer model must be > 0.0

Error - Undefined Port Requested in Output (*name*)

The output variable *name* uses a port which is not defined

Error - Undefined Variable (*name*) Referenced in Net (*net-name*)

The variable *name* was used in the network *net-name* but was not defined, either locally or globally.

Error - Unexpected error in alloc'g data in *NAME* model

This message occurs whenever a model needs to dynamically allocate data but it cannot. Perhaps running fewer other programs, increasing swap space, etc. may solve this problem.

Error - Unexpected error reading file

A system file read error occurred while trying to read the file.

Error - Unexpected Token Type (*type*) in Expression

Basically, there is an item in the expression which is illegal, for example: `x = hz*2` will cause this error since `hz` is a unit.

Error - Unit Required for Argument (*arg-name*)

The model used requires that *arg-name* have a unit specifier, it was missing.

Error - Unknown Function: *name*

The token *name* was used as a function name, but this function is not known.

Error - Unrecognized Name ... Token Type (*type*)

Basically, there is a special token used where a name was expected.

Error - UnRecognized NDL Line ... Starts with Token Type (*type*)

The statement being inserted is completely un-recognized. Sometimes even a seemingly simple NDL line will cause this, consider: HZ = 7, which will generate this message, where item would be UNIT. This is due to the fact that HZ is an unit, not a variable; the form unit=expression is unknown to AGILE.

Please check your input.

Error - Upper Expression is Complex-Valued

The second expression (upper) in a range variable evaluates to a complex value, this is not allowed.

Error - User Abort File Read What more can we say?

Error - Variables on +TAR list do not exist

The +TAR option was used in a tolerance command, but at least one of the variables specified on the list does not exist in the current network.

Error - W Index Must be 0 thru NUM

Many of the models use NUM to represent a number of things, in this case the argument W must be within this range.

Error - W too Large in (*model-name*). Will gen higher order modes

The width of a line in the model *model-name* is too large at the requested frequency of analysis, higher order modes will result.

Error - W1/W2/... Must be Specified for *model-name*

One or more of the widths for *model-name* are missing. Remember that W0 implies all widths.

Error - W2 Must be < W1

For the model listed, this must be the case.

Error - While Inserting Line

An unexpected error occurred while trying to insert an NDL line, this should never happen but AGILE detects it.

Error - Width of line too small

A transmission line width is negative or too small for computation.

Error - Width Ratio in MSSTEP too large...will gen. higher order modes

The MSSTEP component width ratio is large enough to cause moding, AGILE does not analyze this (use EM simulators).

Error - Width W*number* is <= 0.0

A transmission line width is negative or too small for computation.

Error - Width W*number* is Missing

A transmission line width is not specified, remember that W0 specifies all widths.

Error - Wrong Index Entered for File Selection

AGILE asked you to enter an integer to selction which file you wanted selected, you missed the answer. See section on "rr/restore" commands.

Error - Wrong Number of Arguments for Function (*func-name*)

The function *func-name* was used improperly, please check the reference.

Error - Wrong Number of Connections for Component

The named component or NDL line shown has the wrong number of connections. That is the number of connection nodes specified does not match that of the component model.

Error - Wrong Number of Data Points

The number of data points specified on an NDL line must conform to the expression: $1+(2*N)^2$ where N is the number of ports. Please refer to the discussion of tabulated networks for details.

Error - Wrong Number of Expressions for Goal

Each goal-type requires the correct number of expressions for proper definition. Check chapter 6 for details.

Error - Wrong Token Type (*type*) in Node List

The token type *type* appears in the NDL line reserved for node names (integers or $name's).

Error - Wrong Unit for Argument (*arg-name*)

The component argument *arg-name* is used with the wrong kind of unit (for example a width argument with units of frequency).

Error - You Don't Have Data to Display

This message results when using the `display list/plot` commands when no data has been generated (with a list/plot var [vs var] command).

Error - You Have No Display To *something*

There is no display, but you are trying to change *something*. You must create a display before changing it.

Error - You have no line to RLINE on

The rline command was used when there wasn't any line to replace.

Warning - (*name*) lower bound relaxed

Warning - (*name*) upper bound relaxed

This message occurs when the current value for a range variable is close to one of its bounds. A general recommendation: is to not start variables right against their edges.

Warning - Approximation in Matrix Reduction

As AGILE was analyzing the circuit, certain approximations needed to be done to complete analysis, you may have errors in your circuit, or you are trying to analyze very "idealized" circuits.

Warning - Arguments for Function (*func-name*) <= 0.0

Certain functions (i.e., DB) require their arguments to be greater than zero, a warning is issued (but not an error, AGILE continues) when this is not the case.

Warning - Data on *axis* has small range

Where *axis* will be one of: XAXIS, YAXIS or ZAXIS. This message informs you that the data being presented is very flat.

Warning - File Name and Network Name Inconsistent

This only happens when a "system" editor is used to prepare a file called netname.net and there exists a network keyword definition statement is this file specifying a name other than netname. This might happen for instance when you use a "system" file copy command to create a new network from an old one, edit (system-edit) the newly copied file but forget to change the name, and use AGILE to restore it. The network name in NDL is now the same as that of the original file, using a save command (with yes) will overwrite the original definition. This can never occur if AGILE is used for all editing of networks.

Warning - Matrix Ill-Conditioned for Network (*net-name*)

As AGILE was analyzing the circuit, certain approximations needed to be done to complete analysis, you may have errors (disconnects) in your circuit, or you are trying to analyze very "idealized" circuits.

Warning - Name *name* truncated

NDL allows a fixed number of letters and numbers for names (currently 15), if this is exceeded, AGILE automatically truncates to 15 characters and issues this warning.

Warning - Network Analysis Error Occurred at that point

An error in network analysis occurred during an TUNE command.

Warning - No Points Plotted for curve # *integer*

Either the X-scale or Y-scale is set such that none of the data in the curve number *integer* appears on the graph.

Warning - Precision in [*double*] complex division

An NDL complex division is being done where the magnitude of divisor is < 1e25.

Warning - Too Many Nets In MOTIF Selection Box

The number of currently resident networks exceeds that allowed in the AGILE implementation of the X Windows selection box.

Warning - User Interrupt of Nonlinear Analysis

The user interruptted analysis (with a cntrl-C).

11.4 RESERVED WORD NDL SUMMARY

This section simply lists the result of using the `show language` command of AGILE (as was promised in chapter 3). The "tokens" (language elements) listed here are the only ones which are recognized by AGILE. The capabilities of AGILE are continually growing; this is achieved in one way by addition of functions, components, etc. Therefore, the result you get may actually contain additional "goodies".

```
******************** AGILE Language Summary ********************
           *************** Keywords ***************
 NETWORK     FREQUENCIES PORTS       ENVIRONMENT COMPONENTS  CONNECTIONS
 TABULATED   DATA        OUTPUTS     GOALS       END
 SIGNAL      PASS        GLOBAL      TEMPERATURE HARMONICS
           *************** Units ***************
```

```
HZ    -- unit of FREQUENCY   -- to_mks = (1)
KHZ   -- unit of FREQUENCY   -- to_mks = (1000)
MHZ   -- unit of FREQUENCY   -- to_mks = (1e+06)
GHZ   -- unit of FREQUENCY   -- to_mks = (1e+09)
PH    -- unit of INDUCTANCE  -- to_mks = (1e-12)
NH    -- unit of INDUCTANCE  -- to_mks = (1e-09)
UH    -- unit of INDUCTANCE  -- to_mks = (1e-06)
MH    -- unit of INDUCTANCE  -- to_mks = (0.001)
H     -- unit of INDUCTANCE  -- to_mks = (1)
PF    -- unit of CAPACITANCE -- to_mks = (1e-12)
NF    -- unit of CAPACITANCE -- to_mks = (1e-09)
UF    -- unit of CAPACITANCE -- to_mks = (1e-06)
MF    -- unit of CAPACITANCE -- to_mks = (0.001)
F     -- unit of CAPACITANCE -- to_mks = (1)
OHMS  -- unit of RESISTANCE  -- to_mks = (1)
KOHMS -- unit of RESISTANCE  -- to_mks = (1000)
MOHMS -- unit of RESISTANCE  -- to_mks = (1e+06)
UMHOS -- unit of CONDUCTANCE -- to_mks = (1e-06)
MMHOS -- unit of CONDUCTANCE -- to_mks = (0.001)
MHOS  -- unit of CONDUCTANCE -- to_mks = (1)
DEG   -- unit of ANGLE       -- to_mks = (0.0174533)
RAD   -- unit of ANGLE       -- to_mks = (1)
ASRM  -- unit of DISTANCE    -- to_mks = (1e-10)
UIN   -- unit of DISTANCE    -- to_mks = (2.54e-08)
MILS  -- unit of DISTANCE    -- to_mks = (2.54e-05)
INCH  -- unit of DISTANCE    -- to_mks = (0.0254)
FT    -- unit of DISTANCE    -- to_mks = (0.3048)
UM    -- unit of DISTANCE    -- to_mks = (1e-06)
MM    -- unit of DISTANCE    -- to_mks = (0.001)
CM    -- unit of DISTANCE    -- to_mks = (0.01)
M     -- unit of DISTANCE    -- to_mks = (1)
NM    -- unit of DISTANCE    -- to_mks = (1e-09)
DB/M  -- unit of LOSS        -- to_mks = (1)
DB/WL -- unit of ATTENUATION -- to_mks = (1)
SEC   -- unit of TIME        -- to_mks = (1)
MSEC  -- unit of TIME        -- to_mks = (0.001)
USEC  -- unit of TIME        -- to_mks = (1e-06)
NSEC  -- unit of TIME        -- to_mks = (1e-09)
PSEC  -- unit of TIME        -- to_mks = (1e-12)
          *************** Outputs ***************
Y         S         Z         VGAIN     IGAIN     PGAIN     ZIN
YIN       RC        GDELAY    VSWR      STAB      STABL     TFACTOR
GMAX      MATCH     GAINSLOPE NFIG      NFIGMIN   YNOPT     RCNOPT
RNEQV     VSPECTRUM ISPECTRUM PSPECTRUM VSIGNAL   ISIGNAL
     *************** Goal Functions ***************
SUM         MAXIMUM         UPPERLOWER
SUMNORM     MAXIMUMNORM     UPPERLOWERNORM
SUMSP       MAXIMUMSP       UPPERLOWERSP
SUMNORMSP   MAXIMUMNORMSP   UPPERLOWERNORMSP
****************** END of Language Summary ****************
```

11.5 FUNCTION REFERENCE SUMMARY

The following is a list of functions can be obtained in the application using the show function button on the Set/Show tab.

```
Function    Description
--------    -----------
+ - * /     Addition, Subtraction, Times, Divide
MIN  MAX    Minimum and Maximum are binary operators
+ -         Unary Plus and Minus
  **        Power
< > <= >= ==  Relational Operators
AND         Logical And
OR          Logical Or
if(x,y,z)   if x > 0 then value is expr. y, else expr. z
sqrt(x)     Real or Complex Sqrt
fselect(x,y) Real or Complex select freq-point x from y
abs(x)      Absolute Value if arg real, mag. if arg complex
mag(x)      Same as above
db(x)       dB of Complex argument
ptor(x)     polar to rect format conv. of complex arg
rtop(x)     rect to polar format conv. of complex arg
floor(x)    next lowest integer function
dirac(x)    Sampling function =1 when |x| < 1e-5, else =0
unit(x)     Unit Step function =1 when x >= 0, else =0
phase(x)    phase +- 180 of complex argument
aphase(x)   phase 0-360 of real argument
real(x)     real-part of complex argument
imag(x)     imag-part of complex argument
exp(x)      Natural Anti-Logorithm
log(x)      Natural Logorithm
log10(x)    Based 10 Logorithm
cos(x)
sin(x)      trig functions of real arg
tan(x)
ccos(x)
csin(x)     trig functions of complex arg
cosh(x)
sinh(x)     hyperbolic trig functions of real arg
tanh(x)
arccos(x)
arcsin(x)   Arc trig functions of real arg
arctan(x)
acosh(x)
asinh(x)    Arc hyperbolic trig functions of real arg
atanh(x)
ccosh(x)
csinh(x)    Complex hyperbolic trig functions
ctanh(x)
```

11.6 OUTPUT VARIABLE REFERENCES

Table 1: Linear Network Outputs

Name	Represents	Format	Examples
S	S-parameters	(real, imag)	S21 S$in.$out
Y	Y-parameters	(real, imag)	Y23 Y34.67
Z	Z-parameters	(real, imag)	z12 Z12.3

Table 2: Port Termination Dependent Outputs

Name	Type	Represents	Format	Examples
VGAIN	T	Voltage Gain	(real, imag)	Vgain21
IGAIN	T	Current Gain	(real, imag)	Igain23 IGAIN34.67
PGAIN	T	Power Gain	(dB, Angle)	Pgain12 PGAIN12.3
ZIN	R	Input Impedance	(real, imag)	Zin1 Zin14.
YIN	R	Input Admittance	(real, imag)	Yin1 Yin7
RC	R	Reflection-Coefficient	(real, imag)	RC1 rc3
VSWR	R	Voltage Standing Wave Ratio	(real)	VSWR1 VSWR3 vswr4
STAB	T	Rollet's Stability Factor	(real)	STAB12 Stab34
STABL	T	Linville's Stability Factor	(real)	StabL21
TFACTOR	T	Ladbrooke's Tunability Factor	(real)	TfactoroutIN
GMAX	T	Maximum Available Gain	(dB)	Gmax12
MATCH	T	Conjugate match at port	(real, imag)	Match12 MATCH10.2

GDELAY	T	Group Delay	(real)	Gdelay12
GAINSLOPE	T	Gain Slope	(real)	Gainslope3.4

Table 3: Noise Analysis Outputs

Name	Type	Represents	Format	Examples
NFIG	T	Noise Figure	(dB)	nfig21
NFIGMIN	T	Minimum Noise Figure	(dB)	NfigMin23 nfigmin4.67
YNOPT	T	Optimal Noise Input Admittance	(real, imag)	Ynopt12
RCNOPT	T	Optimal Noise Input Refl.	(real, imag)	RcNopt21 rcnopt31
RNEQV	T	Eqv. Noise Resistance	(real)	RNEQV1 Rneqv$in

Table 4: Signal Outputs, linear or non-linear

Name	Type	Represents	Format	Examples
VSPECTRUM	R	Voltage Spectrum	(real, imag)	VSpectrum5
ISPECTRUM	R	Current Spectrum	(real, imag)	ISpectrum5
VSIGNAL	R	Voltage Signal	(real)	VSignal$output
ISIGNAL	R	Current Signal	(real)	ISignal4

11.7 BRIEF NDL LANGUAGE REFERENCE SUMMARY

Variable Assignment (see also section 3.5 on tracing)

```
var-name = expr
var-name = number [ expr : expr ]
var-name = number [ expr : expr : expr ]
var-name = number { dist-name expr }
var-name = number [ expr : expr ] { dist-name expr }
var-name = number [ expr : expr : expr ] { dist-name expr }
```

Frequency / Harmonic Definition

```
number f-unit
number number number f-unit
number number number EXP f-unit
HARMONICS f-unit 1 freq-base h-list
HARMONICS f-unit #(>=2) freq-base-list order lower-band-limit upper-band-limit
```

Port Definition

```
#port-name positive-node negative-node
#port-name positive-node negative-node signal-name
#port-name positive-node negative-node port-termination
#port-name positive-node negative-node signal-name port-termination
```

Signal Definition

```
SIGNAL signal-name sig-type sig-domain expr OR s-list
   Where:
           sig-type: VOLTAGE/CURRENT/MW/DBM
           sig_domain: FREQ/TIME
           s-list: {expr1 : expr2} ...
```

Component Definition / Connection (see also section 3.5 on tracing)

```
ref-name = comp-def connection-list
   Where:
           connection-list: list of integers or $names
           comp-def:  model-name(comp-args) OR
                      :sub-name(comp-args) {sub-node-list}
           comp-args: list of arg-name = expr [unit optional], comma or space separated
ref-name connection-list
```

Environment

```
env-name : comp-arg-list
```

11.8 CONFIDENCE TABLES

The following tables are standard confidence limit tables provided for your convenience. The tables are symmetric around 50% yield, that is the number of samples needed for a 30% true yield is the same as for 70%. More extensive data may be found in statistics texts.

95 % Confidence

True Yield ↓	ERROR									
	± 1%	± 2%	± 3%	± 4%	± 5%	± 6%	± 7%	± 8%	± 9%	+ 10%
50%	9604	2401	1067	600	384	266	196	150	118	96

60%	9219	2304	1024	576	368	256	188	144	113	92
70%	8067	2016	896	504	322	224	164	126	99	80
80%	6146	1536	682	384	245	170	125	96	75	61
90%	3457	864	384	216	138	96	70	54	42	34

99 % Confidence

True Yield ↓	ERROR									
	± 1%	± 2%	± 3%	± 4%	± 5%	± 6%	± 7%	± 8%	± 9%	+ 10%
50%	16576	4144	1841	1036	663	460	338	259	204	165
60%	15913	3978	1768	994	636	442	324	248	196	159
70%	13924	3481	1547	870	556	386	284	217	171	139
80%	10609	2652	1178	663	424	294	216	165	130	106
90%	5967	1491	663	372	238	165	121	93	73	59