Agilent Technologies

Measurement Expressions

September 2004

Notice

The information contained in this document is subject to change without notice.

Agilent Technologies makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Agilent Technologies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms that apply to this software product is available upon request from your Agilent Technologies representative.

Restricted Rights Legend

Use, duplication or disclosure by the U. S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DoD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

Agilent Technologies
395 Page Mill Road
Palo Alto, CA 94304 U.S.A.

Acknowledgments

Mentor Graphics is a trademark of Mentor Graphics Corporation in the U.S. and other countries.

Microsoft®, Windows®, MS Windows®, Windows NT®, and MS-DOS® are U.S. registered trademarks of Microsoft Corporation.

Pentium® is a U.S. registered trademark of Intel Corporation.

PostScript® and Acrobat® are trademarks of Adobe Systems Incorporated.

UNIX® is a registered trademark of the Open Group.

Java™ is a U.S. trademark of Sun Microsystems, Inc.

Contents

Chapter 1: Introduction to Measurement Expressions

This document describes the measurement expressions that are available for use with several Agilent EEsof EDA products. For a complete list of available measurement functions, refer to the *Alphabetical Listing of Measurement Functions* in Chapter 2 or consult the index.

Measurement expressions are equations that are evaluated during simulation post processing. They can be entered into the program using various methods, depending on which product you are using. Unlike the expressions described in the *Simulator Expressions* documentation, these expressions are evaluated after a simulation has completed, not before the simulation is run. Measurement expressions can also be easily used in a Data Display. For more information on entering equations in a data display, refer to the *Data Display* documentation.

Although there is some overlap among many of the more commonly used functions, measurement expressions and simulator expressions are derived from separate sources, evaluated at different times, and can have subtle differences in their usages. Thus, these two types of expressions need to be considered separately. For an overview of how measurement expressions are evaluated, refer to Figure 1-1.

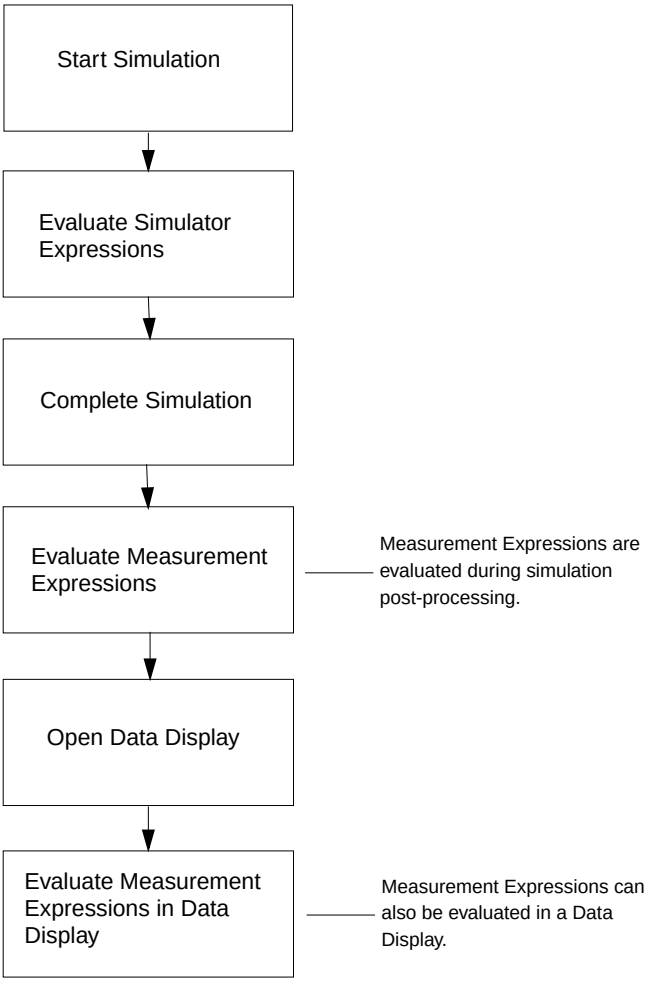


Figure 1-1. How Measurement Expressions are Evaluated.

Within this document you will find information on:

- "Measurement Expressions Syntax" on page 1-3
- "Manipulating Simulation Data with Expressions" on page 1-7
- Information on working with different types of data.
- Information specific to entering simulator expressions in your particular product.

You will also find a complete list of functions that can be used as measurement expressions individually, or combined together as a nested expression. These functions have been separated into libraries and are listed in alphabetical order within each library. The functions available include:

- Chapter 3, Circuit Budget Functions
- Chapter 4, Circuit Envelope Functions
- Chapter 5, Data Access Functions
- Chapter 6, Harmonic Balance Functions
- Chapter 7, Math Functions
- Chapter 8, Signal Processing Functions
- Chapter 9, S-parameter Analysis Functions
- Chapter 10, Statistical Analysis Functions
- Chapter 11, Transient Analysis Functions

For a complete list of all functions provided in this document, refer to the Alphabetical Listing of Measurement Expressions in Chapter 2 or consult the index.

Measurement Expressions Syntax

Use the following guidelines when creating measurement expressions:

- Measurement expressions are based on the mathematical syntax in Application Extension Language (AEL).
- Function names, variable names, and constant names are all case sensitive in measurement expressions.
- Use commas to separate arguments.

- White space between arguments is acceptable.

Case Sensitivity

All variable names, functions names, and equation names are case sensitive in measurement expressions.

Variable Names

Variables produced by the simulator can be referenced in equations with various degrees of rigidity. In general a variable is defined as:

DatasetName.AnalysisName.AnalysisType.CircuitPath.VariableName

By default, in the Data Display window, a variable is commonly referenced as:

DatasetName..VariableName

where the double dot ".." indicates that the variable is unique in this dataset. If a variable is referenced without a dataset name, then it is assumed to be in the current default dataset.

When the results of several analyses are in a dataset, it becomes necessary to specify the analysis name with the variable name. The double dot can always be used to pad a variable name instead of specifying the complete name.

In most cases a dataset contains results from a single analysis only, and so the variable name alone is sufficient. The most common use of the double dot is when it is desired to tie a variable to a dataset other than the default dataset.

Built-in Constants

The following constants can be used in measurement expressions.

Table 1-1. Built-in Constants

Constant	Description	Value
PI (also pi)	π	3.1415926535898
e	Euler's constant	2.718281822
ln10	natural log of 10	2.302585093
boltzmann	Boltzmann's constant	1.380658e–23 J/K
qelectron	electron charge	1.60217733e–19 C
planck	Planck's constant	6.6260755e-34 J*s
c0	Speed of light in free space	2.99792e+08 m/s
e0	Permittivity of free space	8.85419e–12 F/m
u0	Permeability of free space	12.5664e–07 H/m
i, j	sqrt(–1)	1i

Operator Precedence

Measurement expressions are evaluated from left to right, unless there are parentheses. Operators are listed from higher to lower precedence. Operators on the same line have the same precedence. For example, a+b*c means a+(b*c), because * has a higher precedence than +. Similarly, a+b-c means (a+b)–c, because + and – have the same precedence (and because + is left-associative).

The operators !, &&, and || work with the logical values. The operands are tested for the values TRUE and FALSE, and the result of the operation is either TRUE or FALSE. In AEL a logical test of a value is TRUE for non-zero numbers or strings with non-zero length, and FALSE for 0.0 (real), 0 (integer), NULL or empty strings. Note that the right hand operand of && is only evaluated if the left hand operand tests TRUE, and the right hand operand of | | is only evaluated if the left hand operand tests FALSE.

The operators >=, <=, >, <, ==, != , AND, OR, EQUALS, and NOT EQUALS also produce logical results, producing a logical TRUE or FALSE upon comparing the values of two expressions. These operators are most often used to compare two real numbers or integers. These operators operate differently in AEL than C with string

expressions in that they actually perform the equivalent of *strcmp()* between the first and second operands, and test the return value against 0 using the specified operator.

Table 1-2. Operator Precedence

Operator	Name	Example
( )	function call, matrix indexer	foo(expr_list) X(expr,expr)
[ ]	sweep indexer, sweep generator	X[expr_list] [expr_list]
{ }	matrix generator	{expr_list}
**	exponentiation	expr**expr
!	not	!expr
* / .* ./	multiply divide element-wise multiply element-wise divide	expr * expr expr / expr expr .* expr expr ./ expr
+ -	add subtract	expr + expr expr - expr
::	sequence operator wildcard	exp::expr::expr start::inc::stop ::
< <= > >=	less than less than or equal to greater than greater than or equal to	expr < expr expr <= expr expr > expr expr >= expr
==, EQUALS !=, NOTEQUALS	equal not equal	expr == expr expr != expr
&& AND	logical and	expr && expr
|| OR.	logical or	expr || expr

Conditional Expressions

The if-then-else construct provides an easy way to apply a condition on a per-element basis over a complete multidimensional variable. It has the following syntax:

A = **if** (*condition*) **then** *true_expression* **else** *false_expression*

Condition, true_expression, and false_expression are any valid expressions. The dimensionality and number of points in these expressions follow the same matching conditions required for the basic operators.

Multiple nested if-then-else constructs can also be used:

A = **if** (*condition*) **then** *true_expression* **else*if*** *(condition2)* **then** *true_expression* **else** *false_expression*

The type of the result depends on the type of the true and false expressions. The size of the result depends on the size of the condition, the true expression, and the false expression.

Examples

The following information shows several examples of conditional expressions using various operators.

boolV1=1
boolV2=1

eqOp=if (boolV1 == 1) then 1 else 0 eqOp returns 1
eqOp1=if (boolV1 EQUALS 1) then 1 else 0 eqOp1 returns 1

notEqOp=if (boolV1 != 1) then 1 else 0 notEqOp returns 1
notEqOp1=if (boolV1 NOTEQUALS 1) then 1 else 0 notEqOp1 returns 1

andOp=if (boolV1 == 1 AND boolV2 == 1) then 1 else 0 andOp returns 1
andOp1=if (boolV1 == 1 && boolV2 == 1) then 1 else 0 andOp returns 1

orOp=if (boolV1 == 1 OR boolV2 == 1) then 1 else 0 orOp returns 1
orOp1=if (boolV1 == 1 || boolV2 == 1) then 1 else 0 orOp returns 1

Manipulating Simulation Data with Expressions

Expressions defined in this documentation are designed to manipulate data produced by the simulator. Expressions may reference any simulation output, and may be placed in a Data Display window. For details on using and applying simulation data with measurement expressions, refer to *Applying Measurements* in Chapter 2 of the *Circuit Simulation* manual.

Simulation Data

The expressions package has inherent support for two main simulation data features. First, simulation data are normally multidimensional. Each sweep introduces a dimension. All operators and relevant functions are designed to apply themselves automatically over a multidimensional simulation output. Second, the independent (swept) variable is associated with the data (for example, S-parameter data). This

independent is propagated through expressions, so that the results of equations are automatically plotted or listed against the relevant swept variable.

Measurements and Expressions

Measurements are evaluated after a simulation is run and the results are stored in the dataset. The tag *meqn_xxx* (where *xxx* is a number) is placed at the beginning of all measurement results, to distinguish those results from data produced directly by the simulator.

Complex measurement equations are available for both circuit and signal processing simulations. Underlying a measurement is the same generic equations handler that is available in the Data Display window. Consequently, simulation results can be referenced directly, and the expression syntax is identical. All operators and almost all functions are available.

The expression used in an optimization goal or a yield specification is a measurement expression. It may reference any other measurement on the schematic.

Generating Data

The simulator produces scalars and matrices. When a sweep is being performed, the sweep can produce scalars and matrices as a function of a set of swept variables. It is also possible to generate data by using expressions. Two operators can be used to do this. The first is the sweep generator "[ ]", and the second is the matrix generator "{ }". These operators can be combined in various ways to produce swept scalars and matrices. The data can then be used in the normal way in other expressions. The operators can also be used to concatenate existing data, which can be very useful when combined with the indexing operators.

Sweep Generator Examples

Several sweep generator examples are given below:

```
arr1=[0,1,2,3,4,5]   creates an array of six values
arr2=[0::1::5]   generates the above data using the sequence operator

arrCat=[arr1,arr2]   concatenates the two arrays

sunArr1=[arr1[3::5],arr1[0::2]]   re-arranges the existing data in a different order

z=0*[1::50]
vpadded=[arr1,z]   creates a zero-padded array
```

Matrix Generator Examples

Some examples of the matrix builds operator are given below:

v1={1,2,3,4,5} five-element vector
v2={1::5} five-element vector using the sequence operator

v3={{1,0}, {0,1}} 2X2 identity matrix

Simple Sweeps and Using "[ ]"

Parameter sweeps are commonly used in simulations to generate, for example, a frequency response or a set of DC IV characteristics. The simulator always attaches the swept variable to the actual data (the data often being called the *attached independent* in equations).

Often after performing a swept analysis we want to look at a single sweep point or a group of points. The sweep indexer "[ ]" can be used to do this. The sweep indexer is zero offset, meaning that the first sweep point is accessed as index 0. A sweep of *n* points can be accessed by means of an index that runs from 0 to *n*–1. Also, the what() function can be useful in indexing sweeps. Use what() to find out how many sweep points there are, and then use an appropriate index. Indexing out of range yields an invalid result.

The sequence operator can also be used to index into a subsection of a sweep. Given a parameter X, a subsection of X may be indexed as

a=X[start::increment::stop]

Because increment defaults to one,

a=X[start::stop]

is equivalent to

a=X[start::1::stop]

The "::" operator alone is the wildcard operator, so that X and X[::] are equivalent. Indexing can similarly be applied to multidimensional data. As will be shown later, an index may be applied in each dimension.

S-parameters and Matrices

As described above, the sweep indexer "[ ]" is used to index into a sweep. However, the simulator can produce a swept matrix, as when an S-parameter analysis is performed

over some frequency range. Matrix entries can be referenced as S11 through S*nm*. While this is sufficient for most simple applications, it is also possible to index matrices by using the matrix indexer "()". For example, S(1,1) is equivalent to S11. The matrix indexer is offset by one meaning the first matrix entry is X(1,1). When it is used with swept data its operation is transparent with respect to the sweep. Both indexers can be combined. For example, it is possible to access S(1,1) at the first sweep point as S(1,1)[0]. As with the sweep indexer "[ ]", the matrix indexer can be used with wild cards and sequences to extract a submatrix from an original matrix.

Matrices

S-parameters above are an example of a matrix produced by the simulator. Matrices are more frequently found in signal processing applications. Mathematical operators implement matrix operations. Element-by-element operations can be performed by using the dot modified operators (.* and ./).

The matrix indexer conveniently operates over the complete sweep, just as the sweep indexer operates on all matrices in a sweep. As with scalars, the mathematical operators allow swept and non-swept quantities to be combined. For example, the first matrix in a sweep may be subtracted from all matrices in that sweep as

```
Y = X-X[0]
```

Multidimensional Sweeps and Indexing

In the previous examples we looked at single-dimensional sweeps. Multidimensional sweeps can be generated by the simulator by using multiple parameter sweeps. Expressions are designed to operate on the multidimensional data. Functions and operators behave in a meaningful way when a parameter sweep is added or taken away. A common example is DC IV characteristics.

The sweep indexer accepts a list of indices. Up to *N* indices are used to index N-dimensional data. If fewer than *N* lookup indices are used with the sweep indexer, then wild cards are inserted automatically to the left. This is best explained by referring to the above example files.

User-Defined Functions

By writing some Application Extension Language (AEL) code, you can define your own custom functions. The following file is provided specifically for this purpose:

$HPEESOF_DIR/expressions/ael/user_defined_fun.ael

By reviewing the other *_fun.ael* files in this directory, you can see how to write your own code. You can have as many functions as you like in this one file, and they will all be compiled upon program start-up. If you have a large number of functions to define, you may want to organize them into more than one file. In this case, include a line such as:

```
load("more_user_defined_fun.ael");
```

These load statements are added to the *user_defined_fun.ael* in the same directory in order to have your functions all compile. To create your own custom user defined functions:

1. Copy the $HPEESOF_DIR/expressions/ael/user_defined_fun.ael file to one of the following directories.

   $HOME/hpeesof/expressions/ael (User Config)

   $HPEESOF_DIR/custom/expressions/ael (Site Config)

   Create the appropriate subdirectories if they do not already exist. The User Config is setup for a single user. The Site Config can be set up by a CAD Manager or librarian to control a site configuration for a group of users.

2. Edit the new file and add any custom defined functions. If your custom functions reside in another file, you can add a *load* statement to your new *user_defined_fun.ael* file to include your functions in another file. For example:

   ```
   load("my_custom_functions_file.ael");
   ```

3. Save your changes to the new file and restart so your changes take effect. The search path looks in the following locations for user defined functions.

   $HOME/hpeesof/expressions/ael (User Config)

   $HPEESOF_DIR/custom/expressions/ael (Site Config)

   $HPEESOF_DIR/expressions/ael (Default Config)

Note If for some reason your functions are not recognized by the simulator, check to ensure that the *user_defined_fun.atf* (compiled version of user_defined_fun.ael file) was generated after restarting the software.

Functions Reference Format

The information below illustrates how each measurement expression in the functions reference is described.

<function name>

Presents a brief description of what the function does.

Syntax

Presents the general syntax of the function.

Arguments

Presents a table that includes each argument name, description, range, type, default value, and whether or not the argument is optional.

Examples

Presents one or more simple examples that use the function.

Defined in

Indicates whether the measurement function is defined in a script or is built in. All AEL functions are built in.

See also

Presents links to related functions, if there are any.

Notes/Equations

Describes any additional notes and/or equations that may help with understanding the function.

Chapter 2: Using Measurement Expressions in Advanced Design System

Measurement Expressions are equations that are used during simulation post processing. These expressions are entered into the program using the *MeasEqn* (Measurement Equation) component, available on the Simulation palettes in an Analog/RF Systems Schematic window (such as Simulation-AC or Simulation-Envelope), or from the Controllers palette in a Signal Processing Schematic window.

Many of the more commonly used measurement items are built in, and are found in the palettes of the appropriate simulator components. Common expressions are included as measurements, which makes it easy for beginning users to utilize the system. To make simulation and analyses convenient, all the measurement items, including the built-in items, can be edited to meet specific requirements. Underlying each measurement is a *function*; the functions themselves are available for modification. Moreover, it is also possible for you to write entirely new measurements and functions.

The measurement items and their underlying expressions are based on Advanced Design System's Application Extension Language (AEL). Consequently, they can serve a dual purpose:

- They can be used on the schematic page, in conjunction with simulations, to process the results of a simulation (this is useful, for example, in defining and reaching optimization goals). Unlike Simulator Expressions, the MeasEqn items are processed after the simulation engine has finishing its task and just before the dataset is written.
- They can be used in the Data Display window to process the results of a dataset that can be displayed graphically. Here the MeasEqn items are used to post-process the data written after simulation is complete.

In either of the above cases, the same syntax is used. However, some measurements can be used on the schematic page and not the Data Display window, and vice versa. These distinctions will be noted where they occur.

Note Not all Measurement Expression Functions have an explicit measurement component. These functions can be used by means of the MeasEqn component.

MeasEqn (Measurement Equations Component)

For a complete list of Measurement Functions, refer to the "Alphabetical Listing of Measurement Functions" on page 2-11 or consult the Index.

Symbol

Parameters

Instance Name

Displays name of the MeasEqn component in ADS. You can edit the instance name and place more than one MeasEqn component on the schematic.

Select Parameter

Selects an equation for editing.

Add	Add an equation to the Select Parameter field.
Cut	Delete an equation from the Select Parameter field.
Paste	Copy an equation that has been cut and place it in the Select Parameter field.

Meas

Enter your equation in this field.

Display parameter on schematic

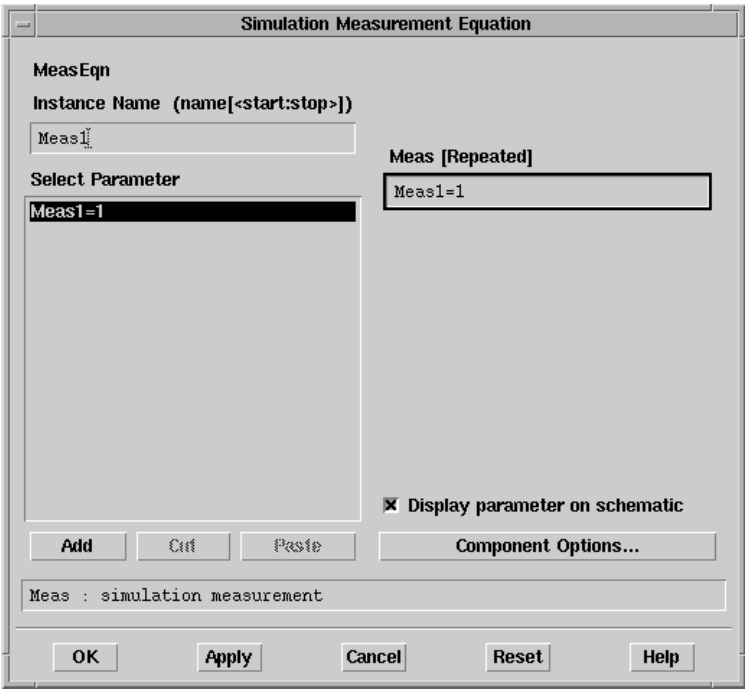
Displays or hides a selected equation on the ADS schematic.

Component Options

For information on this dialog box, refer to "*Editing Component Parameters*" in the ADS "*Schematic Capture and Layout*" documentation.

Notes/Equations

If you are using Advanced Design System, you can place a MeasEqn (Measurement Equation) component in a schematic window. By placing a MeasEqn component on an ADS schematic, you can write an equation that can be evaluated, following a simulation, and displayed in a Data Display window.



Note The if-then-else construct can be used in a MeasEqn component on a schematic. It has the following syntax: A = **if** (*condition*) **then** *true_expression* **else** *false_expression*

Pre-Configured Measurements in ADS

Expressions are available on the schematic page in ADS by means of the *MeasEqn* component. Pre-configured measurements are also available in various simulation palettes. These are designed to help you by presenting an initial equation, which can then be modified to suit the particular instance.

It is not possible to reference an equation in a *VarEqn* (variable equation) component within a *MeasEqn* (measurement equation). In addition, an equation in a MeasEqn component can reference other MeasEqns, any simulation output, and any swept variable. However, a VarEqn component cannot reference a MeasEqn.

The ready-made measurements available in the various simulator palettes in ADS are simply pre-configured expressions. These are designed to help you by presenting an initial equation, which can then be modified to suit the particular instance.

Table 2-1. Pre-configured Measurements Available in ADS

Simulator Palette	Pre-configured Measurement	Measurement Expression
Simulation-AC	BudFreq	“bud_freq()” on page 3-4
	BudGain	“bud_gain()” on page 3-7
	BudGainComp	“bud_gain_comp()” on page 3-10
	BudGamma	“bud_gamma()” on page 3-13
	BudIP3Deg	“bud_ip3_deg()” on page 3-15
	BudNF	“bud_nf()” on page 3-17
	BudNFDeg	“bud_nf_deg()” on page 3-19
	BudNoisePwr	“bud_noise_pwr()” on page 3-21
	BudPwrInc	“bud_pwr_inc()” on page 3-25
	BudPwrRefl	“bud_pwr_refl()” on page 3-27
	BudSNR	“bud_snr()” on page 3-29
	BudTN	“bud_tn()” on page 3-31
	BudVSWR	“bud_vswr()” on page 3-33

Table 2-1. Pre-configured Measurements Available in ADS

Simulator Palette	Pre-configured Measurement	Measurement Expression
Simulation-S-Param	MaxGain	“max_gain()” on page 9-31
	PwrGain	“pwr_gain()” on page 9-39
	VoltGain	“volt_gain()” on page 9-69
	VSWR	“vswr()” on page 9-72
	GainRipple	“ripple()” on page 9-40
	Mu	“mu()” on page 9-32
	MuPrime	“mu_prime()” on page 9-33
	StabFact	“stab_fact()” on page 9-50
	StabMeas	“stab_meas()” on page 9-51
	SmGamma1	“sm_gamma1()” on page 9-44
	SmGamma2	“sm_gamma2()” on page 9-45
	SmY1	“sm_y1()” on page 9-46
	SmY2	“sm_y2()” on page 9-47
	SmZ1	“sm_z1()” on page 9-48
	SmZ2	“sm_z2()” on page 9-49
	Yin	“yin()” on page 9-75
	Zin	“zin()” on page 9-81
	Yopt	“yopt()” on page 9-76
	Zopt	“zopt()” on page 9-82
	NsPwrInt	“ns_pwr_int()” on page 9-36
	NsPwrRefBW	“ns_pwr_ref_bw()” on page 9-37
	DevLinPhase	“dev_lin_phase()” on page 9-11
	GrpDelayRipple	“ripple()” on page 9-40
	GaCircle	“ga_circle()” on page 9-12
	GlCircle	“gl_circle()” on page 9-15
	GpCircle	“gp_circle()” on page 9-17

Table 2-1. Pre-configured Measurements Available in ADS

Simulator Palette	Pre-configured Measurement	Measurement Expression
Simulation-S-Param (cont.)	GsCircle	“gs_circle()” on page 9-19
	S_StabCircle	“s_stab_circle()” on page 9-41
	L_StabCircle	“l_stab_circle()” on page 9-26
	Map1Circle	“map1_circle()” on page 9-29
	Map2Circle	“map2_circle()” on page 9-30
	NsCircle	“ns_circle()” on page 9-34
Simulation-LSSP	MaxGain	“max_gain()” on page 9-31
	PwrGain	“pwr_gain()” on page 9-39
	VoltGain	“volt_gain()” on page 9-69
	VSWR	“vswr()” on page 9-72
	GainRipple	“ripple()” on page 9-40
	StabFact	“stab_fact()” on page 9-50
	StabMeas	“stab_meas()” on page 9-51
	Yin	“yin()” on page 9-75
	Zin	“zin()” on page 9-81
	DevLinPhase	“dev_lin_phase()” on page 9-11
	GainComp	“gain_comp()” on page 9-14
	PhaseComp	“phase_comp()” on page 9-38
Simulation-XDB	CDRange	“cdrange()” on page 6-3
Simulation-Transient	IfcTran	“ifc_tran()” on page 11-10
	IspecTran	“ispec_tran()” on page 11-11
	PfcTran	“pfc_tran()” on page 11-13
	PspecTran	“pspec_tran()” on page 11-15
	VfcTran	“vfc_tran()” on page 11-17
	VspecTran	“vspec_tran()” on page 11-18

Table 2-1. Pre-configured Measurements Available in ADS

Simulator Palette	Pre-configured Measurement	Measurement Expression
Optim/Stat/ Yield/DOE	statHist	“histogram_stat()” on page 10-11
	sensHist	“histogram_sens()” on page 10-9
Simulation-HB	It	“it()” on page 6-12
	Vt	“vt()” on page 6-38
	Pt	“pt()” on page 6-21
	Ifc	“ifc()” on page 6-5
	Pfc	“pfc()” on page 6-17
	Vfc	“vfc()” on page 6-35
	Pspec	“pspec()” on page 6-20
	DCtoRF	“dc_to_rf()” on page 6-4
	PAE	“pae()” on page 6-15
	IP3in	“ip3_in()” on page 6-6
	IP3out	“ip3_out()” on page 6-8
	CarrToIM	“carr_to_im()” on page 6-2
	IPn	“ipn()” on page 6-10
	SFDR	“sfdr()” on page 6-23
	BudFreq	“bud_freq()” on page 3-4
	BudGain	“bud_gain()” on page 3-7
	BudGainComp	“bud_gain_comp()” on page 3-10
	BudGamma	“bud_gamma()” on page 3-13
	BudIP3Deg	“bud_ip3_deg()” on page 3-15
	BudNoisePwr	“bud_nf()” on page 3-17
	BudPwrInc	“bud_nf_deg()” on page 3-19
	BudPwrRefl	“bud_noise_pwr()” on page 3-21
	BudSNR	“bud_pwr_inc()” on page 3-25
	BudVSWR	“bud_pwr_refl()” on page 3-27

Measurement Expressions Examples in ADS

The descriptions of ADS expressions provided in the manual are accompanied by a set of example designs and data display pages. These examples show how expressions are used in Advanced Design System. For specific ADS examples, refer to Table 2-2 and Table 2-3.

Many measurement expression examples can be found in the ADS tutorial project:

$HPEESOF_DIR/examples/Tutorial/

Table 2-2. ADS Examples Using Measurement Expressions

Example Design/Data Display	See Also
express_meas_prj/networks/simple_meas_1.dsn	"Measurements and Expressions" on page 1-8
express_meas_prj/variable.dds	"Generating Data" on page 1-8
express_meas_prj/if_then_else_1.dds	"Conditional Expressions" on page 1-6
express_meas_prj/gen_1.dds	"Generating Data" on page 1-8
express_meas_prj/sweep.dds	"Simple Sweeps and Using "[ ]"" on page 1-9
express_meas_prj/sparam_1.dsn & analysis.dds (see S-parameter 1 page)	"S-parameters and Matrices" on page 1-9
express_meas_prj/Matrix.dds	"Matrices" on page 1-10
express_meas_prj/multidim_1.dsn & multidim_1.dds	"Multidimensional Sweeps and Indexing" on page 1-10
express_meas_prj/analysis.dds (see Harmonic Balance page)	"Working with Harmonic Balance Data" on page 6-1
express_meas_prj/tran_1.dsn & tran_1.dds	"Working with Transient Data" on page 11-1
express_meas_prj/env_1.dds	"Working with Envelope Data" on page 4-1
DataAccess_prj/Truth_MonteCarlo.dds	These functions can be applied to the data in the example: "max_outer()" on page 7-61 "mean_outer()" on page 10-16 "min_outer()" on page 7-64

Table 2-2. ADS Examples Using Measurement Expressions

Example Design/Data Display	See Also
yldex1_prj/measurement_hist.dds & worstcase_measurement_hist.dds	"histogram()" on page 10-6 "histogram_multiDim()" on page 10-8 "histogram_stat()" on page 10-11
ModSources_prj/QAM_16_ConstTraj.dds See also: $HPEESOF_DIR/examples/RF_Board/NADC_PA_prj/ConstEVM_Eqns.dds	"constellation()" on page 11-2
BER_Env_prj/timing_doc.dds See also: $HPEESOF_DIR/examples/RF_Board/NADC_PA_prj/NADC_PA_Test.dsn and ConstEVM.dds	"const_evm()" on page 4-12
The spur_track() and spur_track_with_if() functions can be applied to the data in the example: Com_Sys/Spur_Track_prj/MixerSpurs2MHz.dds	"spur_track()" on page 6-27 "spur_track_with_if()" on page 6-29
$HPEESOF_DIR/examples/RF_Board/NADC_PA_prj/NADC_PA_ACPRtransmitted.dds	"acpr_vi()" on page 4-3
$HPEESOF_DIR/examples/Tutorial/ModSources_prj/IS95RevLinkSrc.dds	"acpr_vr()" on page 4-5
$HPEESOF_DIR/examples/RF_Board/NADC_PA_prj/NADC_PA_ACPRtransmitted.dds	"channel_power_vi()" on page 4-8
$HPEESOF_DIR/examples/RF_Board/NADC_PA_prj/NADC_PA_ACPRreceived.dds	"channel_power_vr()" on page 4-10
$HPEESOF_DIR/examples/Tutorial/sweep.dds	"permute()" on page 5-28

Additional measurement expression examples can be found in the ADS Design Guides:

Table 2-3. Examples Using the ADS Design Guides

Design Guide	See Also
Refer to the *S-Parameter to Time Transform & Single Ended TDR/TDT Impulse Simulation* in the Signal Integrity Applications under the DesignGuide menu. SP_measVSmod_NEW.dds	"tdr_sp_gamma()" on page 9-59 "tdr_sp_imped()" on page 9-61 "tdr_step_imped()" on page 9-63

Table 2-3. Examples Using the ADS Design Guides

Design Guide	See Also
Refer to the *Eye Diagram Jitter Histogram Measurement* in the Signal Integrity Applications under the DesignGuide menu	“cross_hist()” on page 4-15
Refer to the *Eye Closure Measurements* in the Signal Integrity Applications under the DesignGuide menu.	“eye_amplitude()” on page 8-8
	“eye_closure()” on page 8-10
	“eye_fall_time()” on page 8-12
	“eye_height()” on page 8-14
	“eye_rise_time()” on page 8-16

Alphabetical Listing of Measurement Functions

Consult the Index for an alternate method of accessing measurement functions.

For information on simulator functions, refer to the the Simulator Expressions documentation.

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z

Chapter 3: Circuit Budget Functions

This chapter describes the circuit budget functions in detail. The functions are listed in alphabetical order.

Note The circuit budget functions are not directly available in RF Design Environment (RFDE) since they are generally intended for use in Advanced Design System. If you have a need to use circuit budget functions in RFDE, please consult your Agilent Technologies sales representative or technical support to request help from solution services.

Budget Measurement Analysis

Budget analysis determines the signal and noise performance for elements in the top-level design. Therefore, it is a key element of system analysis. Budget measurements show performance at the input and output pins of the top-level system elements. This enables the designer to adjust, for example, the gains at various components, to reduce nonlinearities. These measurements can also indicate the degree to which a given component can degrade overall system performance.

Budget measurements are performed upon data generated during a special mode of circuit simulation. AC and HB simulations are used in budget mode depending upon if linear or nonlinear analysis is needed for a system design. The controllers for these simulations have a flag called, *OutputBudgetIV* which must be set to "*yes*" for the generation of budget data. Alternatively, the flag can be set by editing the AC or HB

simulation component and selecting the *Perform Budget simulation* button on the Parameters tab.

Budget data contains signal voltages and currents, and noise voltages at every node in the top level design. Budget measurements are functions that operate upon this data to characterize system performance parameters including gain, power, and noise figure. These functions use a constant reference impedance for all nodes for calculations. By default this impedance is 50 Ohms. The available source power at the input network port is assumed to equal the incident power at that port.

Budget measurements are available in the schematic and the data display windows. The budget functions can be evaluated by placing the budget components from Simulation-AC or Simulation-HB palettes on the schematic. The results of the budget measurements at the terminal(s) are sorted in ascending order of the component names. The component names are attached to the budget data as additional dependent variables. To use one of these measurements in the data display window, first reference the appropriate data in the default dataset, and then use the equation component to write the budget function. For more detailed information about Budget Measurement Analysis, see "Budget Analysis" in the chapter "Using Circuit Simulators for RF System Analysis" in the *Circuit Simulation* documentation.

Note The budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Frequency Plan

A frequency plan of the network is determined for budget mode AC and HB simulations. This plan tracks the reference carrier frequency at each node in a network. When performing HB budget, there may be more than one frequency plan in a given network. This is the case when double side band mixers are used. Using this plan information, budget measurements are performed upon selected reference frequencies, which can differ at each node. When mixers are used in an AC simulation, be sure to set the *Enable AC frequency conversion* option on the controller, to generate the correct plan.

The budget measurements can be performed on arbitrary networks with multiple signal paths between the input and output ports. As a result, the measurements can be affected by reflection and noise generated by components placed between the

terminal of interest and the output port on the same signal path or by components on different signal paths.

Reflection and Backward-Traveling Wave Effects

The effects of reflections and backward-traveling signal and noise waves generated by components along the signal path can be avoided by inserting a forward-traveling wave sampler between the components. A forward-traveling wave sampler is an ideal, frequency-independent directional coupler that allows sampling of forward-traveling voltage and current waves

This sampler can be constructed using the equation-based linear three-port S-parameter component. To do this, set the elements of the scattering matrix as follows: *S12 = S21 = S31 = 1*, and all other *Sij = 0*. The temperature parameter is set to -273.16 deg C to make the component noiseless. A noiseless shunt resistor is attached to port 3 to sample the forward-traveling waves.

bud_freq()

Returns the frequency plan of a network

Syntax

y = bud_freq(freqIn, pinNumber, "simName") for AC analysis or y = bud_freq(planNumber, pinNumber) for HB analysis

Arguments

Name	Description	Range	Type	Required
freqIn	input source frequency	(0, ∞)	real	no
planNumber	represents the chosen frequency plan and is required when using the bud_freq() function with HB data.	[1, ∞)	integer	yes
pinNumber	used to choose which pins of each network element are referenced †	[1, ∞)	integer	no
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string	no

† If 1 is passed as the pinNumber, the frequency plan displayed references pin 1 of each element; otherwise, the frequency plan is displayed for all pins of each element. (Note that this means it is not possible to select only pin 2 of each element, for example.) By default, the frequency plan is displayed for pin 1 of each element

Examples

```
x = bud_freq()
```
returns frequency plan for AC analysis

```
x = bud_freq(1MHz)
```
returns frequency plan for frequency swept AC analysis. By passing the value of 1MHz, the plan is returned for the subset of the sweep when the source value is 1MHz

```
x = bud_freq(2)
```
for HB, returns a selected frequency plan, 2, with respect to pin 1 of every network element

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

Notes/Equations

Used in AC and harmonic balance (HB) simulations.

This function is used in AC and HB simulations with the budget parameter turned on. For AC, the options are to pass no parameters, or the input source frequency (freqIn), for the first parameter if a frequency sweep is performed. freqIn can still be passed if no sweep is performed, table data is just formatted differently. The first argument must be a real number for AC data and the second argument is an integer, used optionally to choose pin references.

When a frequency sweep is performed in conjunction with AC, the frequency plan of a particular sweep point can be chosen.

For HB, this function determines the fundamental frequencies at the terminal(s) of each component, thereby given the entire frequency plan for a network. Sometimes more than one frequency plan exists in a network. For example when double sideband mixers are used. This function gives the user the option of choosing the frequency plan of interest.

Note that a negative frequency at a terminal means that a spectral inversion has occurred at the terminal. For example, in frequency-converting AC analysis, where vIn and vOut are the voltages at the input and output ports, respectively, the relation may be either vOut=alpha*vIn if no spectral inversion has occurred, or vOut=alpha*conj(vIn) if there was an inversion. Inversions may or may not occur depending on which mixer sidebands one is looking at.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path

variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_gain()

Returns budget transducer-power gain

Syntax

y = bud_gain(vIn, iIn, Zs, Plan, pinNumber, "simName") or y = bud_gain("SourceName", SrcIndx, Zs, Plan)

Arguments

Name	Description	Range	Type	Default	Required
vIn	voltage flowing into the input port	(-∞, ∞)	complex		yes
iIn	current flowing into the input port	(-∞, ∞)	complex		yes
SourceName	component name at the input port		string		yes
SrcIndx †	frequency index that corresponds to the source frequency to determine which frequency to use from a multitone source as the reference signal	[1, ∞)	integer	1	no
Zs	input source port impedance	[0, ∞)	real	50.0	no
Plan †	number of the selected frequency plan(needed only for HB)		string		no
pinNumber	Used to choose which pins of each network element are referenced ‡	[1, ∞)	integer	1	no

Name	Description	Range	Type	Default	Required
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string		no

† Note that for AC simulation, both the SrcIndx and Plan arguments must not be specified; these are for HB only.
‡ If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_gain(PORT1.t1.v, PORT1.t1.i)
```
or
```
x = bud_gain(“PORT1”)
```

```
y= bud_gain(PORT1.t1.v, PORT1.t1.i, 75)
```
or
```
y= bud_gain(“PORT1”, , 75., 1)
```

```
z = bud_gain(PORT1.t1.v[3], PORT1.t1.i[3], , 1)
```
or
```
z= bud_gain(“PORT1”, 3, , 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_gain_comp()

Notes/Equations

Used in AC and harmonic balance simulations

This is the power gain (in dB) from the input port to the terminal(s) of each component, looking into that component. Power gain is defined as power delivered to the resistive load minus the power available from the source. Note that the fundamental frequency at different pins can be different. If vIn and iIn are passed directly, one may want to use the index of the frequency sweep explicitly to reference the input source frequency.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_gain_comp()

Returns budget gain compression at fundamental frequencies as a function of power

Syntax

y = bud_gain_comp(vIn, iIn, Zs, Plan, freqIndex, pinNumber, "simName") or y = bud_gain_comp("SourceName", SrcIndx, Zs, Plan, freqIndex, pinNumber, "simName")

Arguments

Name	Description	Range	Type	Default	Required
vIn	voltage flowing into the input port	(-∞, ∞)	complex		yes
iIn	current flowing into the input port	(-∞, ∞)	complex		yes
SourceName	component name at the input port		string		yes
SrcIndx †	frequency index that corresponds to the source frequency to determine which frequency to use from a multitone source as the reference signal	[1, ∞)	integer	1	no
Zs	input source port impedance	[0, ∞)	real	50.0	no
freqIndex †	index of harmonic frequency	(-∞, ∞)	integer		no
Plan ‡	number of the selected frequency plan(needed only for HB)		string		no
pinNumber	Used to choose which pins of each network element are referenced † ‡	[1, ∞)	integer	1	no

Name	Description	Range	Type	Default	Required
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string		no

† Used if Plan is not selected.
‡ Note that for AC simulation, both the SrcIndx and Plan arguments must not be specified; these are for HB only.
† ‡ If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_gain_comp(PORT1.t1.v[3], PORT1.t1.i[3], , 1)
x = bud_gain_comp("PORT1", 3, , 1)
```
returns the gain compression at the fundamental frequencies as a function of power

```
y= bud_gain_comp(PORT1.t1.v[3], PORT1.t1.i[3], , , 1)
y= bud_gain_comp("PORT1", 3, , , 1)
```
returns the gain compression at the second harmonic frequency as a function of power

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_gain()

Notes/Equations

Used in Harmonic balance simulation with sweep

This is the gain compression (in dB) at the given input frequency from the input port to the terminal(s) of each component, looking into that component. Gain compression is defined as the small signal linear gain minus the large signal gain. Note that the fundamental frequency at each element pin can be different by referencing the frequency plan. A power sweep of the input source must be used in conjunction with HB. The first power sweep point is assumed to be in the linear region of operation.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

This function does not support the budget path feature.

bud_gamma()

Returns the budget reflection coefficient

Syntax

y = bud_gamma(Zref, Plan, pinNumber, "simName")

Arguments

Name	Description	Range	Type	Default	Required
Zref	input source port impedance	[0, ∞)	real	50.0	no
Plan †	number of the selected frequency plan(needed only for HB)		string		no
pinNumber	Used to choose which pins of each network element are referenced ‡	[1, ∞)	integer	1	no
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string		no

† Note that for AC simulation, both the SrcIndx and Plan arguments must not be specified; these are for HB only.
‡ If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_gamma()
```
returns reflection coefficient at all frequencies

```
y = bud_gamma(75, 1)
```
returns reflection coefficient at reference frequencies in plan 1

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_vswr()

Notes/Equations

Used in AC and harmonic balance simulations

This is the complex reflection coefficient looking into the terminal(s) of each component. Note that the fundamental frequency at different pins can in general be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_ip3_deg()

Returns the budget third-order intercept point degradation

Syntax

y = bud_ip3_deg(vOut, LinearizedElement, fundFreq, imFreq, zRef)

Arguments

Name	Description	Range	Type	Default	Required
vOut	signal voltage at the output	(-∞, ∞)	real, complex		yes
LinearizedElement	variable containing the names of the linearized components		string		yes
fundFreq	harmonic frequency indices for the fundamental frequency	(-∞, ∞)	integer		yes
imFreq	harmonic frequency indices for the intermodulation frequency	(-∞, ∞)	integer		yes
Zref	input source port impedance	[0, ∞)	real	50.0	no

Examples

```
y = bud_ip3_deg(vOut, LinearizedElement, {1, 0}, {2, -1})
```

returns the budget third-order intercept point degradation

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

ip3_out(), ipn()

Notes/Equations

Used in Harmonic balance simulation with the BudLinearization Controller.

This measurement returns the budget third-order intercept point degradation from the input port to any given output port. It does this by setting to linear each component in the top-level design, one at a time.

For the components that are linear to begin with, this measurement will not yield any useful information. For the nonlinear components, however, this measurement will indicate how the nonlinearity of a certain component degrades the overall system IP3. To perform this measurement, the BudLinearization Controller needs to be placed in the schematic window. If no component is specified in this controller, all components on the top level of the design are linearized one at a time, and the budget IP3 degradation is computed.

Budget Path measurements

This function does not support the budget path feature.

bud_nf()

Returns the budget noise figure

Syntax

y = bud_nf(vIn, iIn, noisevIn, Zs, BW, pinNumber, "simName") or y = bud_nf("SourceName")

Arguments

Name	Description	Range	Type	Default	Required
vIn	voltage flowing into the input port	(-∞, ∞)	complex		yes
iIn	current flowing into the input port	(-∞, ∞)	complex		yes
noisevIn	noise input at the input port	(-∞, ∞)	complex		yes
Zs	input source port impedance	[0, ∞)	real	50.0	no
BW †	bandwidth	[1, ∞)	real	1	no
pinNumber	Used to choose which pins of each network element are referenced ‡	[1, ∞)	integer	1	no
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string		no
SourceName	component name at the input port		string		yes

† BW must be set as the value of Bandwidth used on the noise page of the AC controller
‡ If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_nf(PORT1.t1.v, PORT1.t1.i, PORT1.t1.v.noise)
x = bud_nf(“PORT1”)
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_nf_deg(), bud_tn()

Notes/Equations

Used in AC simulation

This is the noise figure (in dB) from the input port to the terminal(s) of each component, looking into that component. The noise analysis control parameters in the AC Simulation component must be selected: “Calculate Noise” and “Include port noise”. For the source, the parameter “Noise” should be set to yes. The noise figure is always calculated per IEEE standard definition with the input termination at 290 K.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_nf_deg()

Returns budget noise figure degradation

Syntax

y = bud_nf_deg(vIn, iIn, vOut, iOut, vOut.NC.vnc, vOut.NC.name, Zs, BW)

Arguments

Name	Description	Range	Type	Default	Required
vIn	voltage flowing into the input port	(-∞, ∞)	complex		yes
iIn	current flowing into the input port	(-∞, ∞)	complex		yes
vOut	voltage flowing into the output port	(-∞, ∞)	complex		yes
iOut	current flowing into the output port	(-∞, ∞)	complex		yes
vOut.NC.vnc	noise contributions at the output port		string		yes
vOut.NC.name	noise contributions component names at the output port		string		yes
Zs	input source port impedance	[0, ∞)	real	50.0	no
BW †	bandwidth	[1, ∞)	real	1	no
† BW must be set as the value of Bandwidth used on the noise page of the AC controller					

Examples

```
x = bud_nf_deg(PORT1.t1.v, PORT1.t1.i, Term1.t1.v, Term1.t1.i, vOut.NC.vnc,
vOut.NC.name)
```

```
x = bud_nf_deg(“PORT1”, ”Term1”, “vOut”)
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_nf(), bud_tn()

Notes/Equations

Used in AC simulation

The improvement of system noise figure is given when each element is made noiseless. This is the noise figure (in dB) from the source port to a specified output port, obtained while setting each component noiseless, one at a time. The noise analysis and noise contribution control parameters in the AC Simulation component must be selected. For noise contribution, the output network node must be labeled and referenced on the noise page in the AC Controller. Noise contributors mode should be set to “Sort by Name.” The option “Include port noise “on the AC Controller should be selected. For the source, the parameter “Noise” should be set to yes. For this particular budget measurement the AC controller parameter “OutputBudgetIV” can be set to no. The noise figure is always calculated per IEEE standard definition with the input termination at 290 K.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

This function does not support the budget path feature.

bud_noise_pwr()

Returns the budget noise power

Syntax

y = bud_noise_pwr(Zref, Plan, pinNumber, "simName")

Arguments

Name	Description	Range	Type	Default	Required
Zref	input source port impedance	[0, ∞)	real	50.0	no
Plan	number of the selected frequency plan(needed only for HB)		string		no
pinNumber	Used to choose which pins of each network element are referenced †	[1, ∞)	integer	1	no
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string		no

† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1

Examples

```
x = bud_noise_pwr()
```

returns the noise power at all frequencies

```
y = bud_noise_pwr(75, 1)
```

returns the noise power at reference frequencies in plan 1

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_pwr()

Notes/Equations

Used in AC and harmonic balance simulations

This is the noise power (in dBm) at the terminal(s) of each component, looking into the component. If Zref is not specified, the impedance that relates the signal voltage and current is used to calculate the noise power. Note that the fundamental frequency at different pins can be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

This function does not have the bandwidth parameter, so the results rely entirely on the setting of the bandwidth in the Noise tab of the simulation controller.

For AC noise and budget calculations, the bandwidth parameter flatly scales the noise voltages, and consequently noise powers, SNR, etc., regardless of the frequency response.

If you make a frequency sweep then the narrow band noise voltages properly follow the frequency characteristic of the circuit and are presented as a function of the swept frequency values. However, changing the bandwidth in the simulator controller would again just flatly rescale all the values, regardless of whether the frequency response remains flat or changes drastically over the (local) bandwidth, or whether the adjacent bands overlap or not.

The only true integration over a bandwidth is done for the phase noise (as one of the options in the NoiseCon controller).

A work-around solution is to do an appropriately wide frequency sweep setting the bandwidth value in the AC controller to that of the frequency step. Then adding all the powers (need to be first converted from dBm to watts) would do the integration.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_pwr()

Returns the budget signal power in dBm

Syntax

y = bud_pwr(Plan, pinNumber, "simName")

Arguments

Name	Description	Range	Type	Default	Required
Plan	number of the selected frequency plan(needed only for HB)		string		no
pinNumber	Used to choose which pins of each network element are referenced †	[1, ∞)	integer	1	no
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string		no
† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1					

Examples

```
x = bud_pwr()
```

returns the signal power at all frequencies when used in AC or HB simulations

```
y = bud_pwr(50, 1)
```

returns the signal power at reference frequencies in plan 1 when used for HB simulations

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_noise_pwr()

Notes/Equations

Used in AC and harmonic balance simulations.

This is the signal power (in dBm) at the terminal(s) of each component, looking into the component. Note that the fundamental frequency at different pins can be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_pwr_inc()

Returns the budget incident power

Syntax

y = bud_pwr_inc(Zref, Plan, pinNumber, "simName")

Arguments

Name	Description	Range	Type	Default	Required
Zref	input source port impedance	[0, ∞)	real	50.0	no
Plan	number of the selected frequency plan(needed only for HB)		string		no
pinNumber	Used to choose which pins of each network element are referenced †	[1, ∞)	integer	1	no
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string		no
† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1					

Examples

```
x = bud_pwr_inc()
```
returns incident power at all frequencies

```
y = bud_pwr_inc(75, 1)
```
returns incident power at reference frequencies in plan 1

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_pwr_refl()

Notes/Equations

Used in AC and harmonic balance simulations

This is the incident power (in dBm) at the terminal(s) of each component, looking into the component. Note that the fundamental frequency at different pins can be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_pwr_refl()

Returns the budget reflected power

Syntax

y = bud_pwr_refl(Zref, Plan, pinNumber, "simName")

Arguments

Name	Description	Range	Type	Default	Required
Zref	input source port impedance	[0, ∞)	real	50.0	no
Plan	number of the selected frequency plan(needed only for HB)		string		no
pinNumber	Used to choose which pins of each network element are referenced †	[1, ∞)	integer	1	no
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string		no
† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1					

Examples

```
x = bud_pwr_refl()
```
returns reflected power at all frequencies

```
y = bud_pwr_refl(75, 1)
```
returns reflected power at reference frequencies in plan 1

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_pwr_inc()

Notes/Equations

Used in AC and harmonic balance simulations

This is the reflected power (in dBm) at the terminal(s) of each component, looking into the component. Note that the fundamental frequency at different pins can be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_snr()

Returns the budget signal-to-noise-power ratio

Syntax

y = bud_snr(Plan, pinNumber, "simName")

Arguments

Name	Description	Range	Type	Default	Required
Plan	number of the selected frequency plan(needed only for HB)		string		no
pinNumber	Used to choose which pins of each network element are referenced †	[1, ∞)	integer	1	no
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string		no
† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1					

Examples

```
x = bud_snr()
```
returns the SNR at all frequencies

```
y = bud_snr(1)
```
returns the SNR at reference frequencies in plan 1

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

Notes/Equations

Used in AC and harmonic balance simulations

This is the SNR (in dB) at the terminal(s) of each component, looking into that component. Note that the fundamental frequency at different pins can in general be different, and therefore values are given for all frequencies unless a Plan is

referenced. The noise analysis control parameter in the AC and Harmonic Balance Simulation components must be selected. For the AC Simulation component select: “Calculate Noise” and “Include port noise.” For the source, the parameter “Noise” should be set to yes. In Harmonic Balance select the “Nonlinear noise” option.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_tn()

Returns the budget equivalent output-noise temperature

Syntax

y = bud_tn(vIn, iIn, noisevIn, Zs, BW, pinNumber, "simName") or y = bud_tn("SourceName")

Arguments

Name	Description	Range	Type	Default	Required
vIn	voltage flowing into the input port	(-∞, ∞)	complex		yes
iIn	current flowing into the input port	(-∞, ∞)	complex		yes
noisevIn	noise input at the input port	(-∞, ∞)	complex		yes
Zs	input source port impedance	[0, ∞)	real	50.0	no
BW †	bandwidth	[1, ∞)	real	1	no
pinNumber	Used to choose which pins of each network element are referenced ‡	[1, ∞)	integer	1	no
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string		no
SourceName	component name at the input port		string		yes

† BW must be set as the value of Bandwidth used on the noise page of the AC controller
‡ If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_tn(PORT1.t1.v, PORT1.t1.i, PORT1.t1.v.noise)
x = bud_tn(“PORT1”)
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_nf(), bud_nf_deg()

Notes/Equations

Used in AC simulation

This is an equivalent output-noise temperature (in degrees Kelvin) from the input port to the terminal(s) of each component, looking into that component. The noise analysis and noise contribution control parameters in the AC Simulation component must be selected: “Calculate Noise” and “Include port noise.” For the source, the parameter “Noise” should be set to yes. The output-noise temperature is always calculated per IEEE standard definition with the input termination at 290 K.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_vswr()

Returns the budget voltage-standing-wave ratio

Syntax

y = bud_vswr(Zref, Plan, pinNumber, "simName")

Arguments

Name	Description	Range	Type	Default	Required
Zref	input source port impedance	[0, ∞)	real	50.0	no
Plan	number of the selected frequency plan(needed only for HB)		string		no
pinNumber	Used to choose which pins of each network element are referenced †	[1, ∞)	integer	1	no
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.		string		no
† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1					

Example

```
x = bud_vswr()
```
returns the vswr at all frequencies

```
y = bud_vswr(75, 1)
```
returns the vswr at reference frequencies in plan 1

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_gamma()

Notes/Equations

Used in AC and harmonic balance simulations

This is the VSWR looking into the terminal(s) of each component. Note that the fundamental frequency at different pins can be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

Chapter 4: Circuit Envelope Functions

This chapter describes the circuit envelope functions in detail. The functions are listed in alphabetical order.

Working with Envelope Data

Circuit Envelope Analysis produces complex frequency spectra as a function of time. A single envelope analysis can produce 2-dimensional data where the outermost independent variable is time and the innermost is frequency or harmonic number.

Indexing can be used to look at a harmonic against time, or a spectrum at a particular time index.

acpr_vi()

Computes the adjacent-channel power ratio following a Circuit Envelope simulation

Syntax

ACPRvals = acpr_vi(voltage, current, mainCh, lowerAdjCh, upperAdjCh, winType, winConst)

Arguments

Name	Description	Range	Type	Default	Required
voltage	single complex voltage spectral component (for example, the fundamental) across a load versus time	(-∞, ∞)	complex		yes
current	single complex current spectral component into the same load versus time	(-∞, ∞)	complex		yes
mainCh	two-dimensional vector defining the main channel frequency limits (as an offset from the single voltage and current spectral component)	(-∞, ∞)	real		yes
lowerAdjCh	the two-dimensional vector defining the lower adjacent-channel frequency limits (as an offset from the single voltage and current spectral component);	(-∞, ∞)	real		yes
upperAdjCh	two-dimensional vector defining the upper adjacent channel frequency limits (as an offset from the single voltage and current spectral component);	(-∞, ∞)	real		yes
winType	window type	†	string		no

Name	Description	Range	Type	Default	Required
winConst	window constant that affects the shape of the applied window.	[0, ∞)	real	0.75	no
† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"					

Examples

```
VloadFund = vload[1]
IloadFund = iload.i[1]
mainlimits = {-16.4 kHz, 16.4 kHz}
UpChlimits = {mainlimits + 30 kHz}
LoChlimits = {mainlimits - 30 kHz}
TransACPR = acpr_vi(VloadFund, IloadFund, mainlimits, LoChlimits,
UpChlimits, "Kaiser")
```

where vload is the named connection at a load, and iload.i is the name of the current probe that samples the current into the node. The {} braces are used to define vectors, and the upper channel limit and lower channel limit frequencies do not need to be defined by means of the vector that defines the main channel limits.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

acpr_vr(), channel_power_vi(), channel_power_vr(), relative_noise_bw()

Notes/Equations

Used in Adjacent-channel power computations.

The user must supply a single complex voltage spectral component (for example, the fundamental) across a load versus time and a single complex current spectral component into the same load. The user must also supply the upper and lower adjacent-channel and main-channel frequency limits, as offsets from the spectral component frequency of the voltage and current. These frequency limits must be entered as two-dimensional vectors. An optional window and window constant may also be supplied, for use in processing non-periodic data.

acpr_vr()

Computes the adjacent-channel power ratio following a Circuit Envelope simulation

Syntax

ACPRvals = acpr_vr(voltage, resistance, mainCh, lowerAdjCh, upperAdjCh, winType, winConst)

Arguments

Name	Description	Range	Type	Default	Required
voltage	single complex voltage spectral component (for example, the fundamental) across a load versus time	(-∞, ∞)	complex		yes
resistance	load resistance in ohms	(-∞, ∞)	complex	50	no
mainCh	two-dimensional vector defining the main channel frequency limits (as an offset from the single voltage spectral component)	(-∞, ∞)	real		yes
lowerAdjCh	the two-dimensional vector defining the lower adjacent-channel frequency limits (as an offset from the single voltage spectral component);	(-∞, ∞)	real		yes
upperAdjCh	two-dimensional vector defining the upper adjacent channel frequency limits (as an offset from the single voltage spectral component);	(-∞, ∞)	real		yes
winType	window type	†	string		no

Name	Description	Range	Type	Default	Required
winConst	window constant that affects the shape of the applied window.	[0, ∞)	real	0.75	no
† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"					

Examples

```
Vfund = vOut[1]
mainlimits = {-(1.2288 MHz/2), (1.2288 MHz/2)}
UpChlimits = {885 kHz, 915 kHz}
LoChlimits = {-915 kHz, -885 kHz}
TransACPR = acpr_vr(VloadFund, 50, mainlimits, LoChlimits, UpChlimits,
"Kaiser")
```

where vOut is the named connection at a resistive load. The {} braces are used to define vectors.

Note vOut is a named connection on the schematic. Assuming that a Circuit Envelope simulation was run, vOut is output to the dataset as a two-dimensional matrix. The first dimension is time, and there is a value for each time point in the simulation. The second dimension is frequency, and there is a value for each fundamental frequency, each harmonic, and each mixing term in the analysis, as well as the baseband term.

vOut[1] is the equivalent of vOut[::, 1], and specifies all time points at the lowest non-baseband frequency (the fundamental analysis frequency, unless a multi-tone analysis has been run and there are mixing products). For former MDS users, the notation "vOut[*, 2]" in MDS corresponds to the notation of "vOut[1]".

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

acpr_vi(), channel_power_vi(), channel_power_vr(), relative_noise_bw()

Notes/Equations

Used in Adjacent-channel power computations.

The user must supply a single complex voltage spectral component (for example, the fundamental) across a resistive load versus time and the load resistance. The user must also supply the upper and lower adjacent-channel and main-channel frequency limits, as offsets from the spectral component frequency of the voltage. These frequency limits must be entered as two-dimensional vectors. An optional window and window constant may also be supplied, for use in processing non-periodic data.

channel_power_vi()

Computes the power (in watts) in an arbitrary frequency channel following a Circuit Envelope simulation

Syntax

Channel_power = channel_power_vi(voltage, current, mainCh, winType, winConst)

Arguments

Name	Description	Range	Type	Default	Required
voltage	single complex voltage spectral component (for example, the fundamental) across a load versus time	(-∞, ∞)	complex		yes
current	single complex current spectral component into the same load versus time	(-∞, ∞)	complex		yes
mainCh	two-dimensional vector defining channel frequency limits (as an offset from the single voltage and current spectral component †	(-∞, ∞)	real		yes
winType	window type	†	string		no
winConst	window constant that affects the shape of the applied window.	[0, ∞)	real	0.75	no

† note that these frequency limits do not have to be centered on the voltage and current spectral component frequency.
† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"

Examples

```
VloadFund = vload[1]
IloadFund = iload.i[1]
mainlimits = {-16.4 kHz, 16.4 kHz}
Main_Channel_Power = channel_power_vi(VloadFund, IloadFund, mainlimits,
"Kaiser")
```

where vload is the named connection at a load, and iload.i is the name of the current probe that samples the current into the node. The {} braces are used to define a vector. Note that the computed power is in watts. Use the following equation to convert the power to dBm.

```
Main_Channel_Power_dBm = 10 * log(Main_Channel_Power) + 30
```

Do not use the dBm function, which operates on voltages.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

acpr_vi(), acpr_vr(), channel_power_vr()

Notes/Equations

Used in Channel power computations.

The user must supply a single complex voltage spectral component (for example, the fundamental) across a load versus time, and a single complex current spectral component into the same load. The user must also supply the channel frequency limits, as offsets from the spectral component frequency of the voltage and current. These frequency limits must be entered as a two-dimensional vector. An optional window and window constant may also be supplied, for use in processing non-periodic data.

channel_power_vr()

Computes the power (in watts) in an arbitrary frequency channel following a Circuit Envelope simulation

Syntax

Channel_power = channel_power_vr(voltage, resistance, mainCh, winType, winConst)

Arguments

Name	Description	Range	Type	Default	Required
voltage	single complex voltage spectral component (for example, the fundamental) across a load versus time	(-∞, ∞)	complex		yes
resistance	load resistance in ohms	(-∞, ∞)	complex	50	no
mainCh	two-dimensional vector defining the main channel frequency limits (as an offset from the single voltage spectral component) †	(-∞, ∞)	real		yes
winType	window type	† †	string		no
winConst	window constant that affects the shape of the applied window.	[0, ∞)	real	0.75	no

† note that these frequency limits do not have to be centered on the voltage and current spectral component frequency
† † winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"

Examples

```
Vmain_fund = Vmain[1]
mainlimits = {-16.4 kHz, 16.4 kHz}
Main_Channel_Power = channel_power_vr(Vmain_fund, 50, mainlimits, "Kaiser")
```

where Vmain is the named connection at a resistive load (50 ohms in this case.) The {} braces are used to define a vector. Note that the computed power is in watts. Use the following equation to convert the power to dBm.

```
Main_Channel_Power_dBm = 10 * log(Main_Channel_Power) + 30
```

Do not use the dBm function, which operates on voltages.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

acpr_vi(), acpr_vr(), channel_power_vi()

Notes/Equations

Used in Channel power computations.

The user must supply a single complex voltage spectral component (for example, the fundamental) across a load versus time and the resistance of the load. The user must also supply the channel frequency limits, as offsets from the spectral component frequency of the voltage. These frequency limits must be entered as a two-dimensional vector. An optional window and window constant may also be supplied, for use in processing non-periodic data.

const_evm()

Takes the results of a Circuit Envelope simulation and generates data for the ideal and distorted constellation and trajectory diagrams, as well as the error vector magnitude, in percent, and a plot of the error vector magnitude versus time.

Syntax

data = const_evm(vfund_ideal, vfund_dist, symbol_rate, sampling_delay, rotation, transient_duration, path_delay)

Arguments

Name	Description	Range	Type	Default	Required
vfund_ideal	single complex voltage spectral component (for example the fundamental) that is ideal (undistorted). This could be constructed from two baseband signals instead, by using the function cmplx().	(-∞, ∞)	complex		yes
vfund_dist	single complex voltage spectral component (for example, the fundamental) that has been distorted by the network being simulated. This could be constructed from two baseband signals instead, by using the function cmplx()	(-∞, ∞)	complex		yes
symbol_rate	symbol rate of the modulation signal	[0, ∞)	integer, real		yes
sampling_delay	sampling delay †	[0, ∞)	integer, real	0	no
rotation	parameter that rotates the constellations by that many radian ‡	[0, ∞)	integer, real	0	no

Name	Description	Range	Type	Default	Required
transient_duration	time in seconds that causes this time duration of symbols to be eliminated from the error-vector-magnitude calculation ‡ †	[0, ∞)	integer, real	0	no
path_delay	time in seconds of the sum of all delays in the signal path ‡ ‡	[0, ∞)	integer, real	0	no

† sampling_delay - (if nonzero) throws away the first delay = N seconds of data from the constellation and trajectory plots. It is also used to interpolate between simulation time points, which is necessary if the optimal symbol-sampling instant is not exactly at a simulation time point. Usually this parameter must be nonzero to generate a constellation diagram with the smallest grouping of sample points
‡ rotation does not need to be entered, and it will not affect the error-vector-magnitude calculation, because both the ideal and distorted constellations will be rotated by the same amount.
‡ † Usually the filters in the simulation have transient responses, and the error-vector-magnitude calculation should not start until these transient responses have finished.
‡ ‡ If the delay is 0, this parameter may be omitted. If it is non-zero, enter the delay value. This can be calculated by using the function delay_path().

Examples

```
rotation = -0.21
sampling_delay = 1/sym_rate[0, 0] - 0.5 * tstep[0, 0]
vfund_ideal = vOut_ideal[1]
vfund_dist = vOut_dist[1]
symbol_rate = sym_rate[0, 0]
data = const_evm(vfund_ideal, vfund_dist, symbol_rate, sampling_delay,
rotation, 1.5ms, path_delay)
```

where the parameter sampling_delay can be a numeric value, or in this case an equation using sym_rate, the symbol rate of the modulated signal, and tstep, the time step of the simulation. If these equations are to be used in a Data Display window, sym_rate and tstep must be defined by means of a variable (VAR) component, and they must be passed into the dataset as follows: Make the parameter Other visible on the Envelope simulation component, and edit the parameter so that:

```
Other = OutVar = sym_rate OutVar = tstep
```

In some cases, it may be necessary to experiment with the delay value to get the constellation diagrams with the tightest points.

Note that const_evm() returns a list of data. So in the example above,

```
data[0]= ideal constellation
data[1]= ideal trajectory
data[2]= distorted constellation
data[3]= distorted trajectory
data[4]= error vector magnitude versus time
data[5]= percent error vector magnitude
```

Refer to the example file to see how these data are plotted.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

constellation(), delay_path(), sample_delay_pi4dqpsk(), sample_delay_qpsk()

Notes/Equations

Used in constellation and trajectory diagram generation and error-vector-magnitude calculation.

The user must supply a single complex voltage spectral component (for example, the fundamental) that is ideal (undistorted), as well as a single complex voltage spectral component (for example, the fundamental) that has been distorted by the network being simulated. These ideal and distorted complex voltage waveforms could be generated from baseband I and Q data. The user must also supply the symbol rate, a delay parameter, a rotation factor, and a parameter to eliminate any turn-on transient from the error-vector-magnitude calculation are optional parameters.

The error vector magnitude is computed after correcting for the average phase difference and RMS amplitude difference between the ideal and distorted constellations.

cross_hist()

Returns jitter histogram

Syntax

y = cross_hist(Vout_time, time_start, time_stop, level_low, level_high, number_of_bins, BitRate, No_of_Eye, Delay, steps)

Arguments

Name	Description	Range	Type	Required
Vout_time	time domain voltage waveform	(-∞, ∞)	real	yes
time_start	define the rectangular window points for jitter histogram plot	[0, ∞)	real	yes
time_stop	define the rectangular window points for jitter histogram plot	[0, ∞)	real	yes
level_low	define the rectangular window points for jitter histogram plot	[0, ∞)	real	yes
level_high	define the rectangular window points for jitter histogram plot	[0, ∞)	real	yes
number_of_bins	defines the number of bins on the time axis of eye diagram and controls the resolution of jitter histogram plot	[1, ∞)	integer	yes
BitRate	bit rate of the channel and is expressed in frequency units	(0, ∞)	real	yes
No_of_Eye	Used for multiple eye jitter histogram plots	[0, ∞)	integer	yes

Name	Description	Range	Type	Required
Delay	used to remove initial transient in the eye diagram and is expressed in time units	[0, ∞)	real	yes
steps	represents the number of sampling points between level_low and level_high and is used for controlling the density of jitter histogram.	[1, ∞)	integer	yes

Example

```
Jitter_Histogram = cross_hist(vout, 0 ps, 100 ps, 0 V, 0.1V, 300, 10 GHz,
1,0, 20)
```

Defined in

Built in

See Also

eye_amplitude(), eye_closure(), eye_fall_time(),eye_rise_time(), eye_height()

Notes/Equations

Jitter histogram plots the jitter histogram of a time domain voltage waveform.

delay_path()

This function is used to determine the time delay and the constellation rotation angle between two nodal points along a signal path

Syntax

y = delay_path(vin, vout)

Arguments

Name	Description	Range	Type	Required
vin	envelope (I + j * Q) at the input node	(-∞, ∞)	complex	yes
vout	I + j * Q at the output node	(-∞, ∞)	complex	yes

Examples

```
x = delay_path(vin[1], vout[1])
```

where *vin[1]* and *vout[1]* are complex envelopes around the first carrier frequency in envelope simulation. In return, x[0] is the time delay (in seconds) between vin and vout. x[1] is the rotation angle (in radians) between vin and vout constellations.

or

```
x = delay_path(T1, T2)
```

where *T1* and *T2* are instance names of two TimedSink components.

Defined in

Built in

See Also

ber_pi4dqpsk(), ber_qpsk(), const_evm(), cross_corr()

Notes/Equations

Used in Circuit Envelope simulation, Ptolemy simulation.

This function outputs an array of two values. The first value, data[0], is the time delay between vin and vout. The second value, data[1], is the rotation angle between vin-constellation and vout-constellation.

evm_wlan_dsss_cck_pbcc()

Returns EVM (error vector magnitude) analysis results for WLAN DSSS/CCK/PBCC (IEEE 802.11b and IEEE 802.11g non-OFDM) voltage signals

Syntax

evm = evm_wlan_dsss_cck_pbcc(voltage{, mirrorSpectrum, start, averageType, burstsToAverage, modulationFormat, searchTime, resultLengthType, resultLength, measurementOffset, measurementInterval, chipRate, clockAdjust, equalizationFilter, filterLength, descrambleMode, referenceFilter, referenceFilterBT, output})

Arguments

Name	Description	Range	Type	Default	Required
voltage	complex envelope WLAN DSSS/CCK/PBCC voltage signal	(-∞, ∞)	complex		yes
mirrorSpectrum	specifies whether the input signal should be mirrored or not	See Notes	string	NO	no
start	specifies the start time for the EVM analysis	[0, max(indep(voltage))]	real	first point of the input data	no
averageType	specifies what type of averaging is done	See Notes	string	OFF	no
burstsToAverage	number of bursts over which the results will be averaged	(0, ∞)	integer	20	no
modulationFormat	modulation format	See Notes	string	"Auto Detect"	no
searchTime	search time	[0, max(indep(voltage))]	real	550 usec	no
resultLengthType	specifies how the result length is determined	See Notes	string	"Auto Select"	no
resultLength	result length in chips	[1, 108344]	integer	2816	no
measurementOffset	measurement offset in chips	(0, ∞)	integer	22	no

Name	Description	Range	Type	Default	Required
measurementInterval	measurement interval in chips	(0, ∞)	integer	2794	no
chipRate	chip rate in Hz	(0, ∞)	real	11e6 Hz	no
clockAdjust	clock adjustment as a fraction of a chip	[-0.5, 0.5]	real	0	no
equalizationFilter	specifies whether an equalization filter is used or not	See Notes	string	OFF	no
filterLength	equalization filter length in chips	[3, ∞) must be odd	integer	21	no
descrambleMode	specifies how descrambling is done	See Notes	string	On	no
referenceFilter	specifies the reference filter to be used	See Notes	string	Gaussian	no
referenceFilterBT	BT value for the reference filter if a Gaussian reference filter is selected	[0.05, 100]	real	0.3	no
output	EVM analysis result to be returned	See Notes	string	Avg_EVMrms_pct	no

Examples

```
evmRMS = evm_wlan_dsss_cck_pbcc(Vout[1])
```

where Vout is a named node in a Circuit Envelope simulation, will return the EVM rms value in percent for the voltage envelope at the fundamental frequency. The voltage data Vout[1] must contain at least one complete burst. The EVM result will be from the analysis of the first burst in the input signal.

```
evmPk = evm_wlan_dsss_cck_pbcc(Vout[1], , , “RMS (Video)”, 10, , 300e-6, , , , , , , , , “Gaussian”, 0.5, “EVM_Pk_pct”)
```

where Vout is a named node in a Circuit Envelope simulation, will return a vector with 10 values each representing the peak EVM in percent for the first 10 bursts in the voltage envelope at the fundamental frequency. Since searchTime is set to 300 µsec, the first 300 µsec of Vout[1] must contain at least one complete burst. In addition, since 10 bursts need to be processed, Vout[1] must contain at least 10 complete bursts. A Gaussian filter with BT=0.5 will be used as a reference filter.

```
AvgMagErr = evm_wlan_dsss_cck_pbcc(Vout[1], , , “RMS (Video)”, 3, , 400e-6, , , 100, 500, , , , , , , , “Avg_MagErr_rms_pct”)
```

where Vout is a named node in a Circuit Envelope simulation, will return the average (over 3 bursts) magnitude error in percent for the voltage envelope at the fundamental frequency. Since searchTime is set to 400 µsec, the first 400 µsec of Vout[1] must contain at least one complete burst. In addition, since 3 bursts need to be averaged, Vout[1] must contain at least 3 complete bursts. Only the chips 101 to 600 (measurementOffset = 100 and measurementInterval = 500) will be considered for the EVM analysis.

Defined in

Built in

See Also

evm_wlan_ofdm()

Notes/Equations

Used in Circuit Envelope simulation and Data Flow simulation.

This expression can be used with input data of up to two dimensions. Only complex envelope input signals are allowed as input.

The evm_wlan_dsss_cck_pbcc() expression performs an EVM measurement for WLAN DSSS/CCK/PBCC (IEEE 802.11b and IEEE 802.11g non-OFDM) signals. Table 4-1 displays the available measurement results.

Table 4-1. Available Measurement Results for evm_wlan_dsss_cck_pbcc()

Measurement Result	Description
Avg_WLAN_80211b_1000_chip_Pk_EVM_pct	average EVM in percentage as specified by the standard (section 18.4.7.8 *Transmit modulation accuracy* in 802.11b specification; pages 55-57) except that the EVM value is normalized
WLAN_80211b_1000_chip_Pk_EVM_pct	EVM in percentage as specified by the standard (section 18.4.7.8 *Transmit modulation accuracy* in 802.11b specification; pages 55-57) with the exception that the EVM value is normalized versus burst
Avg_EVMrms_pct	average EVM rms in percentage as defined in the Agilent 89600 VSA
EVMrms_pct	EVM rms in percentage as defined in the Agilent 89600 VSA versus burst
EVM_Pk_pct	peak EVM in percentage versus burst
EVM_Pk_chip_idx	peak EVM chip index versus burst

Table 4-1. Available Measurement Results for evm_wlan_dsss_cck_pbcc()

Measurement Result	Description
Avg_MagErr_rms_pct	average magnitude error rms in percentage
MagErr_rms_pct	magnitude error rms in percentage versus burst
MagErr_Pk_pct	peak magnitude error in percentage versus burst
MagErr_Pk_chip_idx	peak magnitude error chip index versus burst
Avg_PhaseErr_deg	average phase error in degrees
PhaseErr_deg	phase error in degrees versus burst
PhaseErr_Pk_deg	peak phase error in degrees versus burst
PhaseErr_Pk_chip_idx	peak phase error chip index versus burst
Avg_FreqError_Hz	average frequency error in Hz
FreqError_Hz	frequency error in Hz versus burst
Avg_IQ_Offset_dB	average IQ offset in dB
IQ_Offset_dB	IQ offset in dB versus burst
Avg_SyncCorrelation	average sync correlation
SyncCorrelation	sync correlation versus burst

Results whose name is prefixed with “Avg_” are averaged over the number of bursts specified by the user (if averageType is set to “RMS (Video)”). Results whose name is not prefixed with “Avg_” are results versus burst.

The following is a brief description of the algorithm used (the algorithm used is the same as the one used in the Agilent 89600 VSA) and a detailed description of its arguments.

Starting at the time instant specified by the start argument, a signal segment of length searchTime is acquired. This signal segment is searched in order for a complete burst to be detected. The burst search algorithm looks for both a burst on and a burst off transition. In order for the burst search algorithm to detect a burst, an idle part must exist between consecutive bursts and the bursts must be at least 15 dB above the noise floor.

If the acquired signal segment does not contain a complete burst, the algorithm will not detect any burst and the analysis that follows will most likely produce wrong results. Therefore, searchTime must be long enough to acquire at least one complete burst. Since the time instant specified by the start argument can be a little after the beginning of a burst, it is recommended that searchTime is set to a value

approximately equal to 2 x burstLength, where burstLength is the duration of a burst in seconds including the duration of the idle part. If it is known that the time instant specified by the start argument is a little before the beginning of a burst, then searchTime can be set to burstLength.

After a burst is detected, synchronization is performed based on the preamble. The burst is then demodulated. Finally, the burst is analyzed to get the EVM measurement results.

If averageType is set to OFF, only one burst is detected, demodulated, and analyzed.

If averageType is set to RMS (Video), after the first burst is analyzed the signal segment corresponding to it is discarded and new signal samples are acquired to fill in the signal buffer of length searchTime. When the buffer is full again a new burst search is performed and when a burst is detected it is demodulated and analyzed. These steps repeat until burstsToAverage bursts are processed.

If for any reason a burst is misdetected the results from its analysis are discarded. The EVM results obtained from all the successfully detected, demodulated, and analyzed bursts are averaged to give the final result.

The mirrorSpectrum argument accepts the following strings: “NO” and “YES”. This argument can be used to mirror (conjugate) the input signal before any other processing is done. Mirroring the input signal is necessary if the configuration of the mixers in your system has resulted in a mirrored signal compared to the one at the input of the up-converter and if the preamble and header are short format. In this case, if mirrorSpectrum is not set to “YES” the header bits (which carry the modulation format and length information) will not be recovered correctly so the demodulation of the PSDU part of the burst will most likely fail.

The start argument sets the starting point for acquiring the signal to be processed. By default, the starting point is the beginning of the input signal (voltage argument). However, if for any reason an initial part of the input signal needs to be omitted this can be done by setting the start argument appropriately.

The averageType argument accepts the following strings: “Off” and “RMS (Video)”. This argument can be used to turn on/off video averaging. If set to “Off” the EVM result returned is from the processing of only one burst. Otherwise, multiple bursts are processed and the results are averaged.

The burstsToAverage argument set the number of bursts whose results will be averaged if averageType is set to “RMS (Video)”. If averageType is set to “Off” this argument is ignored.

The modulationFormat argument accepts the following strings: “Auto Detect”, “Barker 1”, “Barker 2”, “CCK 5.5”, “CCK 11”, “PBCC 5.5”, “PBCC 11”, “PBCC 22”, “PBCC 33”. This argument sets the modulation format used in the PSDU part of the burst. If modulationFormat is set to “Auto Detect”, the algorithm will use the information detected in the PLCP header part of the burst to automatically determine the modulation format. Otherwise, the modulation format determined from the PLCP header is ignored and the modulation format specified by the modulationFormat argument is used in the demodulation of the PSDU part of the burst.

The searchTime argument sets the duration of the signal segment that is acquired and searched in order to detect a complete burst. Recommendations on how to set this argument are given in the brief description of the algorithm used by this expression earlier in these Notes/Equations section.

The resultLengthType argument accepts the following strings: “Auto Select” and “Manual Override”. The resultLengthType and resultLength arguments control how much data is acquired and demodulated.

- When resultLengthType is set to “Auto Select”, the measurement result length is automatically determined from the information in the PLCP header part of the burst. In this case, the argument resultLength defines a maximum result length for the burst in symbol times; that is, if the measurement result length that is automatically detected is bigger than resultLength it will be truncated to resultLength. The maximum result length specified by the resultLength argument includes the PLCP preamble and PLCP header.
- When resultLengthType is set to “Manual Override”, the measurement result length is set to resultLength regardless of what is detected in the PLCP header part of the burst. The result length specified by the resultLength argument includes the PLCP preamble and PLCP header.

Table 4-2 summarizes the differences between how “Auto Select” and “Manual Override” modes determine the measurement result length. The table lists the measurement result lengths actually used for “Auto Select” and “Manual Override” modes for three different values of the resultLength argument (3300, 2816 and 2200 chips). It is assumed that the input burst is 2816 symbols long.

Table 4-2. Measurement result length setting

resultLengthType	resultLength	Measurement Result Length Actually Used
Auto Select	2200	2200
Auto Select	2816	2816
Auto Select	3300	2816
Manual Override	2200	2200
Manual Override	2816	2816
Manual Override	3300	3300

Note that when resultLengthType is set to “Manual Override” and resultLength=3300 (greater than the actual burst size) the algorithm will demodulate the full 3300 chips even though this is 484 chips beyond the burst width.

The measurementOffset and measurementInterval arguments can be used to isolate a specific segment of the burst for analysis. The values of measurementInterval and measurementOffset are in number of chips and are relative to the ideal starting point of the PLCP preamble portion of the burst. For a signal that uses the long PLCP format, the ideal starting point of the PLCP preamble is exactly 128 symbol times (128 x 11 chips) before the start of the SFD sync pattern. For a signal that uses the short PLCP format, the ideal starting point of the PLCP preamble is exactly 56 symbol times (56 x 11 chips) before the start of the SFD sync pattern.

The chipRate argument sets the fundamental chip rate of the signal to be analyzed. The default is 11 MHz, which matches the chip rate of 802.11b and 802.11g; however, this argument can be used when experimenting with signals that do not follow the standard specifications. A special case is the optional 802.11g 33 Mbit PBCC mode, where the chip rate of the transmitted signal starts at 11 MHz, but changes to 16.5 MHz in the middle of the burst. In this case chipRate should still be set to 11 MHz (the algorithm will automatically switch to 16.5 MHz at the appropriate place in the burst).

Although the algorithm synchronizes to the chip timing of the signal, it is possible for the synchronization to be slightly off. The clockAdjust argument allows the user to specify a timing offset which is added to the chip timing detected by the algorithm. This argument should only be used when trying to debug unusual signals.

The equalizationFilter argument accepts the following strings: “OFF” and “ON”. This argument can be used to turn on/off the equalization filter. The filterLength

argument sets the equalization filter length (in number of chips). Using an equalization filter can dramatically improve the EVM results since the equalizer can compensate for ISI due to the transmit filter. However, it can also compensate the distortion introduced by the DUT. If the filter used in the transmitter is Gaussian, then having the equalizer off and selecting a Gaussian reference filter might be a better option.

The descrambleMode argument accepts the following strings: “Off”, “Preamble Only”, “Preamble & Header Only”, “On”. This argument can be used to control how descrambling is done.

- “Off” does no descrambling.
- “Preamble Only” descrambles only the PLCP preamble.
- “Preamble & Header Only” descrambles only the PLCP preamble and PLCP header.
- “On” descrambles all parts of the burst.

Normally, 802.11b or 802.11g signals have all bits scrambled before transmission, so this parameter should normally be set to “On”. However, when debugging an 802.11b or 802.11g transmitter, it is sometimes helpful to disable scrambling in the transmitter, in which case you should disable descrambling in this component.

If the input signal’s preamble is scrambled but you disable descrambling of the preamble (or vice versa), then the algorithm will not be able to synchronize to the signal properly. Similarly, if the input signal’s header is scrambled but you disable descrambling of the header (or vice versa) then the algorithm will not be able to correctly identify the burst modulation type and burst length from the header.

The referenceFilter argument accepts the following strings: “Rectangular” and “Gaussian”. This argument can be used to select a reference filter for the EVM analysis. Although, the IEEE 802.11b/g standards do not specify either a transmit filter or a receive filter, they do have a spectral mask requirement, and a transmitter must use some sort of transmit filter to meet the spectral mask. On the other hand, the description of the EVM measurement in the standard does not use any receive or measurement filter. The absence of the need to use any transmit or receive filter is partly because the standard has a very loose limit for EVM (35% peak on 1000 chips worth of data).

If the standard definition is followed when computing EVM, no measurement or reference filter should be used (referenceFilter must be set to “Rectangular”). However, what this means is that even a completely distortion-free input signal will

still give non-zero EVM unless the input signal has a zero-ISI transmit filter. If a non-zero-ISI transmit filter is used and there is additional distortion added to the signal due to the DUT, then the EVM will measure the overall error due to both the transmit filter's ISI and the DUT distortion. Turning on the equalizer will remove most of the transmit filter's ISI but it can also remove some of the distortion introduced by the DUT. To get a better idea of the EVM due to the DUT distortion a reference filter that matches the transmit filter can be used. Currently, only "Rectangular" and "Gaussian" filters are available as reference filters.

The referenceFilterBT argument sets the BT (Bandwidth Time product) for the Gaussian reference filter. If referenceFilter is set to "Rectangular" this argument is ignored.

The output argument accepts the following strings (see Table 4-1): "Avg_WLAN_80211b_1000_chip_Pk_EVM_pct", "WLAN_80211b_1000_chip_Pk_EVM_pct", "Avg_EVMrms_pct", "EVMrms_pct", "EVM_Pk_pct", "EVM_Pk_chip_idx", "Avg_MagErr_rms_pct", "MagErr_rms_pct", "MagErr_Pk_pct", "MagErr_Pk_chip_idx". "Avg_PhaseErr_deg", "PhaseErr_deg", "PhaseErr_Pk_deg", "PhaseErr_Pk_chip_idx", "Avg_FreqError_Hz", "FreqError_Hz", "Avg_IQ_Offset_dB", "IQ_Offset_dB", "Avg_SyncCorrelation", "SyncCorrelation". This argument selects which EVM analysis result will be returned.

Table 4-3 summarizes how some of the arguments of the evm_wlan_dsss_cck_pbcc() expression should be set based on the parameter values of the WLAN_802_11b source.

Table 4-3. Relationship Between WLAN_802_11b Source Parameters and evm_wlan_dsss_cck_pbcc() Expression Arguments

WLAN_802_11b	evm_wlan_dsss_cck_pbcc()	Comments
DataRate (default is 11 Mbps)	searchTime (default is 550 µsec)	The recommended searchTime is $2 \times \text{BurstLength}$. BurstLength= tRamp+tPLCP+tPSDU+IdleInterval, where, tRamp = $\begin{Bmatrix} 0, \text{PwrRamp} = \text{None} \\ 4\ \mu sec, \text{otherwise} \end{Bmatrix}$ tPLCP = $\begin{Bmatrix} 192\ \mu sec, \text{PreambleFormat} = \text{Long} \\ 96\ \mu sec, \text{PreambleFormat} = \text{Short} \end{Bmatrix}$ tPSDU = $\text{DataLength} \times 8 / \text{DataRate}$
PreambleFormat (default is Long)		
PwrRamp (default is None)		
IdleInterval (default is 10 µsec)		
DataLength (default is 100)		
MirrorSpectrum (default is NO)	mirrorSpectrum (default is NO)	If DUT introduces spectrum mirroring, then mirrorSpectrum must be set to “NO” (“YES”) when MirrorSpectrum is set to “YES” (“NO”); otherwise mirrorSpectrum must be set to the same value as MirrorSpectrum.
FilterType (default is Gaussian)	referenceFilter (default is Gaussian)	When FilterType is set to “Gaussian”, the recommended setting of referenceFilter is “Gaussian”; otherwise, the recommended setting is “Rectangular”
GaussianFilter_bT (default is 0.3)	referenceFilterBT (default is 0.3)	The recommended setting of referenceFilterBT is the same value as GaussianFilter_bT. This parameter is only used when referenceFilter is set to “Gaussian”.

evm_wlan_ofdm()

Returns EVM (error vector magnitude) analysis results for WLAN OFDM (IEEE 802.11a) voltage signals

Syntax

evm = evm_wlan_ofdm(voltage{, mirrorSpectrum, start, averageType, burstsToAverage, subcarrierModulation, guardInterval, searchTime, resultLengthType, resultLength, measurementOffset, measurementInterval, subcarrierSpacing, symbolTimingAdjust, sync, output})

Arguments

Name	Description	Range	Type	Default	Required
voltage	complex envelope WLAN OFDM (orthogonal frequency division multiplexing) voltage signal	(-∞, ∞)	complex		yes
mirrorSpectrum	specifies whether the input signal should be mirrored or not	See Notes	string	NO	no
start	specifies the start time for the EVM analysis	[0, max(indep(voltage))]	real	first point of the input data	no
averageType	specifies what type of averaging is done	See Notes	string	OFF	no
burstsToAverage	number of bursts over which the results will be averaged	(0, ∞)	integer	20	no
subcarrierModulation	data subcarrier modulation format	See Notes	string	"Auto Detect"	no
guardInterval	guard interval length for the OFDM symbols (as a fraction of the FFT time period)	[0, 1]	real	0.25	no
searchTime	search time	[0, max(indep(voltage))]	real	80 usec	no
resultLengthType	specifies how the result length is determined	See Notes	string	"Auto Select"	no

Name	Description	Range	Type	Default	Required
resultLength	result length in OFDM symbols	[1, 1367]	integer	60	no
measurementOffset	measurement offset in OFDM symbols	[0, ∞)	integer	0	no
measurementInterval	measurement interval in OFDM symbols	(0, ∞)	integer	11	no
subcarrierSpacing	frequency spacing between the subcarriers	(0, ∞)	real	312.5 kH	no
symbolTimingAdjust	specifies (as a percent of the FFT time period) the timing adjustment done on the OFDM symbols before performing the FFT	[-100*guardInterval, 0]	real	-3.125	no
sync	preamble sequence that will be used for synchronization	See Notes	string	Short Training Seq	no
output	EVM analysis result to be returned	See Notes	string	EVMrms_percent	no

Examples

```
evmRMS = evm_wlan_ofdm(Vout[1])
```

where Vout is a named node in a Circuit Envelope simulation, will return the evm rms value in percent for the voltage envelope at the fundamental frequency. The voltage data Vout[1] must contain at least one complete OFDM burst.

```
iqOffset = evm_wlan_ofdm( Vout[1], , , “RMS (Video)”, 5, , 0.125, 200e-6, , , 5, 10, , , , “IQ_Offset_dB”)
```

where Vout is a named node in a Circuit Envelope simulation, will return the IQ offset in dB for the voltage envelope at the fundamental frequency. Five bursts will be analyzed and their results averaged. The guard interval used in the generation of the input signal must be 0.125. Since searchTime is set to 200 µsec, the first 200 µsec of Vout[1] must contain at least one complete OFDM burst. In addition, since 5 bursts need to be averaged Vout[1] must contain at least 5 complete OFDM bursts. Only the OFDM symbols 6 to 15 (measurementOffset = 5 and measurementInterval = 10) will be considered for the EVM analysis.

Defined in

Built in

See Also

evm_wlan_dsss_cck_pbcc()

Notes/Equations

Used in Circuit Envelope simulation and Data Flow simulation.

This expression can be used with input data of up to two dimensions. Only complex envelope input signals are allowed as input.

The evm_wlan_ofdm() expression performs an EVM measurement for WLAN OFDM (IEEE 802.11a) signals. Table 4-4 displays the available measurement results.

Table 4-4. Available Measurement Results for evm_wlan_ofdm()

Measurement Result	Description
EVMrms_percent	average EVM rms in percentage
EVM_dB	average EVM in dB
PilotEVM_dB	average pilot EVM in dB
CPErms_percent	average Common Pilot Error rms in percentage
IQ_Offset_dB	average IQ offset in dB
SyncCorrelation	average sync correlation

The following is a brief description of the algorithm used (the algorithm used is the same as the one used in the Agilent 89600 VSA) and a detailed description of its arguments. Figure 4-1 shows the structure of an OFDM burst. Many of the terms mentioned later in these notes such as the preamble, SIGNAL symbol, DATA symbols, guard intervals (GI) are shown in this figure.

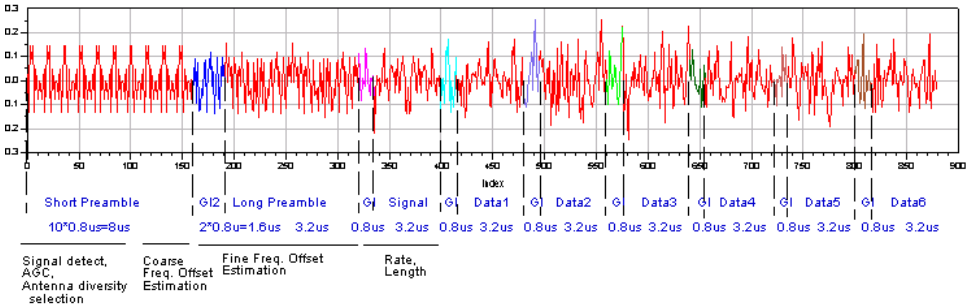


Figure 4-1. Structure of an OFDM burst.

Starting at the time instant specified by the start argument, a signal segment of length searchTime is acquired. This signal segment is searched in order for a complete burst to be detected. The burst search algorithm looks for both a burst on and a burst off transition. In order for the burst search algorithm to detect a burst, an idle part must exist between consecutive bursts and the bursts must be at least 15 dB above the noise floor.

If the acquired signal segment does not contain a complete burst, the algorithm will not detect any burst and the analysis that follows will most likely produce incorrect results. Therefore, searchTime must be long enough to acquire at least one complete burst. Since the time instant specified by the start argument can be a little after the beginning of a burst, it is recommended that searchTime is set to a value approximately equal to 2 x burstLength, where burstLength is the duration of a burst in seconds including the duration of the idle part. If it is known that the time instant specified by the start argument is a little before the beginning of a burst, then searchTime can be set to burstLength.

After a burst is detected, synchronization is performed based on the value of the sync argument. The burst is then demodulated. Finally, the burst is analyzed to get the EVM measurement results.

If averageType is set to Off, only one burst is detected, demodulated, and analyzed.

If averageType is set to RMS (Video), after the first burst is analyzed the signal segment corresponding to it is discarded and new signal samples are acquired to fill in the signal buffer of length searchTime. When the buffer is full again a new burst search is performed and when a burst is detected it is demodulated and analyzed. These steps repeat until burstsToAverage bursts are processed.

If for any reason a burst is mis-detected, the results from its analysis are discarded. The EVM results obtained from all the successfully detected, demodulated, and analyzed bursts are averaged to give the final result.

The mirrorSpectrum argument accepts the following strings: “NO” and “YES”. This argument can be used to mirror (conjugate) the input signal before any other processing is done. Mirroring the input signal is necessary if the configuration of the mixers in your system has resulted in a mirrored signal compared to the one at the input of the up-converter. The demodulation process recovers a lot of the information about the burst from the burst preamble and SIGNAL symbol. If the input signal is mirrored, then some of this information may not be recovered correctly and the demodulation will most likely fail.

The start argument sets the starting point for acquiring the signal to be processed. By default, the starting point is the beginning of the input signal (voltage argument). However, if for any reason an initial part of the input signal needs to be omitted this can be done by setting the start argument appropriately.

The averageType argument accepts the following strings: “Off” and “RMS (Video)”. This argument can be used to turn on/off video averaging. If set to “Off” the EVM result returned is from the processing of only one burst. Otherwise, multiple bursts are processed and the results are averaged.

The burstsToAverage argument set the number of bursts whose results will be averaged if averageType is set to “RMS (Video)”. If averageType is set to “Off” this argument is ignored.

The subcarrierModulation argument accepts the following strings: “Auto Detect”, “BPSK”, “QPSK”, “QAM 16”, and “QAM 64”. This argument sets the data subcarrier modulation format. If subcarrierModulation is set to “Auto Detect”, the algorithm will use the information detected within the OFDM burst (SIGNAL symbol - RATE data field) to automatically determine the data subcarrier modulation format. Otherwise, the format determined from the OFDM burst will be ignored and the format specified by the subcarrierModulation argument will be used in the demodulation for all data subcarriers. This argument has no effect on the demodulation of the pilot subcarriers and the SIGNAL symbol, whose format is always BPSK.

The guardInterval argument sets the guard interval (also called cyclic extension) length for the OFDM symbols. The value is expressed as a fraction of the FFT time period and so its valid range is [0, 1]. The value must match the guard interval length actually used in the generation of the input signal in order for the demodulation to work properly.

The searchTime argument sets the duration of the signal segment that is acquired and searched in order to detect a complete OFDM burst. Recommendations on how to set this argument are given in the brief description of the algorithm used by this expression earlier in these Notes/Equations section.

The resultLengthType argument accepts the following strings: “Auto Select” and “Manual Override”. The resultLengthType and resultLength arguments control how much data is acquired and demodulated.

- When resultLengthType is set to “Auto Select”, the measurement result length is automatically determined from the information in the decoded SIGNAL symbol (LENGTH data field). In this case, the argument resultLength defines a maximum result length for the burst in symbol times; that is, if the measurement result length that is automatically detected is bigger than resultLength it will be truncated to resultLength.
- When resultLengthType is set to “Manual Override”, the measurement result length is set to resultLength regardless of what is detected from the SIGNAL symbol of the burst. The value specified in resultLength includes the SIGNAL symbol but does not include any part of the burst preamble.

Table 4-5 summarizes the differences between how “Auto Select” and “Manual Override” modes determine the measurement result length. The table lists the measurement result lengths actually used for “Auto Select” and “Manual Override” modes for three different values of the resultLength argument (30, 26 and 20 symbols). It is assumed that the input burst is 26 symbols long.

Table 4-5. Measurement Result Length Setting

resultLengthType	resultLength	Measurement Result Length Actually Used
Auto Select	20	20
Auto Select	26	26
Auto Select	30	26
Manual Override	20	20
Manual Override	26	26
Manual Override	30	30

Note that when resultLengthType is set to “Manual Override” and resultLength=30 (greater than the actual burst size) the algorithm will demodulate the full 30 symbols even though this is 4 symbols beyond the burst width.

The measurementOffset and measurementInterval arguments can be used to isolate a specific segment of the burst for analysis. Both are expressed in number of OFDM symbols. The offset starts counting from the SIGNAL symbol. An offset of 0 will include the SIGNAL symbol in the EVM analysis, an offset of 1 will exclude the SIGNAL symbol, and an offset of 5 will exclude the SIGNAL symbol as well as the first 4 DATA symbols. Figure 4-2 shows the interrelationship between searchTime, resultLength, measurementInterval, and measurementOffset.

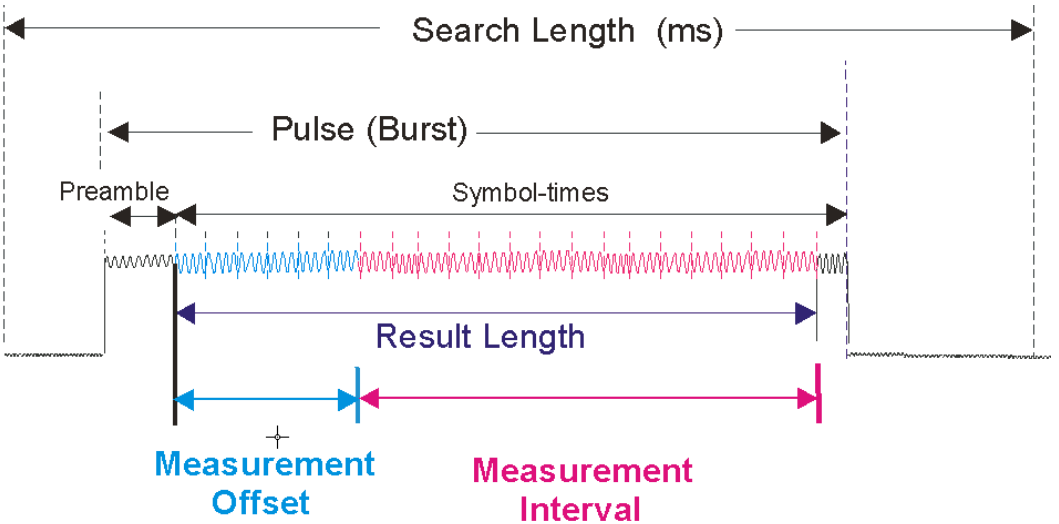


Figure 4-2. Interrelationship between searchTime, resultLength, measurementInterval, and measurementOffset.

The subcarrierSpacing argument sets the frequency spacing between the subcarriers of the OFDM signal. The value must match the subcarrier spacing actually used in the generation of the input signal in order for the demodulation to work properly.

The symbolTimingAdjust argument sets the timing adjustment done on the OFDM symbols before performing the FFT. The value is expressed as a percent of the FFT time period. Its valid range is [-100*guardInterval, 0]. Normally, when demodulating an OFDM symbol, the guard interval is skipped and an FFT is performed on the last portion of the symbol. However, this means that the FFT will include the transition region between this symbol and the following symbol. To avoid this, it is generally beneficial to back away from the end of the symbol and use part of the guard interval. The symbolTimingAdjust argument controls how far the FFT part of the symbol is adjusted away from the end of the symbol. Note that the value of this argument is

negative because the FFT start time is moved back by the amount specified by it. Figure 4-3 explains this concept graphically. When setting this argument, care should be taken to not back away from the end of the symbol too much because this may make the FFT include corrupt data from the transition region at the beginning of the symbol.

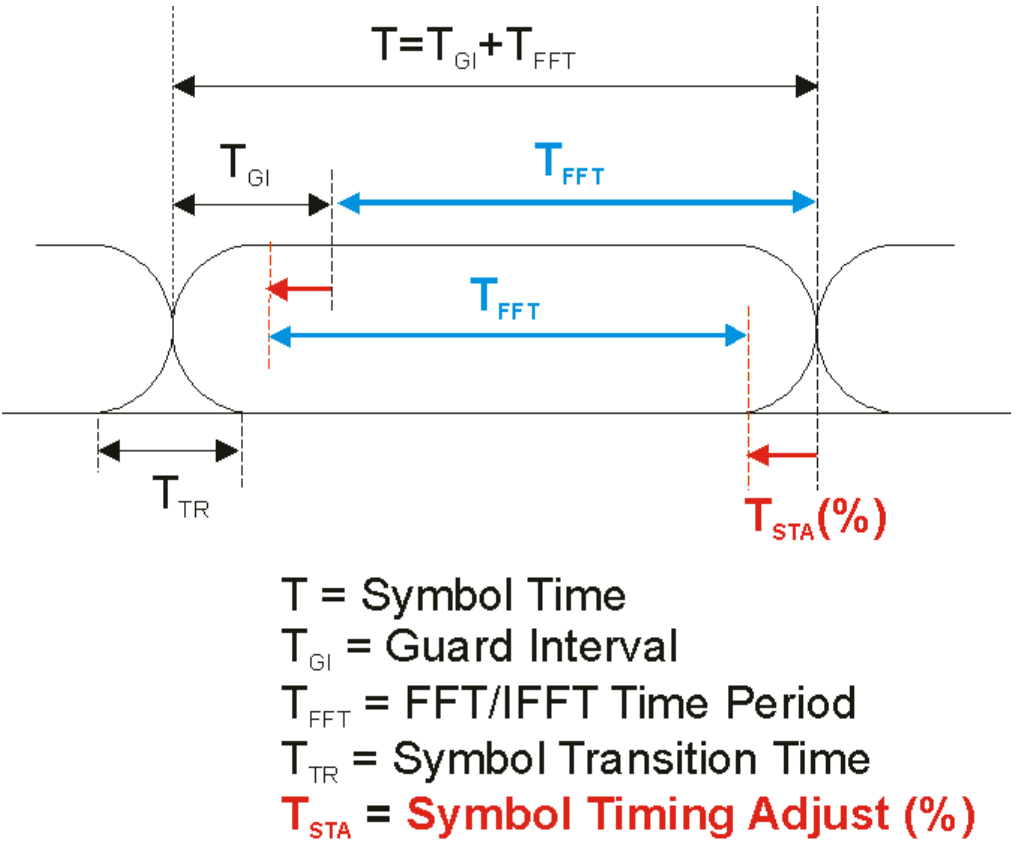


Figure 4-3. Definition of symbolTimingAdjust.

The sync argument accepts the following strings: “Short Training Seq”, “Channel Estimation Seq”. This argument determines which preamble sequence will be used for synchronization.

The output argument accepts the following strings (see Table 4-4): “EVMrms_percent”, “EVM_dB”, “PilotEVM_dB”, “CPErms_percent”, “IQ_Offset_dB”,

“SyncCorrelation”. This argument selects which EVM analysis result will be returned.

Table 4-6 summarizes how some of the arguments of the evm_wlan_ofdm() expression should be set based on the parameter values of the WLAN_802_11a source.

Table 4-6. Relationship Between WLAN_802_11a Source Parameters and evm_wlan_ofdm() Expression Arguments

WLAN_802_11a	evm_wlan_ofdm()	Comments
DataRate (default is 54 Mbps)	searchTime (default is 80 µsec)	The recommended searchTime is $2 \times \text{BurstLength}$. BurstLength= {[5 + (1 + OFDMSymbolsPerBurst) * (1 + GuardInterval)] * 64} / Bandwidth + IdleInterval, where, OFDMSymbolsPerBurst = ceil[(22 + 8 * DataLength) * 250000 / DataRate]
Bandwidth (default is 20 MHz)		
IdleInterval (default is 4 µsec)		
DataLength (default is 100)		
GuardInterval (default is 0.25)		
MirrorSpectrum (default is NO)	mirrorSpectrum (default is NO)	If DUT introduces spectrum mirroring, then mirrorSpectrum must be set to “NO” (“YES”) when MirrorSpectrum is set to “YES” (“NO”); otherwise mirrorSpectrum must be set to the same value as MirrorSpectrum.
Bandwidth (default is 20 MHz)	subcarrierSpacing (default is 312.5 kHz)	subcarrierSpacing must be set to Bandwidth/64.
GuardInterval (default is 0.25)	guardInterval (default is 0.25)	guardInterval must be set to the same value as GuardInterval.

fs()

Performs a time-to-frequency transform

Syntax

y = fs(x, fstart, fstop, numfreqs, dim, windowType, windowConst, tstart, tstop, interpOrder, transformMethod)

Arguments

Name	Description	Range	Type	Default	Required
x	Time-domain data to be transformed	(-∞, ∞)	real		yes
fstart	starting frequency	[0, ∞)	real	0 †	no
fstop	stoping frequency	[0, ∞)	real	1/(2*newdeltat) †	no
numfreqs	number of frequencies	[1, ∞)	integer	(fstop-fstart)*(tstop-tstart)+1	no
dim	dimension to be transformed (not used currently)	[1, ∞)	integer	highest dimension	no
windowType	type of window to be applied to the data	[0, 9] ‡	integer, string	0	no
windowConst	window constant ‡ †	[0, ∞)	integer, real	0	no
tstart	start time ‡ ‡	[0, ∞)	integer, real	first time point in given data	no
tstop	stop time ‡ ‡	[0, ∞)	integer, real	last time point in given data	no
interpOrder	Interpolation Order	[1, 3] ‡ ‡ †	integer	1	no

Name	Description	Range	Type	Default	Required
transformMethod	Transformation method	[1, 3] ‡ ‡ ‡	integer	1	no

† If data is real, fstart = 0, fstop=1/(2*newdeltat). If data is complex, fstart = -1/(time[startIndex+2]-time[stopIndex]) and fstop=1/(time[startIndex+2]-time[stopIndex]). Where newdeltat is the new uniform timestep of the resampled data, and startIndex and stopIndex are the index of tstart and tstop.

‡ The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the time-to-frequency transformation with normal gate shape setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13

‡ † windowConst is not used if windowType is 8510

‡ ‡ If tstart or tstop lies between two time points in the data, then the values are interpolated for these values.

‡ ‡ † If the tranorder variable is not present, or if the user wishes to override the interpolation scheme, then interpOrder may be set to a nonzero value:
1 = use only linear interpolation
2 = use quadratic interpolation
3 = use cubic polynomial interpolation

‡ ‡ ‡ The time-to-frequency transform can be changed by using transformMethod:

1 = Chirp-z transform
2 = Discrete Fourier integral evaluated at each frequency
3 = Fast Fourier transform

Examples

The following example equations assume that a transient simulation was performed from 0 to 5 ns with 176 timesteps, on a 1-GHz-plus-harmonics signal called vOut:

```
y = fs(vOut)
```
returns the spectrum (0, 0.2GHz, ... , 17.6GHz), evaluated from 0 to 5 ns.

```
y = fs(vOut, 0, 10GHz)
```
returns the spectrum (0, 0.2GHz, ... , 10.0GHz), evaluated from 0 to 5 ns.

```
y = fs(vOut, 0, 10GHz, 11)
```
returns the spectrum (0, 1.0GHz, ... , 10.0GHz), evaluated from 0 to 5 ns.

```
y = fs(vOut, , , , , , , 3ns, 5ns)
```
returns the spectrum (0, 0.5GHz, ... , 32.0GHz), evaluated from 3 to 5 ns.

```
y = fs(vOut, 0, 10GHz, 21, , , 3ns, 5ns)
```
returns the spectrum (0, 0.5GHz, ... , 10.0GHz), evaluated from 3 to 5 ns.

```
y=fs(vOut, 0, 10GHz, 11, , "Blackman")
```
returns the spectrum (0, 1.0GHz, ... , 10.0GHz), evaluated from 0 to 5 ns after a Blackman window is applied.

Defined in

Built in

See Also

fft(), fspot()

Notes/Equations

The dim argument is not used and should be left empty in the expression. Entering a value will have no impact on the results.

fs(x) returns the frequency spectrum of the vector x by using a chirp-z transform. The values returned are peak, complex values. The data to be transformed is typically from a transient, signal processing, or envelope analysis.

Transient simulation uses a variable timestep and variable order algorithm. The user sets an upper limit on the allowed timestep, but the simulator will control the timestep so that the local truncation error of the integration is also controlled. The non-uniformly sampled data are uniformly resampled for fs.

If the Gear integration algorithm is used, the order can also change during simulation. fs can use this information when resampling the data. This variable order integration depends on the presence of a special dependent variable, tranorder, which is output by the transient simulator. When the order varies, the Fourier integration will adjust the order of the polynomial it uses to interpolate the data between timepoints.

Only polynomials of degree one to three are supported. The polynomial is fit from the timepoint in question backwards over the last n points. This is because time-domain

data are obtained by integrating forward from zero; previous data are used to determine future data, but future data can never be used to modify past data.

The data are uniformly resampled, with the number of points being determined by increasing the original number of points to the next highest power of two.

The data to be transformed default to all of the data. The user may specify tstart and tstop to transform a subset of the data.

The starting frequency defaults to 0 and the stopping frequency defaults to 1/(2*newdeltat), where newdeltat is the new uniform timestep of the resampled data. The number of frequencies defaults to (fstop-fstart)*(tstop-tstart)+1. The user may change these by using fstart, fstop, and numfreqs. Note that numfreqs specifies the number of frequencies, not the number of increments. Thus, to get frequencies at (0, 1, 2, 3, 4, 5), numfreqs should be set to 6, not 5.

When the data to be operated on is of the baseband type, such as VO[0] from a Circuit Envelope analysis, where VO is an output node voltage and [0] is index for DC, then in order to obtain a single sided spectrum, only the real part of VO[0] should be used as the argument. i.e., x=fs(real(VO[0],...). This is necessary because the fs() function has no way of knowing the data VO[0] is baseband. Even though VO[0] contains an imaginary part of all zeroes, it is still represented by a complex data type. When the first argument of fs() is complex, the result will be a double-sided spectrum by default.

An alternative method of obtaining a single-sided spectrum from the above baseband data is to specify the frequencies ranges in the spectrum, using the fstart, fstop, and numfreqs parameters of the fs() function.

For example, y=fs(VO[0], 0, 25e3, 251). This will yield a spectrum from 0 to 25 kHz with 26 frequencies and 1 kHz spacing.

This does not apply to data from Transient analysis or Ptolemy simulation because voltage data from Transient and baseband data from Ptolemy are real.

For Envelope analysis, the transform is centered around the fundamental tone. For Signal Processing analysis, it is centered around the characterization frequency.

Differences Between the fs() and fft() Functions

For a periodic signal from t=0 to t=per, fs() requires the full data [0,per], where [] means 0 <= t <= per. The fft is defined as needing data [0,per), meaning 0 <= t < per. The last point at per should be the same as value at zero. fs() requires this point, fft() does not. That is why fs() really requires 2^{n+1} points and fft() requires 2^{n}. So for using

the FFT option in the fs() function, there should be 2^{n+1} points. However, for the default Chirp Z-transform option, any number of points will work.

Conditions under which fs() may not work properly:

1. When deltaF equals 0. Where deltaF = (Fstop - Fstart)/(numFreqs).
2. When 1.0/deltaF > Tstop - Tstart. In this case there are not enough time data for requested frequency resolution.
3. When 2*numFreqs > numDataPts. In this case there are not enough data points for frequencies.

peak_pwr()

Returns the peak power of the input voltage data

Syntax

peakP = peak_pwr(voltage, refR, percent, unit)

Arguments

Name	Description	Range	Type	Default	Required
voltage	baseband or complex envelope voltage signal	(-∞, ∞)	integer, real, complex		yes
refR	reference resistance in Ohms	(0, ∞)	real	50.0	no
percent	percentage of time the returned power value is exceeded	[0, 100]	real	0.0	no
unit	power unit to be used	"W"|"dBm"|"dBW"	string	"W"	no

Examples

```
peakP = peak_pwr(Vout[1], 100, , “dBW”)
```

where Vout is a named node in a Circuit Envelope simulation, will return the peak power (in dBW) at the fundamental frequency using 100 Ohms as reference resistance.

```
peakP = peak_pwr(T1, , 5)
```

where T1 is the name of a TimedSink component (in a DSP schematic), will return the power level (in W) that is exceeded 5% of the time for the voltage signal recorded in the TimedSink using 50 Ohms as reference resistance. If the signal recorded by the TimedSink is complex envelope Gaussian noise with a standard deviation of 30 mV for each of the I and Q envelopes, then peakP will be very close to 5.39e-5 W ($-\ln(0.05) / 0.03^2 / 50$).

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

peak_to_avg_pwr(), power_ccdf(), pwr_vs_t(), total_pwr()

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.

This expression can be used with input data (voltage) of any dimensions when the percent argument is 0 and up to three dimensions when the percent argument is greater than 0. It can handle both baseband as well as complex envelope data.

When the percent argument is set to 0, the returned value is the maximum instantaneous power of the input voltage signal. When the percent argument is set to a value greater than 0, the returned value is the power level that is exceeded for percent amount of time. This argument is useful since some wireless standards specify the peak power not as the absolute maximum instantaneous power but as the power level that is exceeded for some percentage of time. For example, the 3GPP standard defines the maximum power as the power level that is exceeded for 1% of the time.

peak_to_avg_pwr()

Returns the peak to average power ratio of the input voltage data

Syntax

peak_avg_ratio = peak_to_avg_pwr(voltage, percent, unit)

Arguments

Name	Description	Range	Type	Default	Required
voltage	baseband or complex envelope voltage signal	(-∞, ∞)	integer, real, complex		yes
percent	percentage of time the returned power value is exceeded	[0, 100]	real	0.0	no
unit	unit to be used	"dB"|"ratio"	string	"dB"	no

Examples

```
peak_avg_ratio = peak_to_avg_pwr(Vout[1], , “ratio”)
```

where Vout is a named node in a Circuit Envelope simulation, will return the peak to average power ratio (as a ratio) at the fundamental frequency.

```
peak_avg_ratio = peak_to_avg_pwr(T1, 5, “ratio”)
```

where T1 is the name of a TimedSink component (in a DSP schematic), will return the ratio of the power level that is exceeded 5% of the time to the average power level for the voltage signal recorded in the TimedSink. If the signal recorded by the TimedSink is complex envelope Gaussian noise with the same standard deviation for both the I and Q envelopes, then peak_avg_ratio will be very close to 2.99 (-ln(0.05)).

```
peak_avg_ratio = peak_to_avg_pwr(T1)
```

where T1 is the name of a TimedSink component (in a DSP schematic), will return the peak to average power ratio (in dB) for the voltage signal recorded in the TimedSink. If the signal recorded by the TimedSink is an ideal QPSK signal (filtered using a raised cosine filter of ExcessBW 0.5), then peak_avg_ratio will be very close to 3.95 dB. If the signal recorded by the TimedSink is an ideal π/4-DQPSK signal (filtered using a raised cosine filter of ExcessBW 0.5), then peak_avg_ratio will be very close to 3.3 dB.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

peak_pwr(), power_ccdf(), pwr_vs_t(), total_pwr()

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.

This expression can be used with input data (voltage) of any dimensions when the percent argument is 0 and up to three dimensions when the percent argument is greater than 0. The expression can accommodate both baseband as well as complex envelope data.

The peak to average ratio is computed by calling the peak_pwr() and total_pwr() expressions and then taking the ratio of the two values returned by these expressions. The percent argument is used for the calculation of the peak power value. For the use and meaning of the percent argument see the peak_pwr() Notes/Equations.

Since a ratio of power values is calculated a reference resistance is not needed for this measurement.

power_ccdf()

Returns the power CCDF (Complementary Cumulative Distribution Function) curve for the input voltage/current data

Syntax

pCCDF = power_ccdf(data, numBins)

Arguments

Name	Description	Range	Type	Default	Required
data	baseband or complex envelope voltage/current signal	(-∞, ∞)	integer, real, complex		yes
numBins	number of points in the returned power CCDF curve	[1, ∞)	integer	log2(num DataPoints)	no

Examples

```
pCCDF = power_ccdf(Vout[1])
```

where Vout is a named node in a Circuit Envelope simulation, will return the power CCDF curve for the voltage at the fundamental frequency. The returned power CCDF curve will have the default number of points.

```
pCCDF = power_ccdf(T1, 10)
```

where T1 is the name of a TimedSink component (in a DSP schematic), will return the power CCDF curve for the voltage signal recorded in the TimedSink. The returned curve will have 10 points.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

peak_pwr(), peak_to_avg_pwr(), power_ccdf_ref(), pwr_vs_t(), total_pwr()

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.

This expression can be used with input data of up to two dimensions. It can accommodate both baseband as well as complex envelope data.

The power CCDF (typically called just CCDF) measurement is a very common measurement performed on 2G and 3G wireless signals. The CCDF curve shows the probability that the instantaneous signal power will be higher than the average signal power by a certain amount of dB. The independent axis of the CCDF curve shows power levels in dB with respect to the signal average power level (0 dB corresponds to the signal average power level). The dependent axis of the CCDF curve shows the probability that the instantaneous signal power will exceed the corresponding power level on the independent axis. Figure 4-4 shows the CCDF curve for a WLAN 802.11a 54 Mbps signal.

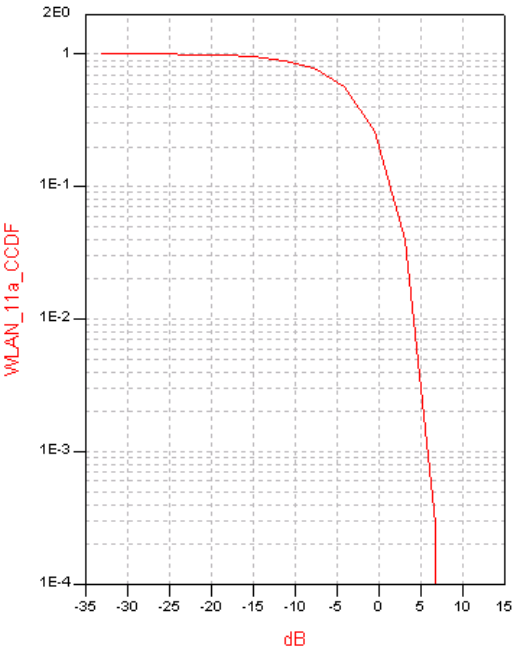


Figure 4-4. CCDF Curve for WLAN 802.11a 54 Mbps Signal

In Figure 4-4, you can see that the instantaneous signal power exceeds the average signal power (0 dB) for 20% of the time. You can also see that the instantaneous signal power exceeds the average signal power by 5 dB for only 0.3% of the time.

power_ccdf_ref()

Returns the power CCDF (Complementary Cumulative Distribution Function) curve for white gaussian noise

Syntax

ccdfRef = power_ccdf_ref(indepPowerRatioValues)

Arguments

Name	Description	Range	Type	Required
indepPowerRatioValues	power levels (in dB with respect to the average power level of a white gaussian noise signal) at which the power CCDF for white gaussian noise will be calculated	(-∞, ∞)	integer, real	yes

Examples

```
pCCDF = power_ccdf(T1)
ccdfRef = power_ccdf_ref( indep( pCCDF) )
```

where T1 is the name of a TimedSink component (in a DSP schematic), will return the power CCDF curve for white gaussian noise evaluated at the same power levels as the pCCDF curve.

```
ccdfRef = power_ccdf_ref( [-10::2::8] )
```

will return the power CCDF curve for white gaussian noise evaluated at the power levels -10, -8, -6, -4, -2, 0, 2, 4, 6, 8 (in dB with respect to the average power level of a white gaussian noise signal).

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

power_ccdf()

Notes/Equations

This expression can be used with input data of up to two dimensions.

The input data must be sorted in increasing or decreasing order. In addition, it is recommended that the input vector argument has uniformly spaced values. The

vector “indep(X)”, where X is the value returned by the power_ccdf() expression, is guaranteed to have uniformly spaced values sorted in increasing order. This would be the most typical value for the input argument of power_ccdf_ref().

A typical power CCDF measurement (see Notes/Equations for power_ccdf()) provides a reference CCDF curve along with the measured CCDF curve. This reference curve is typically the power CCDF curve for a white gaussian noise signal. This expression can generate this reference curve. Figure 4-5 shows the CCDF curve for a WLAN 802.11a signal along with the reference curve.

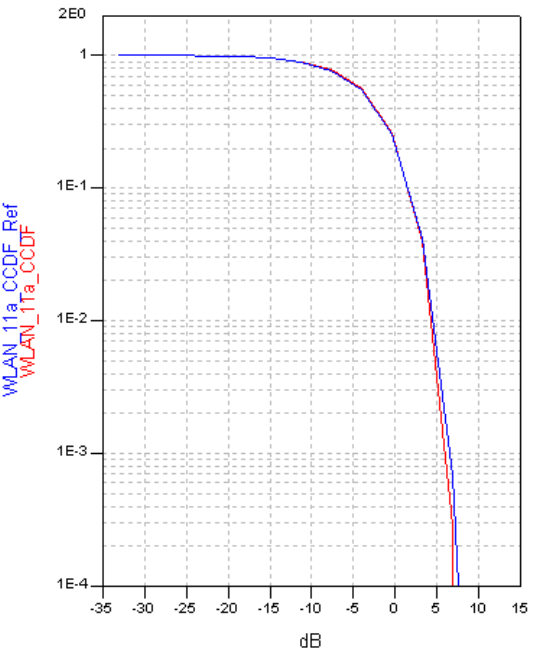


Figure 4-5. CCDF Curve for WLAN 802.11a Signal with Reference Curve

pwr_vs_t()

Returns the power vs. time waveform for the input voltage data

Syntax

p_vs_t = pwr_vs_t(voltage, refR, unit)

Arguments

Name	Description	Range	Type	Default	Required
voltage	baseband or complex envelope voltage signal	(-∞, ∞)	integer, real, complex		yes
refR	reference resistance in Ohms	(0, ∞)	real	50.0	no
unit	power unit to be used	"W"|"dBm"|"dBW"	string	"W"	no

Examples

```
p_vs_t = pwr_vs_t(Vout[1], 100, “dBm”)
```

where Vout is a named node in a Circuit Envelope simulation, will return the power (in dBm) vs. time waveform for the voltage at the fundamental frequency using 100 Ohms as reference resistance.

```
p_vs_t = pwr_vs_t(T1)
```

where T1 is the name of a TimedSink component (in a DSP schematic), will return the power (in W) vs. time waveform for the voltage signal recorded in the TimedSink using 50 Ohms as reference resistance.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

peak_pwr(), peak_to_avg_pwr(), power_ccdf(), total_pwr()

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.

This expression can be used with input data (voltage) of any dimensions. It can accommodate both baseband as well as complex envelope data.

relative_noise_bw()

Computes the relative noise bandwidth of the smoothing windows used by the fs() function

Syntax

y = relative_noise_bw(winType, winConst)

Arguments

Name	Description	Range	Type	Default	Required
winType	window type	†	string		no
winConst	window constant that affects the shape of the applied window.	[0, ∞)	real	0.75	no
† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"					

Examples

```
winType = "Kaiser"
winConst = 8
relNoiseBW = relative_noise_bw("Kaiser", 8) = 1.666
Vfund=vOut[1]
VoltageSpectralDensity = fs(Vfund, , , , , winType, winConst)
PowerSpectralDensity = 0.5 * mag(VoltageSpectralDensity**2)/50/relNoiseBW
```

where vOut is the named connection at a 50-ohm load, and it is an output from a Circuit Envelope simulation.

Note vOut is a named connection on the schematic. Assuming that a Circuit Envelope simulation was run, vOut is output to the dataset as a two-dimensional matrix. The first dimension is time, and there is a value for each time point in the simulation. The second dimension is frequency, and there is a value for each fundamental frequency, each harmonic, and each mixing term in the analysis, as well as the baseband term.

vOut[1] is the equivalent of vOut[::, 1], and specifies all time points at the lowest non-baseband frequency (the fundamental analysis frequency, unless a multitone analysis has been run and there are mixing products). For former MDS users, the

notation "vOut[*, 2]" in MDS corresponds to the notation of "vOut[1]".

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

acpr_vi(), acpr_vr(), channel_power_vi(), channel_power_vr(), fs()

Notes/Equations

Used in The following functions: acpr_vi, acpr_vr, channel_power_vi, channel_power_vr.

The relative noise bandwidth function is used to account for the fact that as windows are applied, the effective noise bandwidth increases with respect to the normal resolution bandwidth. The resolution bandwidth is determined by the time span and not by the displayed frequency resolution.

sample_delay_pi4dqpsk()

This function calculates the optimal sampling point within a symbol for a given pi4dqpsk waveform. "Optimal" is defined as the sampling point that provides the lowest bit error rate

Syntax

y = sample_delay_pi4dqpsk(vlQ, symbolRate, path_delay, timeResolution)

Arguments

Name	Description	Range	Type	Required
vlQ	complex envelope (I + j * Q) of a pi/4 DQPSK signal	(-∞, ∞)	complex	yes
symbolRate	symbol rate of the pi/4 DQPSK signal	(0, ∞)	real	yes
path_delay	time delay on the waveform before the sampling starts †	(0, ∞)	real	yes
timeResolution	time step (typically one-tenth of a symbol time or less) used to search for the best sampling point in a given symbol period	(0, ∞)	real	yes
† If the delay is 0, this parameter may be omitted. If it is non-zero, enter the delay value. This can be calculated using the function delay_path()				

Examples

```
a = sample_delay_pi4dqpsk(vout[1], 25e3, 1.5e-6, 0.15e-6)
```

Defined in

Built in

See Also

ber_pi4dqpsk(), ber_qpsk(), const_evm()

Notes/Equations

This function can be used only with 1-dimensional data.

sample_delay_qpsk()

This function calculates the optimal sampling point within a symbol for a given QPSK waveform."Optimal" is defined as the sampling point that provides the lowest bit error rate

Syntax

y = sample_delay_qpsk(vlQ, symbolRate, path_delay, timeResolution)

Arguments

Name	Description	Range	Type	Required
vlQ	complex envelope (I + j * Q) of a QPSK signal	(-∞, ∞)	complex	yes
symbolRate	symbol rate of the QPSK signal	(0, ∞)	real	yes
path_delay	time delay on the waveform before the sampling starts †	(0, ∞)	real	yes
timeResolution	time step (typically one-tenth of a symbol time or less) used to search for the best sampling point in a given symbol period	(0, ∞)	real	yes
† If the delay is 0, this parameter may be omitted. If it is non-zero, enter the delay value. This can be calculated using the function delay_path()				

Examples

```
a = sample_delay_qpsk(vout[1], 25e3, 1.5e-6, 0.15e-6)
```

Defined in

Built in

See Also

ber_pi4dqpsk(), ber_qpsk(), const_evm()

Notes/Equations

This function can be used only with 1-dimensional data.

spectrum_analyzer()

Performs a spectrum analysis for the input voltage/current data

Syntax

spectrum = spectrum_analyzer(data, fCarrier, start, stop, window, resBW)

Arguments

Name	Description	Range	Type	Default	Required
data	baseband or complex envelope voltage/current signal	(-∞, ∞)	real, complex		yes
fCarrier	frequency around which the spectrum will be centered	[0, ∞)	real	0	no
start	start time for the spectrum analysis	[0, ∞)	integer, real	first point of input data	no
stop	stop time for the spectrum analysis	[0, ∞)	integer, real	last point of input data	no
window	type of window to be used	[0, 7] †	integer, string	0	no
resBW	resolution bandwidth	[0, ∞)	integer, real	0	no
† See Notes for the window type					

Examples

```
spectrum = spectrum_analyzer(Vout[1])
```

where Vout is a named node in a Circuit Envelope simulation, will return the voltage spectrum at the fundamental frequency. The spectrum will be centered around 0 Hz. All the input data will be processed in one block resulting in the highest resolution bandwidth possible. No windowing will be done.

```
spectrum = spectrum_analyzer(Vout[1], 3.5e9, , , “Kaiser 7.865”, 30e3)
```

where Vout is a named node in a Circuit Envelope simulation, will return the voltage spectrum at the fundamental frequency. The spectrum will be centered around 3.5 GHz. The input signal will be broken down in smaller segments in order to achieve 30 kHz of resolution bandwidth. All segments will be windowed with a Kaiser 7.865 window. The spectra of all segments will be averaged.

```
spectrum = spectrum_analyzer(T1, , 1.0e-3, 2.0e-3, “Hanning 0.5”)
```
where T1 is the name of a TimedSink component (in a DSP schematic), will return the voltage spectrum for the segment between 1 msec and 2 msec of the signal recorded in the TimedSink. Of course, the TimedSink component must have recorded a signal that starts before 1 msec and ends after 2 msec. The spectrum will be centered around 0 Hz. A Hanning 0.5 window will be used.

Defined in

Built in

See Also

fs()

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.

This expression can be used with input data of up to two dimensions. It can handle both baseband as well as complex envelope input data. Although non-uniformly sampled data is supported, it is recommended to use this expression with uniformly sampled data.

The spectrum_analyzer() expression returns the voltage or current spectrum of the input (voltage or current) signal. The returned values are the complex amplitude voltages or currents at the frequencies of the spectral tones. The returned values are not the powers at the frequencies of the spectral tones. However, one can calculate and display the power spectrum by using the dbm() expression (for voltage input data) or writing a simple equation (for current input data) in the data display window.

Note that, for baseband signals and for the frequency of 0 Hz, the dBm function returns a power value that is 3 dB less than the actual power. This is because the primary use of the dBm function is with RF signals, where the 0 Hz frequency corresponds to the carrier frequency and not really 0 Hz signal frequency. If the baseband signal has no significant power at DC, this 3 dB error is insignificant and can be ignored—otherwise, it must be considered.

The basis of the algorithm used by the spectrum_analyzer() expression is the fs() expression (the chirp-Z transform option of fs() is used). The input data can be processed as one block and the spectrum calculated over the entire input signal. Alternatively, the input signal can be broken down in smaller blocks and the spectra of all blocks averaged.

The *fCarrier* argument sets the frequency around which the spectrum will be centered. This is only true for a complex envelope signal. For baseband signals, this argument is ignored. The spectrum span is:

- for complex envelope signals [fCarrier-0.5/TStep, fCarrier+0.5/TStep]
- for baseband signals [0, 0.5/TStep]

  where *TStep* is the simulation time step.

If it is not desirable to process all the input data, the *start* and *stop* arguments can be used to define a segment of the input data to be processed. This can be useful when trying to exclude parts of the signal where transients occur.

The *window* argument sets the window that will be used to window the input data. Windowing is often necessary in transform-based (chirp-Z, FFT) spectrum estimation. Without windowing, the estimated spectrum may suffer from spectral leakage that can cause misleading measurements or masking of weak signal spectral detail by spurious artifacts. If the input data is broken down in multiple blocks then the window is applied to each block.

The window argument accepts the following strings: "none", "Hamming 0.54", "Hanning 0.50", "Gaussian 0.75", "Kaiser 7.865", "8510 6.0", "Blackman", "Blackman-Harris". The equations defining these windows are given below:

- none:

$$w(kT_s) = \begin{cases} 1.0 & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$

  where N is the window size

- Hamming 0.54:

$$w(kT_s) = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi k}{N}\right) & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$

  where N is the window size

- Hanning 0.5:

$$w(kT_s) = \begin{cases} 0.5 - 0.5\cos\left(\frac{2\pi k}{N}\right) & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$

where N is the window size

- Gaussian 0.75:

$$w(kT_s) = \begin{cases} exp\left(-\frac{1}{2}\left(0.75\frac{(2k-N)}{N}\right)^2\right) & 0 \le \left|k - \frac{N}{2}\right| \le \frac{N}{2} \\ 0.0 & otherwise \end{cases}$$

where N is the window size

- Kaiser 7.865:

$$w(kT_s) = \begin{cases} \frac{I_0\left(7.865\left[1-\left(\frac{k-\alpha}{\alpha}\right)^2\right]^{1/2}\right)}{I_0(7.865)} & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$

where N is the window size, $\alpha = N / 2$, and $I_0(.)$ is the *0*th order modified Bessel function of the first kind

- 8510 6.0 (Kaiser 6.0):

$$w(kT_s) = \begin{cases} \frac{I_0\left(6.0\left[1-\left(\frac{k-\alpha}{\alpha}\right)^2\right]^{1/2}\right)}{I_0(6.0)} & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$

where N is the window size, $\alpha = N / 2$, and $I_0(.)$ is the *0*th order modified Bessel function of the first kind

- Blackman:

$$w(kT_s) = \begin{cases} 0.42 - 0.5\cos\left(\frac{2\pi k}{N}\right) + 0.08\cos\left(\frac{4\pi k}{N}\right) & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$

where N is the window size

- Blackman-Harris:

$$w(kT_s) = \begin{cases} 0.35875 - 0.48829\cos\left(\frac{2\pi k}{N}\right) + 0.14128\cos\left(\frac{4\pi k}{N}\right) - 0.01168\cos\left(\frac{6\pi k}{N}\right) & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$

where N is the window size

The *resBW* parameter can be used to set the spectrum measurement resolution bandwidth. If set to 0, it is ignored and the signal segment defined by start and stop is processed as one segment. In this mode of operation, the returned spectrum will have the highest possible resolution bandwidth (resolution bandwidth is inversely proportional to the input signal length). If resBW is set to a value greater than 0, then the input signal segment defined by start and stop is broken down in smaller subsegments of the appropriate length. The length of each segment is decided based on the value of resBW and the selected window. The spectrum of each subsegment is calculated and averaged with the spectra of the other subsegments. In this mode of operation, the resolution bandwidth achieved is resBW. Figure 4-6 shows an example where the input signal is broken down in multiple segments. As can be seen in this figure, if an exact integer multiple of subsegments cannot fit in the segment defined by start and stop, then part of the signal may not be used.

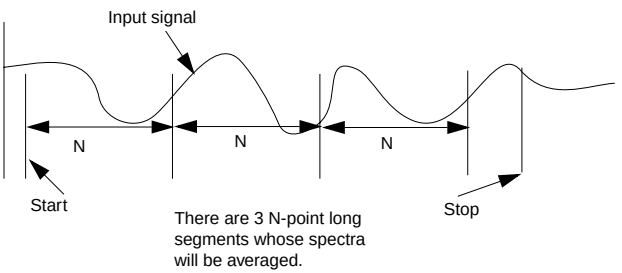


Figure 4-6. Averaging multiple input signal segments.

In a standard swept spectrum analyzer, the resolution bandwidth is determined by the last in a series of analog IF filters. In contrast, the spectrum_analyzer() expression calculates the spectrum using DSP algorithms and so the resolution bandwidth is determined by the length of the input data segment the algorithm processes and the selected window.

ResBW = ENBW = NENBW / T

where ENBW is the window equivalent noise bandwidth

T is the length of the input data segment in seconds and

NENBW is the window normalized equivalent noise bandwidth (Table 4-7 shows the NENBW values for the available windows).

Table 4-7. NENBW for available windows.

Window	NENBW
none	1.0
Hamming 0.54	1.363
Hanning 0.5	1.5
Gaussian 0.75	1.883
Kaiser 7.865	1.653
8510 6.0	1.467
Blackman	1.727
Blackman-Harris	2.021

The equivalent noise bandwidth (ENBW) of a window is defined as the width of a rectangular filter that passes the same amount of white noise as the window. The normalized equivalent noise bandwidth (NENBW) is computed by multiplying ENBW with the time record length. For example, a Hanning window with a 0.5 second input data segment will result in an ENBW (as well as ResBW) of 3 Hz (1.5 / 0.5).

total_pwr()

Returns the total (average) power of the input voltage data

Syntax

totalP = total_pwr(voltage, refR, unit)

Arguments

Name	Description	Range	Type	Default	Required
voltage	baseband or complex envelope voltage signal	(-∞, ∞)	integer, real, complex		yes
refR	reference resistance in Ohms	(0, ∞)	real	50.0	no
unit	power unit to be used	"W"|"dBm"|"dBW"	string	"W"	no

Examples

```
totalP = total_pwr(Vout[1], 100, "dBm")
```

where Vout is a named node in a Circuit Envelope simulation, will return the total power (in dBm) at the fundamental frequency using 100 Ohms as reference resistance.

```
totalP = total_pwr(T1)
```

where T1 is the name of a TimedSink component (in a DSP schematic), will return the total power (in W) for the voltage signal recorded in the TimedSink using 50 Ohms as reference resistance. If the signal recorded by the TimedSink is baseband Gaussian noise with a standard deviation of 30 mV, then totalP will be very close to 1.8e-5 W (0.03^2 / 50).

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

peak_pwr(), peak_to_avg_pwr(), power_ccdf(), pwr_vs_t()

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.

This expression can be used with input data (voltage) of any dimensions. It can accommodate both baseband as well as complex envelope data.

trajectory()

Generates the trajectory diagram from I and Q data, which are usually produced by a Circuit Envelope simulation

Syntax

Traj = trajectory(i_data, q_data)

Arguments

Name	Description	Range	Type	Required
i_data	in-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †	(-∞, ∞)	complex	yes
q_data	quadrature-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †	(-∞, ∞)	complex	yes
† This could be a baseband signal instead, but in either case it must be real valued versus time				

Examples

```
Rotation = –0.21
Vfund=vOut[1] *exp(j * Rotation)
Vimag = imag(Vfund)
Vreal = real(Vfund)
Traj = trajectory(Vreal, Vimag)
```

where Rotation is a user-selectable parameter that rotates the trajectory diagram by that many radians and vOut is the named connection at a node.

Note vOut is a named connection on the schematic. Assuming that a Circuit Envelope simulation was run, vOut is output to the dataset as a two-dimensional matrix. The first dimension is time, and there is a value for each time point in the simulation. The second dimension is frequency, and there is a value for each fundamental frequency, each harmonic, and each mixing term in the analysis, as well

as the baseband term.

vOut[1] is the equivalent of vOut[::, 1], and specifies all time points at the lowest non-baseband frequency (the fundamental analysis frequency, unless a multitone analysis has been run and there are mixing products). For former MDS users, the notation "vOut[*, 2]" in MDS corresponds to the notation of "vOut[1]".

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

constellation(), const_evm()

Notes/Equations

Used in Trajectory diagram generation.

The I and Q data do not need to be baseband waveforms. For example, they could be the in-phase (real or I) and quadrature-phase (imaginary or Q) part of a modulated carrier. The user must supply the I and Q waveforms versus time.

Chapter 5: Data Access Functions

This chapter describes data access and data manipulation functions in detail. The functions are listed in alphabetical order.

Note You can use these functions to find information about a piece of data (e.g., independent values, size, type, attributes, etc.). You can also use some functions to generate data for plotting circles and contours.

---

build_subrange()

Builds the subrange data according to the innermost independent range. Use with all swept data

Syntax

y = build_subrange(data, innermostIndepLow, innermostIndepHigh)

Arguments

Name	Description	Range	Type	Default	Required
data	analysis results or user input.	(-∞, ∞)	integer, real, complex		yes
innermostIndepLow	lowest value of innermost independent	(-∞, ∞)	integer, real, complex	minimum value of the inner most independent variable	no
innermostIndepHigh	highesr value of innermost independent	(-∞, ∞)	integer, real, complex	maximum value of the inner most independent variable	no

Examples

Given S-parameter data swept as a function of frequency with a range of 100 MHz to 500 MHz, find the values of S12 in the range of 200 MHz to 400 MHz.

```
subrange_S12 = build_subrange(S12, 200MHz, 400MHz)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

chop()

Replace numbers in x with magnitude less than dx with 0

Syntax

y = chop(x, dx)

Arguments

Name	Description	Range	Type	Required
x	numbers to replace	(-∞, ∞)	integer, real, complex	yes
dx	value to compare to	(-∞, ∞)	integer, real, complex	yes

Examples

```
chop(1)
```
returns 1

```
chop(1e-12)
```
returns 0

```
chop(1+1e-12i)
```
returns 1+0i

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

Notes/Equations

The chop() function acts independently on the real and complex components of x, comparing each to mag(dx). For example,

```
y = x if mag(x) >= mag(dx)
y = 0 if mag(x)
```

chr()

Returns the character representation of an integer

Syntax

y = chr(x)

Arguments

Name	Description	Range	Type	Required
x	valid number representing a ASCII character	[0, 127]	integer	yes

Examples

```
a = chr(64)
```
returns @

```
a = chr(60)
```
returns <

```
a = chr(117)
```
returns u

Defined in

Built in

circle()

Used to draw a circle on a Data Display page. Accepts the arguments center, radius, and number of points. Can only be used on polar plots and Smith charts

Syntax

a = circle(center, radius, numPts)

Arguments

Name	Description	Range	Type	Required
center	center coordinate	(-∞, ∞)	integer, real, complex	yes
radius	radius	[0, ∞)	integer, real	yes
numPts	number of points in the circle	[0, ∞)	integer	yes

Examples

```
x = circle(1,1,500)
y = circle(1+j*1,1,500)
```

Defined in

Built in

collapse()

Collapses the inner independent variable and returns one dimensional data

Syntax

y = collapse(x)

Arguments

Name	Description	Range	Type	Required
x	multi-dimensional data to be collapsed (dimension is larger than one and less than four)	(-∞, ∞)	integer, real, complex	yes

Examples

Given monte carlo analysis results for the S11 of a transmission line:
It is two-dimensional data: the outer sweep is mcTrial; the inner sweep is the frequency from 100 MHz to 300 MHz and is given in the following format:

mcTrial	freq	S11
1	100MHz	0.2
	200MHz	0.4
	300MHz	0.6
2	100MHz	0.3
	200MHz	0.5
	300MHz	0.7

Returns a one dimensional data with mcTrail, containing all of the previous data.

```
collapsed_S11 = collapse(S11)
```

mcTrial	S11
1	0.2
1	0.4
1	0.6
2	0.3

2	0.5
2	0.7

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

expand()

contour()

Generates contour levels on surface data

Syntax

y = contour(data, contour_levels, interpolation_type)

Arguments

Name	Description	Range	Type	Default	Required
data	data to be contoured, which must be at least two-dimensional real, integer or implicit	(-∞, ∞)	integer, real		yes
contour_levels	one-dimensional quantity specifying the levels of the contours †	(-∞, ∞)	integer, real	six levels equally spaced between the maximum and the minimum of the data	no
interpolation_type	specifies the type of interpolation to perform	[0, 1,2] † †	integer	0	no

† Normally specified by the sweep generator "[ ]," but can also be specified as a vector
† † Interpolation types are: 0 - No Interpolation,1 - Cubic Spline,2 - B-Spline

Examples

```
a = contour(dB(S11), [1::3::10])
```

or

```
a = contour(dB(S11), {1, 4, 7, 10})
```

produces a set of four equally spaced contours on a surface generated as a function of, say, frequency and strip width.

```
a = contour(dB(S11), {1, 4, 7, 10}, 1)
```

produces the same set of contours as the above example, but with cubic spline interpolation.

Defined in

Built in

See Also

contour_polar()

Notes/Equations

This function introduces three extra inner independents into the data. The first two are “level”, the contour level, and “number”, the contour number. For each contour level there may be n contours. The contour is an integer running from 1 to n. The contour is represented as an (x, y) pair with x as the inner independent.

contour_polar()

Generates contour levels on polar or Smith chart surface data

Syntax

y = contour_polar(data, contour_levels, InterpolationType, DataFormat)

Arguments

Name	Description	Range	Type	Default	Required
data	polar or Smith chart data to be contoured, (and therefore is surface data)	(-∞, ∞)	integer, real, complex		yes
contour_levels	one-dimensional quantity specifying the levels of the contours †	(-∞, ∞)	integer, real	six levels equally spaced between the maximum and the minimum of the data	no
InterpolationType	specifies the type of interpolation to perform	[0, 1,2] † †	integer	0	no
DataFormat	format of swept data	"RI", "MA" ‡	string	"RI"	no

† Normally specified by the sweep generator "[ ]," but can also be specified as a vector
† † Interpolation types are: 0 - No Interpolation,1 - Cubic Spline,2 - B-Spline
‡ DataFormat are: "RI" = real-imaginary, "MA" = magnitude-phase

Examples

```
a = contour_polar(data_polar, [1::4])
```

or

```
a = contour_polar(data_polar, {1, 2, 3, 4})
```

produces a set of four equally spaced contours on a polar or Smith chart surface.

```
a = contour_polar(data_polar, {1, 2, 3, 4}, 2)
```

produces the same set of contours as the above example, but with B-spline interpolation.

Defined in

$HPEESOF_DIR/expressions/ael/display_fun.ael

See Also

contour()

Notes/Equations

This function introduces three extra inner independents into the data. The first two are "level", the contour level, and "number", the contour number. For each contour level there may be n contours. The contour is an integer running from 1 to n. The contour is represented as an (x,y) pair with x as the inner independent.

copy()

Makes a copy of a multi-dimensional data variable

Syntax

y = copy(DataVar)

Arguments

Name	Description	Type	Required
DataVar	data variable or array that is to be copied	boolean, integer, real, complex, string	yes

Examples

```
result = copy(S21)
```
returns the copy of the data stored in the data variable S21

The array or data variable created above can be used as follows:

```
indepV = indep(result,"Index");
result[0] = complex(1, 2); Sets the first value to a complex number
indepV[0] = 1GHz;
```

Defined in

Built in

See Also

create(), delete()

Notes/Equations

Makes a copy of a multi-dimensional data variable, so that the contents of the copy can be manipulated. Data Variables in ADS/RFDE are data strcutures that are used to hold multi-dimensional data. Internally they are not implemented as arrays, and therefore do not have the performance of an array. Accessing and setting data in these arrays are performance intensive and should be noted.

create()

Creates a multi-dimensional data variable

Syntax

y = create(Dimensionality, DependDataType, IndepName, IndepType, NumRows, NumColumns)

Arguments

Name	Description	Range	Type	Default	Required
Dimensionality	dimensionality of the data variable or array	[1, ∞)	integer		yes
DependDataType	Dependent data type	†	string	"Real"	no
IndepName	Name(s) of independent		string	"__i"	no
IndepType	Independent data type	†	string	"Real"	no
NumRows	Number of rows	[0, ∞)	integer	0	no
NumColumns	Number of columns	[0, ∞)	integer	0	no
† "Boolean", "Integer", "Real", "Complex", "String" or "Byte16"					

Examples

```
result = create(1, "Complex", {"Index"}, {"Real"},1, 1)
```
returns a 1 dimensional data variable with dependent type Complex, independent name "Index" and type real with 1 row and column

The array or data variable created above can be used as follows:

```
indepV = indep(result,"Index");
result[0] = 1.1;
indepV[0] = 1.0;
```

A 2-dimensional example is given in the Measurement Expression AEL function below. Test the function by doing the following:

1. Copy the code below into the file user_defined_fun.ael in the directory $HOME/hpeesof/expressions/ael.
2. Launch Advanced Design System. In a Data Display window, create an equation:

```
cre2D = datavar_test_create2()
```

3. Display the equation `cre2D` to view the contents.

```
defun datavar_test_create2()
{
  decl result = create(2, "Complex", {"Index1", "Index2"}, {"String",
"Real"},1,1);
  decl idenp1V = indep(result, "Index1");
  decl idenp2V = indep(result, "Index2");
  decl iD1, iD2;
  for (iD1=0; iD1 < 2; iD1++) {
    for (iD2=0; iD2 < 3; iD2++) {
      result[iD1,iD2] = complex(iD1, iD2);
      idenp2V[iD2] = iD2;
    } //for
    idenp1V[iD1] = strcat("Val", iD1);
  } //for
  return result;
}
```

Defined in

Built in

See Also

copy(), delete()

Notes/Equations

This function is used to create multi-dimensional data variable or arrays. Data Variables in ADS/RFDE are data strcutures that are used to hold multi-dimensional data. Internally they are not implemented as arrays, and therefore do not have the performance of an array. Accessing and setting data in these arrays are performance intensive and should be noted.

delete()

Deletes the multi-dimensional data variable

Syntax

y = delete(DataVar)

Arguments

Name	Description	Type	Required
DataVar	data variable or array that is to be deleted	boolean, integer, real, complex, string	yes

Examples

```
result = delete(S21)
```

returns true or false depending on whether the data variable was deleted or not

Defined in

Built in

See Also

copy(), create()

expand()

Expands the dependent data of a variable into single points by introducing an additional inner independent variable

Syntax

y = expand(x)

Arguments

Name	Description	Range	Type	Required
x	data to be expanded (dimension is larger than one and less than four)	(-∞, ∞)	integer, real, complex	yes

Examples

Given a dependent data A which has independent variables B: If A is a 1 dimensional data containing 4 points (10, 20, 30, and 40) and similarly B is made up of 4 points (1, 2, 3, and 4),

```
Eqn A = [10,20,30,40]
Eqn B = [1,2,3,4]
Eqn C = vs(A,B,"X")
```

Using expand(C) increases the dimensionality of the data by 1 where each inner dependent variable ("X") consists of 1 point.

```
Eqn Y = expand(C)
```

X	C	A	B	Y
1 2 3 4	10 20 30 40	10 20 30 40	1 2 3 4	X=1 10 X=2 20 X=3 30 X=4 40

Defined in

Built in

See Also

collapse()

find()

Finds the indices of the conditions that are true. Use with all simulation data

Syntax

indices = find(condition)

Arguments

Name	Description	Type	Required
condition	condition	string	yes

Examples

Given an S-parameter data swept as a function of frequency, find the value of S_{11} at 1GHz.

```
index_1 = find(freq == 1GHz)
data = S11[index_1]
```

Given an S-parameter data swept as a function of frequency, find the values of the frequencies where the magnitude of S_{11} is greater than a given value.

```
lookupValue = 0.58
indices = find(mag(S11) > lookupValue))
firstPoint = indices[0]
lastPoint = indices[sweep_size(indices)-1]
freqDifference = freq[lastPoint]- freq[firstPoint]
```

The following examples assume a Harmonic Balance data vtime, and a marker m1.

Find the dependent value at the marker:

```
vVal = find(indep(vtime) >= indep(m1) && indep(vtime) <=indep(m1))
```

Find all the dependent values less than that of the m1 or the value at m1:

```
vVal = find(indep(vtime) < indep(m1) || indep(vtime) == indep(m1))
```

Find all the dependent values that are not equal to m1:

```
vVal = find(indep(vtime) != indep(m1))
```

Defined in

$HPEESOF_DIR/expressions/ael/utility_fun.ael

See Also

find_index(), mix()

Notes/Equations

The find function will return all the indices of the conditions that are true. If none of the conditions are true, then a -1 is returned. The find function performs an exhaustive search on the given data. The supplied data can be an independent or dependent data. In addition, the dimension of the data that is returned will be identical to the dimension of the input data.

The find function can accept conditionals such as ==, !=, >, <, >= and <=, and logical operators such as && and ||.

find_index()

Finds the closest index for a given search value. Use with all simulation data

Syntax

index = find_index(data_sweep, search_value)

Arguments

Name	Description	Range	Type	Required
data_sweep	data to search	(-∞, ∞)	integer, real, complex, string	yes
search_value	value to search	(-∞, ∞)	integer, real, complex, string	yes

Examples

Given S-parameter data swept as a function of frequency, find the value of S_{11} at 1 GHz:

```
index = find_index(freq, 1GHz)
a = S11[index]
```

Defined in

Built in

See Also

find(), mix()

Notes/Equations

To facilitate searching, the find_index function finds the index value in a sweep that is closest to the search value. Data of type int or real must be monotonic. find_index also performs an exhaustive search of complex and string data types.

generate()

This function generates a sequence of real numbers. The modern way to do this is to use the sweep generator "[ ]."

Syntax

y = generate(start, stop, npts)

Arguments

Name	Description	Range	Type	Required
start	start value of sequence	(-∞, ∞)	integer, real	yes
stop	stop value of sequence	(-∞, ∞)	integer, real	yes
npts	Numper of points in the sequence	[2, ∞)	integer	yes

Examples

```
a = generate(9, 4, 6)
```

returns the sequence 9., 8., 7., 6., 5., 4.

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

get_attr()

Gets a data attribute. This function only works with frequency swept variables

Syntax

y = get_attr(data, "attr_name", eval)

Arguments

Name	Description	Range	Type	Default	Required
data	frequency swept variable	(-∞, ∞)	integer, real, complex		yes
attr_name	name of the attribute		string		yes
eval	specifies whether to evaluate the attribute	false|true	boolean	true	no

Examples

```
y = get_attr(data, "fc", true)
```
returns 10GHz

```
y = get_attr(data, "dataType")
```
returns "TimedData"

```
y = get_attr(data, "TraceType", false)
```
returns "Spectral"

Defined in

Built in

See Also

set_attr()

get_indep_values()

Returns the independent values associated with the given dependent value as an array

Syntax

indepVals = get_indep_values(Data, LookupValue)

Arguments

Name	Description	Range	Type	Default	Required
Data	1 to 5 dimensional array.	(-∞, ∞)	integer, real, complex		yes
LookupValue	Dependent value for which the corresponding independent values have to be found.	(-∞, ∞)	real, complex		yes
Tolerance	tolerance to be used while comparing numbers	[0, ∞)	real	0	no
All	Finds all matches of the LookupValue. Default behavior is to return after the first match.	[0, 1]	integer	0	no

Examples

We assume that the data is 2-dimensional, that is, two independent variables created from a Harmonic Balance Analysis with Pout being the output data.

```
indepVals = get_indep_values(Pout, max(max(Pout))
```
returns the values of the independent as an array

```
indepVals = get_indep_values(Pout, [m1,m2])
```
returns the indepenent values of the markers m1and m2

Defined in

$HPEESOF_DIR/expressions/ael/utility_fun.ael

See Also

indep()

Notes/Equations

This function can be used only on 1 to 5 dimensional data. The independent values have to be real. The dependent value to be looked up can be a single value or multiple values.

indep()

Returns the independent attached to the data

Syntax

Y = indep(x, NumberOrName)

Arguments

Name	Description	Range	Type	Default	Required
x	data to access the independent values	(-∞, ∞)	integer, real, complex		yes
NumberOrName	number or name of independent	[0, 6]	integer, string	0	no

Examples

Given S-parameters versus frequency and power: Frequency is the innermost independent, so its index is 1. Power has index 2.

```
freq = indep(S, 1)
freq = indep(S, "freq")
power = indep(S, 2)
power = indep(S, "power")
```

Defined in

Built in

See Also

find_index(), get_indep_values()

Notes/Equations

The indep() function returns the independent (normally the swept variable) attached to simulation data. When there is more than one independent, then the independent of interest may be specified by number or by name. If no independent specifications are passed, then indep() returns the innermost independent.

max_index()

Returns the index of the maximum

Syntax

max_index(x)

Arguments

Name	Description	Range	Type	Required
x	data to find maximum index	(-∞, ∞)	integer, real or complex	yes

Examples

```
y = max_index([1, 2, 3])
```
returns 2

```
y = max_index([3, 2, 1])
```
returns 0

Defined in

Built-in

See Also

min_index()

min_index()

Returns the index of the minimum

Syntax

y = min_index(x)

Arguments

Name	Description	Range	Type	Required
x	data to find the minimum index	(-∞, ∞)	integer, real or complex	yes

Examples

```
a = min_index([3, 2, 1])
```
returns 2

```
a = min_index([1, 2, 3])
```
returns 0

Defined in

Built in

See Also

max_index()

permute()

Permutes data based on the attached independents

Syntax

y = permute(data, permute_vector)

Arguments

Name	Description	Range	Type	Required
data	any N-dimensional square data (all inner independents must have the same value N)	(-∞, ∞)	integer, real, complex	yes
permute_vector	any permutation vector of the numbers 1 through N †	(-∞, ∞)	integer, real, complex	yes
† The permute_vector defaults to {N::1}, representing a complete reversal of the data with respect to its independent variables. If permute_vector has fewer than N entries, the remainder of the vector, representing the outer independent variables, is filled in. In this way, expressions remain robust when outer sweeps are added				

Examples

This example assumes that the variable data has three independent variables; therefore:

```
a = permute(data)
```
reverses the (three inner independents of) the data

```
a2 = permute(data, {3, 2, 1})
```
same as above

```
aOrig = permute(data, {1, 2, 3})
```
preserves the data

The example below assumes that a DC analysis has been performed with two independent variables, VGS and VDS, and IDS.i is the dependent variable. To see a plot of IDS vs VGS for different values of VDS, the data can be permuted as follows:

```
permutedData = permute(IDS.i,{2,1})
```

Defined in

Built in

See Also

plot_vs()

Notes/Equations

The permute() function is used to swap the order of the independent variables that are attached to a data variable. For example, a data could have two independent variables in a particular order. To swap the order so that it can be easily plotted, the order of the independents must be swapped. The permute() function can be used for this purpose.

The permute() function cannot be used to swap the rows and columns of a matrix. However, it can be used to swap the orders of the independent, even if the dependent is a matrix. For example, a parameter sweep of an S-parameter analysis.

plot_vs()

Attaches an independent to data for plotting

Syntax

y = plot_vs(dependent, independent)

Arguments

Name	Description	Range	Type	Required
dependent	any N-dimensional square data (all inner independents must have the same value N)	(-∞, ∞)	integer, real, complex	yes
independent	independent variable	(-∞, ∞)	integer, real	yes

Examples

```
a = [1, 2, 3]
b = [4, 5, 6]
c = plot_vs(a, b)
```

The first example above builds c with independent b, and dependent a

In the next example, assume that an S-parameter analysis has been performed with one swept variable Cval (of say 10 values) for 20 frequency points. The dependent data dbS11=db(S11) is of 2 dimension and Dependency of [10, 20]. A standard plot would display dbS11 vs freq(the inner independent), for 10 values of Cval. Instead to plot dbS11 vs Cval, the plot_vs() function can used as follows:

```
plot_vs(dbS11, Cval)
```

To plot dbS11 for half the values of Cval:

```
CvalH = Cval/2
plot_vs(dbS11, CvalH)
```

The last example below assumes that a DC analysis has been performed with two independent variables, Vgs and Vds, and Ids.i as the dependent variable. To see a plot of Ids vs Vgs for different values of Vds the data can be plotted as follows:

```
plot_vs(Ids.i, Vgs)
```

Defined in

$HPEESOF_DIR/expressions/ael/display_fun.ael

See Also

indep(), permute(), vs()

Notes/Equations

When using plot_vs(), the independent and dependent data should be the same size (i.e., not irregular). This function works as follows:

- Checks to see if the argument "independent", is an independent of argument "depend" or argument "independent" is dis-similar to independent of argument "depend".
- If one of the above conditions is met, then the data is swapped or sliced, and the new result formed with the argument "independent" is returned.

set_attr()

Sets the data attribute

Syntax

y = set_attr(data, "attr_name", attribute_value)

Arguments

Name	Description	Range	Type	Required
data	data	(-∞, ∞)	integer, real, complex	yes
attr_name	name of the attribute		string	yes
attribute_value	value of the attribute	(-∞, ∞)	boolean, integer, real, complex	yes

Examples

```
a = set_attr(data, "TraceType", "Spectral")
a = set_attr(data, "TraceType", "Histogram")
```

Defined in

Built in

See Also

get_attr()

size()

Returns the row and column size of a vector or matrix

Syntax

y = size(x)

Arguments

Name	Description	Range	Type	Required
x	data	(-∞, ∞)	integer, real, complex	yes

Examples

Given 2-port S-parameters versus frequency, and given 10 frequency points. Then for ten 2×2 matrices, size() returns the dimensions of the S-parameter matrix, and its companion function sweep_size() returns the size of the sweep:

```
Y = size(S)
```
returns {2, 2}

```
Y = sweep_size(S)
```
returns 10

Defined in

Built in

See Also

sweep_size()

sort()

This measurement returns a sorted variable in ascending or descending order. The sorting can be done on the independent or dependent variables. String values are sorted by folding them to lower case

Syntax

y = sort(data, sortOrder, indepName)

Arguments

Name	Description	Range	Type	Default	Required
data	data to be sorted (multidimensional scalar variable)	(-∞, ∞)	integer, real or complex		yes
sortOrder	sorting order	"ascending" or "descending"	string	"ascending"	no
indepName	specify the name of the independent variable for sorting		string	dependent value †	no
† if indepName not specified, the sorting is done on the dependent					

Examples

```
a = sort(data)
a = sort(data, "descending", "freq")
```

Defined in

Built in

sweep_dim()

Returns the dimensionality of the data

Syntax

y = sweep_dim(x)

Arguments

Name	Description	Range	Type	Required
x	data	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = sweep_dim(1)
```
returns 0

```
a = sweep_dim([1, 2, 3])
```
returns

Defined in

Built in

See Also

sweep_size()

sweep_size()

Returns the sweep size of a data object

Syntax

y = sweep_size(x)

Arguments

Name	Description	Range	Type	Required
x	data	(-∞, ∞)	integer, real, complex	yes

Examples

Given 2-port S-parameters versus frequency, and given 10 frequency points, there are then ten 2 × 2 matrices. sweep_size() is used to return the sweep size of the S-parameter matrix, and its companion function size() returns the dimensions of the S-parameter matrix itself:

```
a = sweep_size(S)
```
returns 10

```
a = size(S)
```
returns {2, 2}

Irregular data:

Assume that the data is 3 dimensional with the last dimension being irregular. The independents are: Vsrc, size, time with dimension [3,2,irreg]. Then:

```
SwpSz=sweep_size(data)
will return:
__SIZE                  SwpSz
Vsrc=1.0,size=1.0
                  1       20
Vsrc=1.0,size=2.0
                  1       21
Vsrc=2.0,size=1.0
                  1       59
Vsrc=2.0,size=2.0
                  1       61
Vsrc=3.0,size=1.0
                  1       76
Vsrc=3.0,size=2.0
                  1       78
```

where `__SIZE` is an independent added by the sweep_size() function.

sweep_size(SwpSz) would return the correct size of the two outer variables:

```
(1)         (2)         (3)
 3           2           1
```

Defined in

Built in

See Also

size(), sweep_dim()

Notes/Equations

For regular data, this function returns a vector with an entry corresponding to the length of each sweep. For irregular data, the function returns a multi-dimensional data, which needs to be processed further to get the size. See example above.

type()

Returns the type of the data

Syntax

y = type(x)

Arguments

Name	Description	Range	Type	Required
x	data to find the type	(-∞, ∞)	integer, real, complex, string	yes

Examples

```
a = type(1)
```
returns “Integer”

```
a = type(1.1)
```
returns "Real"

```
a = type(1i)
```
returns “Complex”

```
a = type("type")
```
returns “String”

Defined in

Built in

See Also

what()

vs()

Attaches an independent to dependent data

Syntax

y = vs(dependent, independent, indepName)

Arguments

Name	Description	Range	Type	Required
dependent	dependent values	(-∞, ∞)	integer, real	yes
independent	independent values	(-∞, ∞)	integer, real, string, complex	yes
indepName	independent name		string	no

Examples

```
a = [1, 2, 3]
b = [4, 5, 6]
c = vs(a, b)
```

builds c with independent b, and dependent a

Defined in

Built in

See Also

indep(), plot_vs()

Notes/Equations

Use the plot_vs() function to plot the dependent with the order of independent changed. For example, to plot Ids vs Vgs, in a DC analysis data with two independent variables, Vgs and Vds, and a dependent variable Ids.i, use as below:

plot_vs(Ids.i, Vgs)

what()

Returns size and type of data

Syntax

y = what(x, DisplayBlockName)

Arguments

Name	Description	Range	Type	Default	Required
x	data	(-∞, ∞)	integer, real, complex, string		yes
DisplayBlockName	Displays block name	[0, 1] †	integer	0	no
† If DisplayBlockName equals 0, no block name is specified (default behavior). If DisplayBlockName equals 1, then block name is displayed. If DisplayBlockName is not equal to 0 or 1, it defaults to 0					

Examples

```
x = [10,20,30,40]
y = what(x)
```

y
Dependency : [ ] Num. Points : [4] Matrix Size : scalar Type : Integer

Defined in

Built in

See Also

type()

Notes/Equations

This function is used to determine the dimensions of a piece of data, the attached independents, the type, and (in the case of a matrix) the number of rows and columns. Use what() by entering a listing column and using the trace expression what(x).

write_var()

Writes dataset variables to a file

Syntax

y = write_var(FileName, WriteMode, Comment, Delimiter, Format, Precision, Var1, Var2,...,VarN)

Arguments

Name	Description	Range	Type	Default	Required
FileName	Name of the output file		string		yes
WriteMode	Describes the write mode - overwrite or append	"W"|"A" †	string		yes
Comment	Text to be written at the top of the file		string	""	no
Delimiter	Delimiter that separates the data		string	""	no
Format	Format of the data	"f"|"s" ‡	string	"f"	no
Precision	precision of the data	[1, 64]	integer	6	no
Var1,...,VarN	Data variables to be written		dataset variable		yes

† WriteMode: "W" - overwrite the file, "A" - append to the file
‡ Format: "f" - full notation, "s" - scientific notation

Examples

```
write_var_f=write_var("output_S21.txt","W","! Freq real(S21)
imag(S21)"," ", "f", freq, S21)
```

writes S21 to the output file output_S21.txt as:

```
! Freq real(S21) imag(S21)
1000000000 0.60928892074273 -0.10958342264718
2000000000 0.52718867597783 -0.13319167002392
3000000000 0.4769067837712 -0.12080489345341
```

```
wv_ib=write_var("output_hbIb.txt","W","! HB Ib.i", " ", "f", freq, Ib.i)
```

write the Harmonic Balance frequency and current Ib.i to the output file output_hbIb.txt.

Defined in

$HPEESOF_DIR/expressions/ael/utility_fun.ael

See Also

indep()

Notes/Equations

This function can be used to write multiple dataset variables to a file. Currently only 1 dimensional data is supported. All variables that are to be written must be of the same size. Each variable data is written in column format. Complex data type is written in 2 columns as real and imaginary.

Chapter 6: Harmonic Balance Functions

This chapter describes the Harmonic Balance functions in detail. The functions are listed in alphabetical order.

C,D,I,M,P,S,T,V

Working with Harmonic Balance Data

Harmonic Balance (HB) Analysis produces complex voltages and currents as a function of frequency or harmonic number. A single analysis produces 1-dimensional data. Individual harmonic components can be indexed by means of "[ ]". Multi-tone HB also produces 1-dimensional data. Individual harmonic components can be indexed as usual by means of "[ ]". However, the function "mix()" on page 6-13 provides a convenient way to select a particular mixing component.

carr_to_im()

This measurement gives the suppression (in dB) of a specified IMD product below the fundamental power at the output port

Syntax

y = carr_to_im(vOut, fundFreq, imFreq, Mix)

Arguments

Name	Description	Range	Type	Required
vOut	signal voltage at the output port	[0, ∞)	real, complex	yes
fundFreq	harmonic frequency indices for the fundamental frequency	(-∞, ∞)	integer array	yes
imFreq	harmonic frequency indices for the IMD product of interest	(-∞, ∞)	integer array	yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	(-∞, ∞)	integer array	no
† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums				

Examples

```
a = carr_to_im(out, {1, 0}, {2, -1})
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ip3_out()

cdrange()

Returns compression dynamic range

Syntax

y = cdrange(nf, inpwr_lin, outpwr_lin, outpwr)

Arguments

Name	Description	Range	Type	Required
nf	noise figure at the output port	[0, ∞)	real	yes
inpwr_lin	input power in the linear region	[0, ∞)	real	yes
outpwr_lin	output power in the linear region	[0, ∞)	real	yes
outpwr	output power at 1 dB compression	[0, ∞)	real	yes

Examples

```
a = cdrange(nf2, inpwr_lin, outpwr_lin, outpwr)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

sfdr()

Notes/Equations

Used in XDB simulation.

The compressive dynamic range ratio identifies the dynamic range from the noise floor to the 1-dB gain-compression point. The noise floor is the noise power with respect to the reference bandwidth.

dc_to_rf()

This measurement computes the DC-to-RF efficiency of any part of the network

Syntax

y = dc_to_rf(vPlusRF, vMinusRF, vPlusDC, vMinusDC, currentRF, currentDC, harm_freq_index, Mix)

Arguments

Name	Description	Range	Type	Required
vPlusRF	voltage at the positive terminal	(-∞, ∞)	real, complex	yes
vMinusRF	voltage at the negative terminal	(-∞, ∞)	real, complex	yes
vPlusDC	DC voltage at the positive terminal	(-∞, ∞)	real, complex	yes
vMinusDC	DC voltage at the negative terminal	(-∞, ∞)	real, complex	yes
currentRF	RF current for power calculation	(-∞, ∞)	real, complex	yes
currentDC	DC current for power calculation	(-∞, ∞)	real, complex	yes
harm_freq_index	harmonic index of the RF frequency at the output por	(-∞, ∞)	integer array	yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	(-∞, ∞)	matrix	no
† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums				

Examples

```
a = dc_to_rf(vrf, 0, vDC, 0, I_Probe1.i, SRC1.i, {1,0})
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

ifc()

This measurement gives the RMS current value of one frequency-component of a harmonic balance waveform

Syntax

y = ifc(iOut, harm_freq_index, Mix)

Arguments

Name	Description	Range	Type	Required
iOut	current through a branch	(-∞, ∞)	real, complex	yes
harm_freq_index	harmonic index of the desired frequency †	(-∞, ∞)	integer array	yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis † †	(-∞, ∞)	matrix	no

† Note that the harm_freq_index argument's entry should reflect the number of tones in the harmonic balance controller. For example, if one tone is used in the controller, there should be one number inside the braces; two tones would require two numbers separated by a comma.
† † Mix is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums

Examples

The following example is for two tones in the Harmonic Balance controller:

```
y = ifc(I_Probe1.i, {1, 0})
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pfc(), vfc()

Notes/Equations

This function should not be used for DC measurements. If used, the results will be less by a factor of sqrt(2.0).

ip3_in()

This measurement determines the input third-order intercept point (in dBm) at the input port with reference to a system output port

Syntax

y = ip3_in(vOut, ssGain, fundFreq, imFreq, zRef, Mix)

Arguments

Name	Description	Range	Type	Default	Required
vOut	signal voltage at the output port	[0, ∞)	real, complex		yes
ssGain	small signal gain in dB	[0, ∞)	real		yes
fundFreq	harmonic frequency indices for the fundamental frequency	(-∞, ∞)	integer array		yes
imFreq	harmonic frequency indices for the intermodulation frequency	(-∞, ∞)	integer array		yes
zRef	reference impedance	(-∞, ∞)	real, complex	50.0	no
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	(-∞, ∞)	integer array		no
† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums					

Examples

```
y = ip3_in(vOut, 22, {1, 0}, {2, -1}, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ip3_out(), ipn()

Notes/Equations

To measure the third-order intercept point, you must setup a Harmonic Balance simulation with the input signal driving the circuit in the linear range. Input power is typically set 10 dB below the 1 dB gain compression point. If you simulate the circuit in the nonlinear region, the calculated results will be incorrect.

ip3_out()

This measurement determines the output third-order intercept point (in dBm) at the system output port

Syntax

y = ip3_out(vOut, fundFreq, imFreq, zRef, Mix)

Arguments

Name	Description	Range	Type	Default	Required
vOut	signal voltage at the output port	[0, ∞)	real, complex		yes
fundFreq	harmonic frequency indices for the fundamental frequency	(-∞, ∞)	integer array		yes
imFreq	harmonic frequency indices for the intermodulation frequency	(-∞, ∞)	integer array		yes
zRef	reference impedance	(-∞, ∞)	real, complex	50.0	no
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	(-∞, ∞)	integer array		no
† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums					

Examples

```
y = ip3_out(vOut, {1, 0}, {2, -1}, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ip3_in(), ipn()

Notes/Equations

To measure the third-order intercept point, you must setup a Harmonic Balance simulation with the input signal driving the circuit in the linear range. Input power

is typically set 10 dB below the 1 dB gain compression point. If you simulate the circuit in the nonlinear region, the calculated results will be incorrect.

ipn()

This measurement determines the output nth-order intercept point (in dBm) at the system output port

Syntax

y = ipn(vPlus, vMinus, iOut, fundFreq, imFreq, n, Mix)

Arguments

Name	Description	Range	Type	Required
vPlus	voltage at the positive output terminal	(-∞, ∞)	real, complex	yes
vMinus	voltage at the negative output terminal	(-∞, ∞)	real, complex	yes
iOut	current through a branch	(-∞, ∞)	real, complex	yes
fundFreq	harmonic indices of the fundamental frequency	(-∞, ∞)	integer array	yes
imFreq	harmonic indices of the intermodulation frequency	(-∞, ∞)	integer array	yes
n	order of the intercept	[1, ∞)	integer	yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	(-∞, ∞)	matrix	no
† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums				

Examples

```
y = ipn(vOut, 0, I_Probe1.i, {1, 0}, {2, -1}, 3)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ip3_in(), ip3_out()

Notes/Equations

To measure the third-order intercept point, you must setup a Harmonic Balance simulation with the input signal driving the circuit in the linear range. Input power is typically set 10 dB below the 1 dB gain compression point. If you simulate the circuit in the nonlinear region, the calculated results will be incorrect.

it()

This measurement converts a harmonic-balance current frequency spectrum to a time-domain current waveform

Syntax

it(iOut, tmin, tmax, numOfPnts)

Arguments

Name	Description	Range	Type	Default	Required
iOut	current through a branch	(-∞, ∞)	real, complex		yes
tmin	start time	[0, ∞)	real	0	no
tmax	stop time	[0, ∞)	real	2*cycle time	no
numOfPnts	number of points	[0, ∞)	integer	101	no

Examples

```
y = it(I_Probe1.i, 0, 10nsec, 201)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

vt()

mix()

Returns a component of a spectrum based on a vector of mixing indices

Syntax

mix(xOut, harmIndex, Mix)

Arguments

Name	Description	Range	Type	Default	Required
xOut	voltage or a current spectrum	(-∞, ∞)	real, complex		yes
harmIndex	desired vector of harmonic frequency indices (mixing terms)	(-∞, ∞)	integer array		yes
Mix	variable consisting of all possible vectors of harmonic frequency indices (mixing terms) in the analysis	(-∞, ∞)	matrix	†	no
† Mix, is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums. This is not required if the voltage is a named node or if the current is from a probe					

Examples

In the example below, vOut is the voltage at a named node. Therefore the third argument Mix is not required.

```
y = mix(vOut, {2, -1})
```

In the example below, vExp is an expression. Therefore in the mix() function, the third argument Mix is required.

```
vExp=vOut*vOut/50
z = mix(vExp, {2, -1}, Mix)
```

Defined in

Built in

See Also

find_index()

Notes/Equations

Used in Harmonic Balance analysis.

It is used to obtain the mixing component of a voltage or a current spectrum corresponding to particular harmonic frequency indices or mixing terms.

pae()

This measurement computes the power-added efficiency (in percent) of any part of the circuit

Syntax

y = pae(vPlusOut, vMinusOut, vPlusIn, vMinusIn, vPlusDC, vMinusDC, iOut, iIn, iDC, outFreq, inFreq)

Arguments

Name	Description	Range	Type	Required
vPlusOut	output voltage at the positive terminal	(-∞, ∞)	real, complex	yes
vMinusOut	output voltage at the negative terminal	(-∞, ∞)	real, complex	yes
vPlusIn	input voltage at the positive terminal	(-∞, ∞)	real, complex	yes
vMinusIn	input voltage at the negative terminal	(-∞, ∞)	real, complex	yes
vPlusDC	DC voltage at the positive terminal	(-∞, ∞)	real, complex	yes
vMinusDC	DC voltage at the negative terminal	(-∞, ∞)	real, complex	yes
iOut	output current	(-∞, ∞)	real, complex	yes
iIn	input current	(-∞, ∞)	real, complex	yes
iDC	DC current	(-∞, ∞)	real, complex	yes
outFreq	harmonic indices of the fundamental frequency at the output port	(-∞, ∞)	integer array	yes
inFreq	harmonic indices of the fundamental frequency at the input port	(-∞, ∞)	integer array	yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	(-∞, ∞)	matrix	no
† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums				

Examples

```
y = pae(vOut, 0, vIn, 0, v1, 0, I_Probe1.i, I_Probe2.i, I_Probe3.i, 1, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

db(), dbm()

pfc()

This measurement gives the RMS power value of one frequency component of a harmonic balance waveform

Syntax

y = pfc(vPlus, vMinus, iOut, harm_freq_index)

Arguments

Name	Description	Range	Type	Required
vPlus	voltage at the positive output terminal	(-∞, ∞)	real, complex	yes
vMinus	voltage at the negative output terminal	(-∞, ∞)	real, complex	yes
iOut	current through a branch	(-∞, ∞)	real, complex	yes
harm_freq_index	harmonic index of the desired frequency †	(-∞, ∞)	integer array	yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis † †	(-∞, ∞)	matrix	no

† Note that the harm_freq_index argument's entry should reflect the number of tones in the harmonic balance controller. For example, if one tone is used in the controller, there should be one number inside the braces; two tones would require two numbers separated by a comma.
† † Mix is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums

Examples

The following example is for two tones in the Harmonic Balance controller:

```
y = pfc(vOut, 0, I_Probe1.i, {1, 0})
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ifc(), vfc()

Notes/Equations

This function should not be used for DC measurements. If used, the results will be less by a factor of 2.0.

phase_gain()

Returns the gain associated with the phase (normally zero)

Syntax

y = phase_gain(Gain, DesiredPhase)

Arguments

Name	Description	Range	Type	Default	Required
Gain	Two dimensional data representing gain. E.g. Loop-gain of an oscillator.	(-∞, ∞)	complex		yes
DesiredPhase	A single value representing the desired phase.	(-∞, ∞)	real	0	no

Examples

It is assumed that a Harmonic Balance analysis has been performed at different power.

```
gainAtZeroPhase = phase_gain(Vout/Vin, 0)
```

returns the gain at zero phase.

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

pspec()

This measurement gives a power frequency spectrum in harmonic balance analyses

Syntax

y = pspec(vPlus, vMinus, iOut)

Arguments

Name	Description	Range	Type	Default	Required
vPlus	voltage at the positive node	(-∞, ∞)	real, complex		yes
vMinus	voltage at the negative node	(-∞, ∞)	real, complex		yes
iOut	current through a branch	(-∞, ∞)	real	0	no

Examples

```
a = pspec(vOut, 0, I_Probe1.i)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pt(), ispec(), vspec()

pt()

This measurement calculates the total power of a harmonic balance frequency spectrum

Syntax

y = pt(vPlus, vMinus, iOut)

Arguments

Name	Description	Range	Type	Default	Required
vPlus	voltage at the positive node	(-∞, ∞)	real, complex		yes
vMinus	voltage at the negative node	(-∞, ∞)	real, complex		yes
iOut	current through a branch	(-∞, ∞)	real	0	no

Examples

```
y = pt(vOut, 0, I_Probe1.i)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pspec()

remove_noise()

Removes noise floor data from noise data and returns an array

Syntax

nd = remove_noise(NoiseData, NoiseFloor)

Arguments

Name	Description	Range	Type	Required
NoiseData	Two dimensional array representing noise data	(-∞, ∞)	real, complex	yes
NoiseFloor	Single dimensional array representing noise floor	(-∞, ∞)	real, complex	yes

Examples

```
nd = remove_noise(vnoise, noiseFloor)
```

returns the noise data with the noise floor removed

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

Notes/Equations

Used in Harmonic Balance analysis.

NoiseData is [m,n] where m is receive frequency and n is interference offset frequency. If NoiseData is [m,n], then NoiseFloor must be [m]. If NoiseData - NosieFloor is less than zero, then -200 dBm is used.

sfdr()

Returns the spurious-free dynamic range

Syntax

y = sfdr(vOut, ssGain, nf, noiseBW, fundFreq, imFreq, zRef{, Mix})

Arguments

Name	Description	Range	Type	Default	Required
vOut	signal voltage at the output port	[0, ∞)	real, complex		yes
ssGain	small signal gain in dB	[0, ∞)	real		yes
nf	noise figure at the output port	[0, ∞)	real		yes
fundFreq	harmonic frequency indices for the fundamental frequency	(-∞, ∞)	integer array		yes
imFreq	harmonic frequency indices for the intermodulation frequency	(-∞, ∞)	integer array		yes
zRef	reference impedance	(-∞, ∞)	real, complex	50.0	no
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	(-∞, ∞)	integer array		no
† Mix is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums					

Examples

```
a = sfdr(vIn, 12, nf2, , {1, 0}, {2, -1}, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ip3_out()

Notes/Equations

Used in a Harmonic Balance and Small-signal S-parameter. It appears in the HB Simulation palette.

This measurement determines the spurious-free dynamic-range ratio for noise power with respect to the reference bandwidth.

To measure the third-order intercept point, you must setup a Harmonic Balance simulation with the input signal driving the circuit in the linear range. Input power is typically set 10 dB below the 1 dB gain compression point. If you simulate the circuit in the nonlinear region, the calculated results will be incorrect.

snr()

This measurement gives the ratio of the output signal power (at the fundamental frequency for a harmonic balance simulation) to the total noise power (in dB)

Syntax

y = snr(vOut, vOut.noise, fundFreq, Mix)

Arguments

Name	Description	Range	Type	Required
vOut	signal voltage at the output port	[0, ∞)	real, complex	yes
vOut.noise	noise voltage at the output port	[0, ∞)	real, complex	yes
fundFreq	harmonic frequency indices for the fundamental frequency †	(-∞, ∞)	integer array	yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis † †	(-∞, ∞)	integer array	no

† Note that fundFreq is not optional; it is required for harmonic balance simulations, but it is not applicable in AC simulations.
† † Mix is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums

Examples

```
a = snr(vOut, vOut.noise, {1, 0})
```
returns the signal-to-power noise ratio for a Harmonic Balance simulation.

```
a = snr(vOut, vOut.noise)
```
returns the signal-to-power noise ratio for an AC simulation.

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ns_pwr_int(), ns_pwr_ref_bw()

Notes/Equations

If the second argument is of higher dimension than the first, the noise bandwidth used for the purpose of computing snr will be equal to the frequency spacing of the innermost dimension of the noise data, instead of the standard value of 1 Hz.

spur_track()

Returns the maximum power of all signals appearing in a user-specifiable IF band, as a single RF input signal is stepped. If there is no IF signal appearing in the specified band, for a particular RF input frequency, then the function returns an IF signal power of -500 dBm

Syntax

IFspur = spur_track(vs(vout, freq), if_low, if_high, rout)

Arguments

Name	Description	Range	Type	Required
vout	IF output node name		string	yes
if_low	lowest frequency in the IF band	[0, ∞)	real	yes
if_high	highest frequency in the IF band	[0, ∞)	real	yes
rout	load resistance connected to the IF port, necessary for computing power delivered to the load	[0, ∞)	real	yes

Examples

```
IFspur = spur_track(vs(HB.VIF1, freq), Fiflow[0, 0], Fifhigh[0, 0], 50)
```

where:

VIF1 is the named node at the IF output.

Fiflow is the lowest frequency in the IF band.

Fifhigh is the highest frequency in the IF band.

50 is the IF load resistance.

Fiflow and Fifhigh are passed parameters from the schematic page (although they can be defined on the data display page instead.) These parameters, although single-valued on the schematic, become matrices when passed to the dataset, where each element of the matrix has the same value. The [0, 0] syntax just selects one element from the matrix.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

spur_track_with_if()

Notes/Equations

Used in Receiver spurious response simulations.

IFspur computed above will be the power in dBm of the maximum signal appearing in the IF band, versus RF input frequency. Note that it would be easy to modify the function to compute dBV instead of dBm.

This function is meant to aid in testing the response of a receiver to RF signals at various frequencies. This function shows the maximum power of all signals appearing in a user-specifiable IF band, as a single RF input signal is stepped. There could be fixed, interfering tones present at the RF input also, if desired. The maximum IF signal power may be plotted or listed versus the stepped RF input signal frequency. If there is no IF signal appearing in the specified band, for a particular RF input frequency, then the function returns an IF signal power of -500 dBm.

spur_track_with_if()

Returns the maximum power of all signals appearing in a user-specifiable IF band, as a single RF input signal is stepped. In addition, it shows the IF frequencies and power levels of each signal that appears in the IF band, as well as the corresponding RF signal frequency

Syntax

IFspur = spur_track_with_if(vs(vout, freq), if_low, if_high, rout)

Arguments

Name	Description	Range	Type	Required
vout	IF output node name		string	yes
if_low	lowest frequency in the IF band	[0, ∞)	real	yes
if_high	highest frequency in the IF band	[0, ∞)	real	yes
rout	load resistance connected to the IF port, necessary for computing power delivered to the load	[0, ∞)	real	yes

Examples

```
IFspur=spur_track_with_if(vs(HB.VIF1, freq), Fiflow[0, 0], Fifhigh[0, 0],
50)
```

where

VIF1 is the named node at the IF output.

Fiflow is the lowest frequency in the IF band.

Fifhigh is the highest frequency in the IF band.

50 is the IF load resistance.

Fiflow and Fifhigh are passed parameters from the schematic page (although they can be defined on the data display page instead.) These parameters, although single-valued on the schematic, become matrices when passed to the dataset, where each element of the matrix has the same value. The [0, 0] syntax just selects one element from the matrix.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

spur_track()

Notes/Equations

Used in Receiver spurious response simulations.

IFspur computed above will be the power in dBm of the maximum signal appearing in the IF band, versus RF input frequency. Note that it would be easy to modify the function to compute dBV instead of dBm.

This function is meant to aid in testing the response of a receiver to RF signals at various frequencies. This function, similar to the spur_track function, shows the maximum power of all signals appearing in a user-specifiable IF band, as a single RF input signal is stepped. In addition, it shows the IF frequencies and power levels of each signal that appears in the IF band, as well as the corresponding RF signal frequency. There could be fixed, interfering tones present at the RF input also, if desired. The maximum IF signal power may be plotted or listed versus the stepped RF input signal frequency.

thd_func()

This measurement returns the Total Harmonic Distortion percentage

Syntax

y = thd_func(v)

Arguments

Name	Description	Range	Type	Required
v	voltage	(-∞, ∞)	real, complex	yes

Examples

```
y = thd_func(Vload)
```

Defined In

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

ts()

Performs a frequency-to-time transform

Syntax

y= ts(x, tstart, tstop, numtpts, dim, windowType, windowConst, nptsspec)

Arguments

Name	Description	Range	Type	Default	Required
x	frequency-domain data to be transformed	(-∞, ∞)	real		yes
tstart	starting time	[0, ∞)	integer, real	0	no
tstop	stoping time	[0, ∞)	integer, real	tstop = tstart + 2.0/fabs(freq[0])	no
numtpts	number of time points	[1, ∞)	integer	101	no
dim	dimension to be transformed (not used currently)	[1, ∞)	integer	highest dimension	no
windowType	type of window to be applied to the data	[0, 9] †	integer, string	0	no
windowConst	window constant ‡	[0, ∞)	integer, real	0	no
nptsspec	number of first harmonics to be transformed	[1, NumFreqs]	integer	1	no

† The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the time-to-frequency transformation with normal gate shape setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13
‡ windowConst is not used if windowType is 8510

Examples

The following examples of ts assume that a Harmonic Balance simulation was performed with a fundamental frequency of 1 GHz and order = 8:

```
Y = ts(vOut)
```
returns the time series (0, 20ps, ... , 2ns)

```
Y = ts(vOut, 0, 1ns)
```
returns the time series (0, 10ps, ..., 1ns)

```
Y = ts(vOut, 0, 10ns, 201)
```
returns the time series (0, 50ps, ... , 10ns)

```
Y = ts(vOut, , , , , , , 3)
```
returns the time series (0, 20ps, ... , 2ns), but only uses harmonics from 1 to 3 GHz

Defined in

Built in

See Also

fft(), fs(), fspot()

Notes/Equations

Used in Harmonic Balance and Circuit Envelope simulations.

The dim argument is not used and should be left empty in the expression. Entering a value will have no impact on the results.

ts(x) returns the time domain waveform from a frequency spectrum. When x is a multidimensional vector, the transform is evaluated for each vector in the specified dimension. For example, if x is a matrix, then ts(x) applies the transform to every row of the matrix. If x is three dimensional, then ts(x) is applied in the lowest dimension over the remaining two dimensions. The dimension over which to apply the transform may be specified by dimension; the default is the lowest dimension (dimension=1). ts() originated in MDS and is similar to vt().

x must be numeric. It will typically be data from a Harmonic Balance analysis.

By default, two cycles of the waveform are produced with 101 points, starting at time zero, based on the lowest frequency in the input spectrum. These may be changed by setting tstart, tstop, or numtpts.

All of the harmonics in the spectrum will be used to generate the time domain waveform. When the higher-order harmonics are known not to contribute

significantly to the time domain waveform, only the first n harmonics may be requested for the transform, by setting nptsspec = n.

ts(x) can be used to process more than Harmonic Balance. For example, ts(x) can be used to convert AC simulation data to a time domain waveform using only one frequency point in the AC simulation.

Note that if the data does not have an explicit independent variable "freq", it is assumed to be starting at 0.0 and incremented in steps of 1. In some cases, this might lead to an incorrect time waveform. For example to obtain the time waveform of the second tone in a single tone analysis, using ts(Vout[2]) would give incorrect results. In this case use ts(Vout[2::3],,,,,,,1) to obtain the correct waveform.

In harmonic balance analysis if variables are swept, the data set saved has an inner most independent *harmindex* and a first dependent *freq*. In the case of argument *tstop* not being given the default is calculated using the dependent variable *freq*. But if the argument *x* in the ts() function is arithmetically operated or sub-indexed, the dependent *freq* is not maintained and in such cases the ts() function returns incorrect *time* values. This can be prevented by first using the ts() function on such data and then obtaining the necessary data. The example below illustrates this point.

Assume that the harmonic balance analysis has swept variable *Pin*. In this case the data has two independents [Pin, harmindex]. If *Idd.i* and *Iout.i* are 2 currents then the expression below:

```
tsERR = ts(Idd.i - Iout.i)
```
would return the incorrect time axis values. This can be solved by the expression:

```
tsWORKS = ts(Idd.i) - ts(Iout.i)
```

Similarly the expression:

```
tsERR1 = ts(Idd.i[0,::])
```
would return the incorrect time axis values. This can be solved by the expressions:

```
ts_Idd = ts(Idd.i)
ts_Idd_0 = ts_Idd[0,::]
```

vfc()

This measurement gives the RMS voltage value of one frequency-component of a harmonic balance waveform

Syntax

y = vfc(vPlus, vMinus, harm_freq_index)

Arguments

Name	Description	Range	Type	Required
vPlus	voltage at the positive output terminal	(-∞, ∞)	real, complex	yes
vMinus	voltage at the negative output terminal	(-∞, ∞)	real, complex	yes
harm_freq_index	harmonic index of the desired frequency †	(-∞, ∞)	integer array	yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis † †	(-∞, ∞)	matrix	no

† Note that the harm_freq_index argument's entry should reflect the number of tones in the harmonic balance controller. For example, if one tone is used in the controller, there should be one number inside the braces; two tones would require two numbers separated by a comma.
† † Mix is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums

Examples

The following example is for two tones in the Harmonic Balance controller:

```
a = vfc(vOut, 0, {1, 0})
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ifc(), pfc()

Notes/Equations

This function should not be used for DC measurements. If used, the results will be less by a factor of sqrt(2.0).

vspec()

Returns the voltage frequency spectrum

Syntax

y = vspec(vPlus, vMinus)

Arguments

Name	Description	Range	Type	Required
vPlus	voltage at the positive node	(-∞, ∞)	real, complex	yes
vMinus	voltage at the negative node	(-∞, ∞)	real, complex	yes

Examples

```
a = vspec(v1, v2)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ispec(), pspec()

Notes/Equations

Used in Harmonic Balance analysis.

This measurement gives a voltage frequency spectrum across any two nodes. The measurement gives a set of RMS voltages at each frequency.

vt()

This measurement converts a harmonic-balance voltage frequency spectrum to a time-domain voltage waveform

Syntax

y = vt(vPlus, vMinus, tmin, tmax, numOfPnts)

Arguments

Name	Description	Range	Type	Default	Required
vPlus	voltage at the positive node	(-∞, ∞)	real, complex		yes
vMinus	voltage at the negative node	(-∞, ∞)	real, complex		yes
tmin	start time	[0, ∞)	real	0	no
tmax	stop time	[0, ∞)	real	2*cycle time	no
numOfPnts	number of points	[0, ∞)	integer	101	no

Examples

```
a = vt(vOut, 0, 0, 10nsec, 201)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

it(), ts()

Notes/Equations

The vt() function originated in Series IV and simply calls the frequency to time domain transformer function, ts(). In some cases, if default values are used for tmin, tmax, and numOfPnts, the proper results may not be obtained due to an insufficient number of points. In such cases, the appropriate values for tmin, tmax, and numOfPnts need to be used. This function uses default values for window type and window constant, and in certain cases the correct results may not be obtained due to this fact. In this situation, use the ts() function instead with the proper windowing. For more information, refer to the Notes/Equations section for the ts() function.

Chapter 7: Math Functions

This chapter describes the math functions in detail. The functions are listed in alphabetical order.

I,J, L

M,N,O

P,R

S,T,X,Z

Note You can generally use these functions with data from any type of analysis. They consist of traditional math (e.g., trigonometric functions and matrix operations) and other functions.

abs()

Returns the absolute value of a integer, real or complex number

Syntax

y = abs(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = abs(-45)
```
returns 45

Defined in

Built in

See Also

cint(), exp(), float(), int(), log(), log10(), pow(), sgn(), sqrt()

Notes/Equations

In the case of a complex number, the abs function accepts one complex argument and returns the magnitude of its complex argument as a positive real number.

acos()

Returns the inverse cosine, or arc cosine, in radians

Syntax

y = acos(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = acos(-1)
```
returns 3.142

Defined in

Built in

See Also

asin(), atan(), atan2()

Notes/Equations

Returned value ranges from 0 to pi.

acosh()

Returns the inverse hyperbolic cosine

Syntax

y = acosh(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = acosh(1.5)
```
returns 0.962

Defined in

Built in

See Also

acos(), asin(), atan(), atan2()

acot()

Returns the inverse cotangent

Syntax

y = acot(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = acot(1.5)
```
returns 0.588

Defined in

Built in

See Also

asin(), atan(), atan2()

acoth()

Returns the inverse hyperbolic cotangent

Syntax

y = acoth(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = acoth(1.5)
```
returns 0.805

Defined in

Built in

See Also

acot(), asin(), atan(), atan2()

asin()

Returns the inverse sine, or arc sine, in radians

Syntax

y = asin(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = asin(-1)
```
returns –1.571

Defined in

Built in

See Also

acos(), atan(), atan2()

asinh()

Returns the inverse hyperbolic sine

Syntax

y = asinh(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = asinh(.5)
```
returns 0.481

Defined in

Built in

See Also

asin(), acos(), atan(), atan2()

atan()

Returns the inverse tangent, or arc tangent, in radians

Syntax

y = atan(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = atan(-1)
```
returns –0.785

Defined in

Built in

See Also

acos(), asin(), atan2()

Notes/Equations

Returned value range is -pi/2 to pi/2.

atan2()

Returns the inverse tangent, or arc tangent, of the rectangular coordinates y and x

Syntax

y = atan2(y, x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes
y	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = atan2(1, 0)
```
returns 1.571

Defined in

Built in

See Also

acos(), asin(), atan()

Notes/Equations

Returned value range is -pi to pi. atan2(0,0) returns -pi/2.

atanh()

Returns the inverse hyperbolic tangent

Syntax

y = atanh(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = atanh(.5)
```
returns 0.549

Defined in

Built in

See Also

acos(), asin(), atan(), atan2()

Notes/Equations

Returned value ranges from 0 to pi.

ceil()

Given a real number, returns the smallest integer not less than its argument; that is, its argument rounded to the next highest number

Syntax

y = ceil(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	real	yes

Examples

```
a = ceil(5.27)
```
returns 6

Defined in

Built in

cint()

Given a non-integer real number, returns a rounded integer

Syntax

y = cint(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	real	yes

Examples

```
a = cint(45.6)
```
returns 46

```
a = cint(-10.7)
```
returns -11

Defined in

Built in

See Also

abs(), exp(), float(), int(), log(), log10(), pow(), sgn(), sqrt()

Notes/Equations

0.5 rounds up, -0.5 rounds down (up in magnitude).

cmplx()

Returns complex number given real and imaginary

Syntax

y = cmplx(x, y)

Arguments

Name	Description	Range	Type	Required
x	real part of complex number	(-∞, ∞)	integer, real	yes
y	imaginary part of complex number	(-∞, ∞)	integer, real	yes

Examples

```
a = cmplx(2, -1)
```
returns 2 – 1j

Defined in

Built in

See Also

complex(), imag(), real()

complex()

Returns complex number given real and imaginary

Syntax

y = complex(x, y)

Arguments

Name	Description	Range	Type	Required
x	real part of complex number	(-∞, ∞)	integer, real	yes
y	imaginary part of complex number	(-∞, ∞)	integer, real	yes

Examples

```
a = complex(2, –1)
```
returns 2 – 1j

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

See Also

cmplx(), imag(), real()

conj()

Returns the conjugate of a complex number

Syntax

y = conj(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	complex	yes

Examples

```
a = conj(3-4*j)
```

returns 3.000 + j4.000 or 5.000 / 53.130 in magnitude / degrees

Defined in

Built in

See Also

mag()

convBin()

Returns a binary string of an integer with n-digits

Syntax

y = convBin(val, num)

Arguments

Name	Description	Range	Type	Required
val	integer to be converted to a binary string	(-∞, ∞)	integer, real	yes
num	number of digits in the binary string	[0, ∞)	integer	yes

Examples

```
a = convBin(1064, 8)
```
returns 00101000

Defined in

Built in

See Also

convHex(), convInt(), convOct()

convHex()

Returns a hexadecimal string of an integer with n-digits

Syntax

y = convHex(val, num)

Arguments

Name	Description	Range	Type	Required
val	integer to be converted to a hexadecimal string	(-∞, ∞)	integer, real	yes
num	number of digits in the hexadecimal string	[0, ∞)	integer	yes

Examples

```
a = convHex(1064, 8)
```
returns 00000428

Defined in

Built in

See Also

convBin(), convOct(), convInt()

convInt()

Returns an integer of a binary, octal or hexadecimal number

Syntax

y = convInt(val, base)

Arguments

Name	Description	Range	Type	Required
val	string representation of the binary, octal or hexadecimal number to be converted		string	yes
base	base of the conversion	2|8|16 †	integer	yes
† base values: 2:binary, 8:octal, 16:hexadecimal				

Examples

```
b2I = convInt(“11100”, 2)
```
returns 28

```
o2I = convInt(“34”, 8)
```
returns 28

```
h2I = convInt(“1c”, 16)
```
returns 28

Defined in

Built in

See Also

convBin(), convHex(), convOct()

convOct()

Returns an octal string of an integer with n-digits

Syntax

y = convOct(val, num)

Arguments

Name	Description	Range	Type	Required
val	integer to be converted to a octal string	(-∞, ∞)	integer, real	yes
num	number of digits in the octal string	[0, ∞)	integer	yes

Examples

```
a = convOct(1064, 8)
```
returns 00002050

Defined in

Built in

See Also

convBin(), convHex(), convInt()

cos()

Returns the cosine

Syntax

y = cos(x)

Arguments

Name	Description	Range	Type	Required
x	number in radians	(-∞, ∞)	integer, real, complex	yes

Examples

```
y = cos(pi/3)
```
returns 0.500

Defined in

Built in

See Also

sin(), tan()

cosh()

Returns the hyperbolic cosine

Syntax

y = cosh(x)

Arguments

Name	Description	Range	Type	Required
x	number in radians	(-∞, ∞)	integer, real, complex	yes

Examples

```
y = cosh(0)
```
returns 1

```
y = cosh(1)
```
returns 1.543

Defined in

Built in

See Also

sinh(), tanh()

cot()

Returns the cotangent

Syntax

y = cot(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = cot(1.5)
```
returns 0.071

Defined in

Built in

See Also

tan(), tanh()

coth()

Returns the hyperbolic cotangent

Syntax

y = coth(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = coth(1.5)
```
returns 1.105

Defined in

Built in

See Also

cot(), tan(), tanh()

cum_prod()

Returns the cumulative product

Syntax

y = cum_prod(x)

Arguments

Name	Description	Range	Type	Required
x	data to find cumulative product	(∞, ∞)	integer, real or complex	yes

Examples

```
y = cum_prod(1)
```
returns 1.000

```
y = cum_prod([1, 2, 3])
```
returns [1.000, 2.000, 6.000]

```
y = cum_prod([i, i])
```
returns [i, i^2]

Defined in

Built in

See Also

cum_sum(), max(), mean(), min(), prod(), sum()

cum_sum()

Returns the cumulative sum

Syntax

y = cum_sum(x)

Arguments

Name	Description	Range	Type	Required
x	data to find cumulative sum	(-∞, ∞)	integer, real or complex	yes

Examples

```
y = cum_sum([1, 2, 3])
```
returns [1.000, 3.000, 6.000]

```
y = cum_sum([i, i]
```
returns [i,2i]

Defined in

Built in

See Also

cum_prod(), max(), mean(), min(), prod(), sum()

db()

Returns the decibel measure of a voltage ratio

Syntax

y = db(r, z1, z2)

Arguments

Name	Description	Range	Type	Default	Required
r	voltage ratio (vOut/vIn)	(-∞, ∞)	integer, real, complex		yes
z1	source impedance	(-∞, ∞)	integer, real, complex	50.0	no
z2	load impedance	(-∞, ∞)	integer, real, complex	50.0	no

Examples

```
y = db(100)
```
returns 40

```
y = db(8-6*j)
```
returns 20

Defined in

Built in

See Also

dbm(), pae()

Notes/Equations

dbValue = 20 log(mag(r)) - 10 log(zOutfactor/zInfactor)

zOutfactor = mag(z2)**2 / real (z2)

zInfactor = mag(z1)**2 / real (z1).

dbm()

Returns the decibel measure of a voltage referenced to a 1 milliwatt signal

Syntax

y = dbm(v, z)

Arguments

Name	Description	Range	Type	Default	Required
v	voltage (the peak voltage)	(-∞, ∞)	integer, real, complex		yes
z	impedance	(-∞, ∞)	integer, real, complex	50.0	no

Examples

```
y = dbm(100)
```
returns 50

```
y = dbm(8-6*j)
```
returns 30

Defined in

Built in

See Also

db(), pae()

Notes/Equations

The voltage is assumed to be a peak value. Signal voltages stored in the dataset from AC and harmonic balance simulations are in peak volts. However, noise voltages obtained from AC and HB simulations are in rms volts. Using the dbm() function with noise voltages will yield a result that is 3 dB too low unless the noise voltage is first converted to peak:

```
noise_power = dbm(vout.noise * sqrt(2));
```

Understanding the dbm() Function

Given a power Po in Watts, the power in *dB* is:

$$Po\_dBW = 10*\log(mag(Po/(1\ W)))$$

while the power in *dBm* is:

$$\begin{aligned} Po\_dBm &= 10*\log(mag(Po/(1\ mW))) \\ &= 10*\log(mag(Po/(1\ W)))+ 30 \\ &= Po\_dB + 30 \end{aligned}$$

Given a voltage Vo in Volts, the voltage in *dB* is:

$$V\_dBV = 20*\log(mag((Vo/(1\ V)))$$

This is the db() function - voltage in *dB* relative to 1V. Although dB is a dimensionless quantity, it is normal to attach dB to a value in order to differentiate it from the absolute value.

Given a real impedance Zo, the power-voltage relation is:

$$Po = (Vo)^2/(2*Zo)$$

Using the above, Po in *dBm* is then:

$$\begin{aligned} Po\_dBm &= 10*\log(mag(Po/(1\ W))) + 30 \\ &= 10*\log(mag((Vo/(1\ V))^2/(2*Zo/(1\ Ohm)))) + 30 \\ &= 10*\log(mag((Vo/(1\ V))^2)) - 10*\log(mag(2*Zo/(1\ Ohm))) + 30 \\ &= 20*\log(mag(Vo/(1\ V))) - 10*\log(mag(Zo/(50\ Ohm))) + 10 \end{aligned}$$

This is the dbm() function - voltage in dBm in a Zo environment.

dbmtow()

Converts dBm to watts

Syntax

wValue = dbmtow(P)

Arguments

Name	Description	Range	Type	Required
P	power expressed in dBm	[0, ∞)	real	yes

Examples

```
y = dbmtow(0)
```

returns .001 W

```
y = dbmtow(-10)
```

returns 1.000E-4 W

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

See Also

dbm(), wtodbm()

deg()

Converts radians to degrees

Syntax

y = deg(x)

Arguments

Name	Description	Range	Type	Required
x	number in radians	(-∞, ∞)	integer, real	yes

Examples

```
y = deg(1.5708)
```
returns 90

```
y = deg(pi)
```
returns 180

Defined in

Built in

See Also

rad()

diagonal()

Returns the diagonal of a square matrix as a matrix

Syntax

y = diagonal(Matrix)

Arguments

Name	Description	Range	Type	Required
Matrix	square matrix to find the diagonal	(-∞, ∞)	integer, real or complex	yes

Examples

```
mat = {{1,2,3},{4,5,6},{7,8,9}}
diag = diagonal(mat)
```
returns {1,5,9}

For a 2-port S-parameter analysis of 10 freq points:

```
diagS = diagonal(S)
```
returns S11 and S22 for each frequency point

Defined in

Built In

See Also

transpose(), inverse()

diff()

Calculates the simple numerical first derivative. Can be used to calculate group delay

Syntax

y = diff(data, pad)

Arguments

Name	Description	Range	Type	Default	Required
data	data to find numerical derivative	(-∞, ∞)	real, complex		yes
pad	pad the differentiated data with an extra value	[0, 1] †	integer	0	no
† If pad is 1, then the differentiated data is padded with an extra value (last value of differentiated data) to make it the same length as the data to be differentiated. If 0 (default) then the length of the differentiated data is one less than the length of data to be differentiated					

Examples

```
group_delay = -diff(unwrap(phaserad(S21),pi) )/ (2*pi)
```

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

See Also

dev_lin_phase(), integrate(), phasedeg(), phaserad(), ripple(), unwrap()

Notes/Equations

This function calculates the first derivative of the dependent data with respect to the inner independent value i.e. dy/dx. The function uses the simple forward finite-divided-difference formulas of 2 values. The error is O(h), where h is the independent step size. The error decreases with smaller values of h. If the data to be differentiated does not have an explicit independent-name, the differentiated data is given an independent name "diffX'.

erf()

Calculates the error function, the area under the Gaussian curve exp(-x**2)

Syntax

y = erf(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real	yes

Examples

```
a = -erf(0.1)
```
returns 0.112

```
a = -erf(0.2)
```
returns 0.223

Defined in

Built in

See Also

erfc()

erfc()

Calculates the complementary error function, or 1 minus the error function. For large x, this can be calculated more accurately than the plain error function

Syntax

y = erfc(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real	yes

Examples

```
a = -erfc(0.1)
```
returns 0.888

```
a = -erfc(0.2)
```
returns 0.777

Defined in

Built in

See Also

erf()

erfcinv()

Returns the inverse complementary error function as a real number

Syntax

y = erfcinv(x)

Arguments

Name	Description	Range	Type	Required
x	number	[0, 2] †	integer, real	yes
† For numbers outside the range, erfcinv returns < -+infinity> . If val is 0, erfcinv returns < infinity> . If val is 2, then < -infinity>				

Examples

```
res = erfcinv(0.5)
```
returns 0.477

```
res = erfcinv(1.9)
```
returns -1.163

Defined in

Built In

See Also

erfc()

erfinv()

Returns the inverse error function as a real number

Syntax

y = erfinv(x)

Arguments

Name	Description	Range	Type	Required
x	number	[-1, 1] †	integer, real	yes
† For numbers outside the range, erfinv returns < -+infinity> . If x is +1, erfinv returns < infinity> . If x is -1, then < -infinity>				

Examples

```
res= erfinv(-0.4)
```
returns -0.371

```
res= erfinv(0.8)
```
returns 0.906

Defined in

Built In

See Also

erf()

exp()

The exponential function is used to calculate powers of e. Given a complex number, x, the exp(x) function calculates e to the power of x (i.e. ex)

Syntax

y = exp(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = exp(1)
```
returns 2.71828

```
b = exp(1+j1)
```
returns 1.469 + j*2.287

Defined in

Built in

See Also

abs(), cint(), float(), int(), log(), log10(), pow(), sgn(), sqrt()

Notes/Equations

If

```
x = a+j*b
```

then

```
ex = ea+j*b = (ea)*(ej*b) = (ea)*(cos(b)+j*sin(b))
```

fft()

Performs the discrete Fourier transform

Syntax

y = fft(x, length)

Arguments

Name	Description	Range	Type	Required
x	data to be transformed	(-∞, ∞)	integer, real, complex	yes
length	length of the transform	[1, ∞)	integer	yes

Examples

```
fft([1, 1, 1, 1])
```
returns [4+0i, 0+0i]

```
fft([1, 0, 0, 0]
```
returns [1+0i, 1+0i]

```
fft(1, 4)
```
returns [1+0i, 1+0i]

Defined in

Built in

See Also

fs(), ts()

Notes/Equations

The fft() function uses a high-speed radix-2 fast Fourier transform when the length of x is a power of two. fft(x, n) performs an n-point discrete Fourier transform, truncating x if length(x) > n and padding x with zeros if length(x) < n.

fft() uses a real transform if x is real and a complex transform if x is complex. If the length of x is not a power of two, then a mixed radix algorithm based on the prime factors of the length of x is used.

The fft() function is designed to work with uniformly spaced waveforms. If a non-uniform waveform is input, then the output spectrum will be incorrect. For non-uniformly spaced data, use the fs() function.

fix()

Takes a real number argument, truncates it, and returns an integer value

Syntax

y = fix(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real	yes

Examples

```
a = fix(5.9)
```
returns 5

Defined in

Built in

float()

Converts an integer to a real (floating-point) number

Syntax

y = float(x)

Arguments

Name	Description	Range	Type	Required
x	number to convert	(-∞, ∞)	integer	yes

Examples

```
a = float(10)
```
returns 10.000

Defined in

Built in

See Also

abs(), cint(), int(), log10(), pow(), sgn(), sqrt()

floor()

Returns the largest integer not more than its argument from a real number, that is, returns its argument rounded to the next highest number

Syntax

y = floor(x)

Arguments

Name	Description	Range	Type	Required
x	number to convert	(-∞, ∞)	real	yes

Examples

```
a = floor(4.3)
```
returns 4

```
a = floor(5.6)
```
returns 5

Defined in

Built in

fmod()

Returns the remainder of the division of two real numbers

Syntax

y = fmod(fNum, fDenom)

Arguments

Name	Description	Range	Type	Required
fNum	Value of numerator	(-∞, ∞)	integer, real	yes
fDenom	Value of denominator	(-∞, ∞)	integer, real	yes

Examples

```
fmodV = fmod(4.2, 2.0)
```

returns 0.2

Defined In

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

hypot()

Returns the hypotenuse

Syntax

y = hypot(xVal, yVal)

Arguments

Name	Description	Range	Type	Required
xVal	Value of X	(-∞, ∞)	real, complex	yes
yVal	Value of Y	(-∞, ∞)	real, complex	yes

Examples

```
y = hypot(1,2)
```

returns 5

Defined In

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

identity()

Returns the identity matrix

Syntax

y = identity(rows, columns)

Arguments

Name	Description	Range	Type	Required
rows	number of rows †	[i, ∞)	integer	yes
columns	number of columns †	[i, ∞)	integer	no
† If one argument is supplied, then a square matrix is returned with ones on the diagonal and zeros elsewhere. If two arguments are supplied, then a matrix with size rows cols is returned, again with ones on the diagonal				

Examples

```
y = identity(2)
```

y(1,1)	y(1,2)	y(2,1)	y(2,2)
1	0	0	1

Defined in

Built in

See Also

ones(), zeros()

im()

Returns the imaginary component of a complex number

Syntax

y = im(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	complex	yes

Examples

```
a = im(1-1*j)
```
returns -1.000

Defined in

Built in

See Also

imag(), cmplx(), real()

imag()

Returns the imaginary component of a complex number

Syntax

y = imag(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	complex	yes

Examples

```
a = imag(1-1*j)
```
returns -1.000

Defined in

Built in

See Also

cmplx(), im(), real()

int()

Converts an real to an integer

Syntax

y = int(x)

Arguments

Name	Description	Range	Type	Required
x	number to convert	(-∞, ∞)	real	yes

Examples

```
a = int(4.3)
```
returns 4

Defined in

Built in

See Also

abs(), cint(), exp(), float(), log10(), pow(), sgn(), sqrt()

integrate()

Returns the integral of data

Syntax

y = integrate(data, start, stop, incr)

Arguments

Name	Description	Range	Type	Default	Required
data	data to be intergated	(-∞, ∞)	integer, real		yes
start	starting value of the integration	(-∞, ∞)	integer, real	first point in the data	no
stop	stop value of the integration	(-∞, ∞)	integer, real †	last point in the data	no
incr	increment	[0, ∞)	integer, real	(stop - start)/(# data points - 1)	no
† stop can be an array					

Examples

```
x = [0::0.01::1.0]
y = vs(2*exp(-x*x) / sqrt(pi), x)
z = integrate(y, 0.1, 0.6, 0.001)
```

returns 0.491

```
xx = [1::0.1::2]
yy = vs(sin(xx),xx)
Stop = [1.9,2.0]
intgYY=integrate(yy,1,Stop,0.1)
```

returns [0.767, 0.958]

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

diff()

Notes/Equations

Returns the integral of data from start to stop with increment incr using the composite trapezoidal rule on uniform subintervals. The Stop limit can be an array of values. In this case, the function returns integration for limits [start, stop[0]], [start, stop[1]], etc.

interp()

Returns linearly interpolated data between start and stop with increment

Syntax

y = interp(Data, Start, Stop, Increment)

Arguments

Name	Description	Range	Type	Default	Required
Data	Data to be interpolated	(-∞, ∞)	integer, real		yes
Start	Independent value specifying start	(-∞, ∞) †	integer, real	First data point	no
Stop	Independent value specifying stop	(-∞, ∞) †	integer, real	Last data point	no
Increment	Increment between interpolated data points	(-∞, ∞)	integer, real	(Stop-Start)/NumDataPoints ‡	no

† first value of independent≥ Start≤ last value's independent
Start≥ Stop≤ last value's independent
‡ Where NumDataPoints is number of original data points

Examples

```
y = interp(data{, start, stop{, incr}})
```

Defined in

Built in

inverse()

Returns the inverse of a matrix

Syntax

y = inverse(Matrix)

Arguments

Name	Description	Range	Type	Required
Matrix	square matrix to find the inverse	(-∞, ∞)	integer, real or complex	yes

Examples

```
inverse({{1, 2}, {3, 4}})
```
returns {{–2, 1}, {1.5, –0.5}}

Defined in

Built in

See Also

diagonal(), transpose()

jn()

Computes the Bessel function of the first kind and returns a real number

Syntax

y = jn(n, x)

Arguments

Name	Description	Range	Type	Required
n	order	(-∞, ∞)	integer	yes
x	value for which the Bessel value is to be found	(-∞, ∞)	real	yes

Examples

```
jn0_15 = jn(0, 15)
```
returns -0.014

```
jn1_xV = jn(1, 5.23)
```
returns -0.344

```
jn10_15 = jn(10, 15)
```
returns -0.09

Defined in

in-built

ln()

Returns the natural logarithm (ln)

Syntax

y = ln(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = ln(e)
```
returns 1

Defined in

Built in

See Also

abs(), cint(), exp(), float(), int(), pow(), sgn(), sqrt()

log()

Returns the base 10 logarithm of an integer or real number

Syntax

y = log(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = log(10)
```
returns 1

```
a = log(0+0i)
```
returns NULL and an error message "log of zero"

Defined in

Built in

See Also

abs(), cint(), exp(), float(), int(), log10(), pow(), sgn(), sqrt()

log10()

Returns the base 10 logarithm of an integer or real number

Syntax

y = log10(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = log10(10)
```
returns 1

```
a = log10(0+0i)
```
returns NULL and an error message "log of zero"

Defined in

Built in

See Also

abs(), cint(), exp(), float(), int(), log(), pow(), sgn(), sqrt()

mag()

Returns the magnitude

Syntax

y = mag(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = mag(3-4*j)
```

returns 5.000

Defined in

Built in

See Also

conj()

max()

Returns the maximum value

Syntax

y = max(x)

Arguments

Name	Description	Range	Type	Required
x	data to find max	(-∞, ∞)	integer, real or complex	yes

Examples

```
a = max([1, 2, 3])
```
returns 3

Defined in

Built in

See Also

cum_prod(), cum_sum(), max2(), mean(), min(), prod(), sum()

max_outer()

Computes the maximum across the outer dimension of two-dimensional data

Syntax

y = max_outer(data)

Arguments

Name	Description	Range	Type	Required
data	2-dimensional data to find max	(-∞, ∞)	integer, real, complex	yes

Examples

```
y = max_outer(data)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistcal_fun.ael

See Also

fun_2d_outer(), mean_outer(), min_outer()

Notes/Equations

The max function operates on the inner dimension of two-dimensional data. The max_outer function just calls the fun_2d_outer function, with max being the applied operation. As an example, assume that a Monte Carlo simulation of an amplifier was run, with 151 random sets of parameter values, and that for each set the S-parameters were simulated over 26 different frequency points. S21 becomes a [151 Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being frequency, and the outer dimension being Monte Carlo index. Now, assume that it is desired to know the maximum value of the S-parameters at each frequency. Inserting an equation max(S21) computes the maximum value of S21 at each Monte Carlo iteration. If S21 is simulated from 1 to 26 GHz, it computes the maximum value over this frequency range, which usually is not very useful. Inserting an equation max_outer(S21) computes the maximum value of S21 at each Monte Carlo iteration.

max2()

Returns the larger value of two numeric values, or NULL if parameters are invalid

Syntax

y = max2(x, y)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes
y	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = max2(1.5, -1.5)
```
returns 1.500

Defined in

Built in

See Also

cum_prod(), cum_sum(), max(), mean(), min(), prod(), sum()

min()

Returns the minimum value

Syntax

y = min(x)

Arguments

Name	Description	Range	Type	Required
x	data to find min	(-∞, ∞)	integer, real or complex	yes

Examples

```
a = min([1, 2, 3])
```
returns 1

Defined in

Built in

See Also

cum_prod(), cum_sum(), max(), max2(), mean(), prod(), sum()

min_outer()

Computes the minimum across the outer dimension of two-dimensional data

Syntax

y = min_outer(data)

Arguments

Name	Description	Range	Type	Required
data	2-dimensional data to find min	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = min_outer(data)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistcal_fun.ael

See Also

fun_2d_outer(), max_outer(), mean_outer(),

Notes/Equations

The min function operates on the inner dimension of two-dimensional data. The min_outer function just calls the fun_2d_outer function, with min being the applied operation. As an example, assume that a Monte Carlo simulation of an amplifier was run, with 151 random sets of parameter values, and that for each set the S-parameters were simulated over 26 different frequency points. S21 becomes a [151 Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being frequency, and the outer dimension being Monte Carlo index. Now, assume that it is desired to know the minimum value of the S-parameters at each frequency. Inserting an equation min(S21) computes the minimum value of S21 at each Monte Carlo iteration. If S21 is simulated from 1 to 26 GHz, it computes the minimum value over this frequency range, which usually is not very useful. Inserting an equation min_outer(S21) computes the minimum value of S21 at each Monte Carlo iteration.

min2()

Returns the lesser value of two numeric values, or NULL if parameters are invalid

Syntax

y = min2(x, y)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes
y	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = min2(1.5, -1.5)
```
returns -1.500

Defined in

Built in

See Also

cum_prod(), cum_sum(), max(), max2(), mean(), min(), prod(), sum()

num()

Returns an integer that represents an ASCII numeric value of the first character in the specified string

Syntax

y = num(str)

Arguments

Name	Description	Type	Required
str	string to convert to integer	string	yes

Examples

```
a = num("/users/myhome/fullpath")
```
returns 47

```
a = num("alpha")
```
returns 97

Defined in

Built in

ones()

Returns ones matrix

Syntax

y = ones(rows, columns)

Arguments

Name	Description	Range	Type	Required
rows	number of rows †	[i, ∞)	integer	yes
columns	number of columns †	[i, ∞)	integer	no
† If only one argument is supplied, then a square matrix is returned. If two are supplied, then a matrix of ones with size rows X cols is returned				

Examples

```
a = ones(2)
```
returns {{1, 1}, {1, 1}}

Defined in

Built in

See Also

identity(), zeros()

phase()

Phase in degrees

Syntax

y = phase(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	complex	yes

Examples

```
a = phase(1*i)
```
returns 90

```
a = phase(1+1i)
```
returns 45

Defined in

Built-in

See Also

phaserad()

phasedeg()

Phase in degrees

Syntax

y = phasedeg(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	complex	yes

Examples

```
a = phasedeg(1*i)
```
returns 90

```
a = phasedeg(1+1i)
```
returns 45

Defined in

Built-in

See Also

dev_lin_phase(), diff(), phase(), phaserad(), ripple(), unwrap()

phaserad()

Phase in radians

Syntax

y = phaserad(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	complex	yes

Examples

```
a = phaserad(1*i)
```
returns 1.5708

```
a = phaserad(1+1i)
```
returns 0.785398

Defined in

Built in

See Also

dev_lin_phase(), diff(), phase(), phasedeg(), ripple(), unwrap()

polar()

Builds a complex number from magnitude and angle (in degrees)

Syntax

y = polar(mag, angle)

Arguments

Name	Description	Range	Type	Required
mag	magnitude part of complex number	(-∞, ∞)	integer, real	yes
angle	angle part of complex number	(-∞, ∞)	integer, real	yes

Examples

```
a = polar(1, 90)
```
returns 0+1i

```
a = polar(1, 45)
```
returns 0.707107+0.707107i

Defined in

Built in

pow()

Raises a number to a given power

Syntax

yPow = pow(x, y)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real	yes
y	exponent	(-∞, ∞)	integer, real	yes

Examples

```
a = pow(4, 2)
```
returns 16

```
a = pow(1+j*1,2+j*2)
```
returns -0.266+j*0.32

Defined in

Built in

See Also

abs(), cint(), exp(), float(), int(), log10(), sgn(), sqrt()

prod()

Returns the product

Syntax

y = prod(x)

Arguments

Name	Description	Range	Type	Required
x	data to find product	(∞, ∞)	integer, real or complex	yes

Examples

```
a = prod([1, 2, 3]
```
returns 6

```
a = prod([4, 4, 4])
```
returns 64

Defined in

Built-in

See Also

sum()

rad()

Converts degrees to radians

Syntax

y = rad(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real	yes

Examples

```
a = rad(90)
```
returns 1.5708

```
a = rad(45)
```
returns 0.785398

Defined in

Built in

See Also

deg()

re()

Returns the real component of a complex number

Syntax

y = re(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	complex	yes

Examples

```
a = re(1-1*j)
```
returns 1.000

Defined in

Built in

See Also

cmplx(), imag(), real()

real()

Returns the real component of a complex number

Syntax

y = real(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	complex	yes

Examples

```
a = real(1-1*j)
```
returns 1.000

Defined in

Built in

See Also

cmplx(), imag(), re()

rms()

Returns the root mean square value

Syntax

y = rms(Value)

Arguments

Name	Description	Range	Type	Required
Value	Value to find RMS	(-∞, ∞)	integer, real, complex	yes

Examples

```
rmsR = rms(2)
```
returns 1.414

```
rmsR = rms(complex(3, 10))
```
returns 7.382/73.301

```
rmsR = rms( [1, 2, 3, 4, 5,] )
```
returns [0.707, 1.414, 2.121, 2.828, 3.536]

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

Notes/Equations

The rms() function calculates the root mean square value. If the data's inner independent is freq, and if frequency equals 0 (DC), then the function returns mag() rather than rms value.

round()

Rounds to the nearest integer

Syntax

y = round(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	real	yes

Examples

```
a = round(0.1)
```
returns 0

```
a = round(0.5)
```
returns 1

```
a = round(0.9)
```
returns 1

```
a = round(–0.1)
```
returns 0

```
a = round(–0.5)
```
returns –1

```
a = round(–0.9)
```
returns –1

Defined in

Built in

See Also

int()

sgn()

Returns the integer sign of an integer or real number, as either 1 or -1

Syntax

y = sgn(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real	yes

Examples

```
a = sgn(-1)
```
returns –1

```
a = sgn(1)
```
returns 1

Defined in

Built in

See Also

abs(), cint(), exp(), float(), int(), log10(), pow(), sqrt()

sin()

Returns the sine

Syntax

y = sin(x)

Arguments

Name	Description	Range	Type	Required
x	number in radians	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = sin(pi/2)
```
returns 1

Defined in

Built in

See Also

cos(), tan()

sinc()

Returns the sinc of a number

Syntax

y = sinc(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = sinc(0.5)
```

returns 0.959

Defined in

Built in

See Also

sin()

Notes/Equations

The sinc function is defined as sinc(x) = sin(pi*x)/ (pi*x) and sinc(0)=1.

sinh()

Returns the hyperbolic sine

Syntax

y = sinh(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = sinh(0)
```
returns 0

```
a = sinh(1)
```
returns 1.1752

Defined in

Built in

See Also

cosh(), tanh()

sqr()

Returns the square of a number

Syntax

y = sqr(x)

Arguments

Name	Description	Range	Type	Required
x	number to square	(-∞, ∞)	integer, real, complex	yes

Examples

```
y = sqr(2)
```
returns 4

Defined In

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

sqrt()

Returns the square root of number

Syntax

y = sqrt(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = sqrt(4)
```
returns 2

```
a = sqrt(2+j*1)
```
returns 1.455+j*0.344

Defined in

Built in

See Also

abs(), cint(), exp(), float(), int(), log10(), pow(), sgn()

step()

Returns 0, 0.5, or 1

Syntax

y= step(x)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = step(-1.5)
```
returns 0.000

```
a = step(0)
```
returns 0.500

```
a = step(1.5)
```
returns 1.000

Defined in

Built in

sum()

Returns the sum

Syntax

y = sum(x)

Arguments

Name	Description	Range	Type	Required
x	data to find sum	(-∞, ∞)	integer, real or complex	yes

Examples

```
a = sum([1, 2, 3])
```
returns 6

Defined in

Built in

See Also

max(), mean(), min()

tan()

Returns the tangent

Syntax

y = tan(x)

Arguments

Name	Description	Range	Type	Required
x	number in radians	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = tan(pi/4)
```
returns 1

```
a = tan(+/-pi/2)
```
returns +/- 1.633E16

Defined in

Built in

See Also

cos(), sin()

tanh()

Returns the hyperbolic tangent

Syntax

y = tanh(x)

Arguments

Name	Description	Range	Type	Required
x	number in radians	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = tanh(0)
```
returns 0

```
a = tanh(1)
```
returns 0.761594

```
a = tanh(–1)
```
returns –0.761594

Defined in

Built in

See Also

cosh(), sinh()

transpose()

Returns the transpose of a matrix

Syntax

y = transpose(Matrix)

Arguments

Name	Description	Range	Type	Required
Matrix	square matrix to find the transpose	(-∞, ∞)	integer, real or complex	yes

Examples

```
a = {{1, 2}, {3, 4}}
b = transpose(a)
```

returns {{1, 3}, {2, 4}}

Defined in

Built in

See Also

diagonal(), inverse()

wtodbm()

Converts Watts to dBm and returns a real or complex number

Syntax

dbmVal = wtodbm(Value)

Arguments

Name	Description	Range	Type	Required
Value	Value in Watts	(-∞, ∞)	real, complex	yes

Examples

```
wtodbm01_M = wtodbm(0.01)
```
returns 10

```
wtodbm1_M = wtodbm(1)
```
returns 30

```
wtodbmC_M = wtodbm(complex(10,2))
```
returns 40.094/1.225

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

See Also

dbmtow()

xor()

Returns an integer that represents the exclusive OR between arguments

Syntax

yXor = xor(x, y)

Arguments

Name	Description	Range	Type	Required
x	number	(-∞, ∞)	integer	yes
y	number	(-∞, ∞)	integer	yes

Examples

```
a = xor(16, 32)
```

returns 48

Defined in

Built in

zeros()

Returns zeros matrix

Syntax

y = ones(rows, columns)

Arguments

Name	Description	Range	Type	Required
rows	number of rows †	[i, ∞)	integer	yes
columns	number of columns †	[i, ∞)	integer	yes
† If only one argument is supplied, then a square matrix is returned. If two are supplied, then a matrix of zeros with size rows X cols is returned				

Examples

```
a = zeros(2)
```
returns {{0, 0}, {0, 0}}

```
b = (2, 3)
```
returns {{0, 0, 0}, {0, 0, 0}}

Defined in

Built in

See Also

identity(), ones()

Chapter 8: Signal Processing Functions

This chapter describes the signal processing functions in detail. The functions are listed in alphabetical order.

A,B,D,E,P,S,T

add_rf()

Returns the sum of two Timed Complex Envelope signals defined by the triplet in-phase (real or I(t)) and quadrature-phase (imaginary or Q(t)) part of a modulated carrier frequency(Fc)

Syntax

y = add_rf(T1, T2)

Arguments

Name	Description	Range	Type	Required
T1	Timed Complex Envelope signals at carrier frequencies Fc1	(-∞, ∞)	complex	yes
T2	Timed Complex Envelope signals at carrier frequencies Fc2	(-∞, ∞)	complex	yes

Examples

```
y = add_rf(T1, T2)
```

Defined in

$HPEESOF_DIR/expressions/ael/signal_proc_fun.ael

Notes/Equations

Used in Signal processing designs that output Timed Signals using Timed Sinks

This equation determines the sum of two Timed Complex Envelope at a new carrier frequency Fc3. Given Fc1 and Fc2 as the carrier frequencies of the two input waveforms, the output carrier frequency Fc3 will be the greater of the two.

ber_pi4dqpsk()

Returns the symbol probability of error versus signal-to-noise ratio per bit for pi/4 DQPSK modulation

Syntax

data = ber_pi4dqpsk(vIn, vOut, symRate, noise, samplingDelay, rotation, tranDelay, pathDelay)

Arguments

Name	Description	Range	Type	Required
vIn	complex envelope voltage signals at the input node	(-∞, ∞)	complex	yes
vOut	complex envelope voltage signals at the output node	(-∞, ∞)	complex	yes
symRate	symbol rate (real) of the modulation signal	(0, ∞)	real	yes
noise	RMS noise vector	(0, ∞)	real	yes
samplingDelay	clock phase in seconds	[0, ∞)	real	yes
rotation	carrier phase in radians	[0, ∞)	real	yes
tranDelay	time in seconds that causes this time duration of symbols to be eliminated from the bit error rate calculation †	[0, ∞)	real	yes
pathDelay	delay from input to output in seconds	[0, ∞)	real	yes
† Usually the filters in the simulation have transient responses, and the bit error rate calculation should not start until these transient responses have finished				

Examples

```
y = ber_pi4dqpsk(videal[1], vout[1], 0.5e6, {0.1::-0.01::0.02})
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

ber_qpsk(), constellation()

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.

The arguments *vIn* and *vOut* usually come from a circuit envelope simulation, while *noise* usually comes from a harmonic balance simulation, and is assumed to be additive white Gaussian. It can take a scalar or vector value. The function uses the quasi-analytic approach for estimating BER: for each symbol, Eb / N0 and BER are calculated analytically; then the overall BER is the average of the BER values for the symbols.

Note that ber_pi4dqpsk returns a list of data:

```
data[0]= symbol probability of error versus Eb / N0
data[1]= path delay in seconds
data[2]= carrier phase in radians
data[3]= clock phase in seconds
data[4]= complex(Isample, Qsample)
```

ber_qpsk()

Returns the symbol probability of error versus signal-to-noise ratio per bit for QPSK modulation

Syntax

data = ber_qpsk(vIn, vOut, symRate, noise{, samplingDelay, rotation, tranDelay, pathDelay})

Arguments

Name	Description	Range	Type	Required
vIn	complex envelope voltage signals at the input node	(-∞, ∞)	complex	yes
vOut	complex envelope voltage signals at the output node	(-∞, ∞)	complex	yes
symRate	symbol rate (real) of the modulation signal	(0, ∞)	real	yes
noise	RMS noise vector	(0, ∞)	real	yes
samplingDelay	clock phase in seconds	[0, ∞)	real	yes
rotation	carrier phase in radians	[0, ∞)	real	yes
tranDelay	time in seconds that causes this time duration of symbols to be eliminated from the bit error rate calculation †	[0, ∞)	real	yes
pathDelay	delay from input to output in seconds	[0, ∞)	real	yes
† Usually the filters in the simulation have transient responses, and the bit error rate calculation should not start until these transient responses have finished				

Examples

```
y = ber_qpsk(videal[1], vout[1], 1e6, {0.15::-0.01::0.04})
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

ber_pi4dqpsk(), constellation()

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.

The arguments *vIn* and *vOut* usually come from a circuit envelope simulation, while *noise* usually comes from a harmonic balance simulation, and is assumed to be additive white Gaussian. It can take a scalar or vector value. The function uses the quasi-analytic approach for estimating BER: for each symbol, Eb / N0 and BER are calculated analytically; then the overall BER is the average of the BER values for the symbols.

Note that ber_qpsk returns a list of data:

```
data[0]= symbol probability of error versus Eb / N0
data[1]= path delay in seconds
data[2]= carrier phase in radians
data[3]= clock phase in seconds
data[4]= complex(Isample, Qsample)
```

eye()

Creates data for an eye diagram plot

Syntax

y = eye(data, symbolRate, Cycles, Delay)

Arguments

Name	Description	Range	Type	Default	Required
data	either numeric data or a time domain waveform typically from the I or Q data channel	(-∞, ∞)	complex		yes
symbolRate	symbol rate of the channel. For numeric data, the symbol rate is the reciprocal of the number of points in one cycle; for a waveform, it is the frequency	(0, ∞)	real		yes
Cycles	number of cycles to repeat	[1, ∞)	integer	1	no
Delay	sampling delay	[0, ∞)	integer, real	0	no

Examples

```
y = eye(I_data, symbol_rate)
```

Defined in

Built in

See Also

constellation(), eye_amplitude(), eye_closure(), eye_fall_time(), eye_height(), eye_rise_time()

eye_amplitude()

Returns eye amplitude

Syntax

y = eye_amplitude(Vout_time, Delay, BitRate, SamplePoint, WindowPct)

Arguments

Name	Description	Range	Type	Required
Vout_time	time domain voltage waveform	(-∞, ∞)	complex	yes
Delay	used to remove initial transient in the eye diagram and is expressed in time units	[0, ∞)	real	yes
BitRate	bit rate of the channel and is expressed in frequency units	[0, ∞)	real	yes
SamplePoint	marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.		string	yes
WindowPct	used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.	[0, 1]	real	yes

Examples

```
Eye_amp = eye_amplitude(vout,12 ps,10 GHz, m1, 0.2)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

cross_hist(), eye(), eye_closure(), eye_fall_time(), eye_height(), eye_rise_time()

Notes/Equations

The eye_amplitude() function essentially takes the vertical histogram of the eye voltages and subtracts the “0” level mean from “1” level mean within a given measurement window.

eye_closure()

Returns eye closure

Syntax

y = eye_closure(Vout_time, Delay, BitRate, SamplePoint, WindowPct)

Arguments

Name	Description	Range	Type	Required
Vout_time	time domain voltage waveform	(-∞, ∞)	complex	yes
Delay	used to remove initial transient in the eye diagram and is expressed in time units	[0, ∞)	real	yes
BitRate	bit rate of the channel and is expressed in frequency units	[0, ∞)	real	yes
SamplePoint	marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.		string	yes
WindowPct	used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.	[0, 1]	real	yes

Examples

```
Eye_Close = eye_closure(vout,12 ps,10 GHz, m1, 0.2)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

cross_hist(), eye(), eye_amplitude(), eye_fall_time(), eye_height(), eye_rise_time()

Notes/Equations

Computes the ratio of eye height to eye amplitude to provide eye closure.

eye_fall_time()

Returns eye fall time

Syntax

y = eye_fall_time(Vout_time,Delay,BitRate,SamplePoint,WindowPct)

Arguments

Name	Description	Range	Type	Required
Vout_time	time domain voltage waveform	(-∞, ∞)	complex	yes
Delay	used to remove initial transient in the eye diagram and is expressed in time units	[0, ∞)	real	yes
BitRate	bit rate of the channel and is expressed in frequency units	[0, ∞)	real	yes
SamplePoint	marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.		string	yes
WindowPct	used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.	[0, 1]	real	yes

Examples

```
Eye_Fall = eye_fall_time(vout,12 ps,10 GHz, m1, 0.2)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

cross_hist(), eye(), eye_amplitude(), eye_closure(), eye_height(), eye_rise_time()

Notes/Equations

Computes 20% – 80% fall time of a time domain waveform.

eye_height()

Returns eye height

Syntax

y = eye_height(Vout_time, Delay, BitRate, SamplePoint, WindowPct)

Arguments

Name	Description	Range	Type	Required
Vout_time	time domain voltage waveform	(-∞, ∞)	complex	yes
Delay	used to remove initial transient in the eye diagram and is expressed in time units	[0, ∞)	real	yes
BitRate	bit rate of the channel and is expressed in frequency units	[0, ∞)	real	yes
SamplePoint	marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.		string	yes
WindowPct	used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.	[0, 1]	real	yes

Examples

```
Eye_Ht = eye_height(vout,12 ps,10 GHz, m1, 0.2)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

cross_hist(), eye(), eye_amplitude(), eye_closure(), eye_fall_time(), eye_rise_time()

Notes/Equations

The eye_height() function essentially takes the vertical histogram of the eye voltages and computes inner bounds of the eye opening.

eye_rise_time()

Returns eye rise time

Syntax

y = eye_rise_time(Vout_time,Delay,BitRate,SamplePoint,WindowPct)

Arguments

Name	Description	Range	Type	Required
Vout_time	time domain voltage waveform	(-∞, ∞)	complex	yes
Delay	used to remove initial transient in the eye diagram and is expressed in time units	[0, ∞)	real	yes
BitRate	bit rate of the channel and is expressed in frequency units	[0, ∞)	real	yes
SamplePoint	marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.		string	yes
WindowPct	used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.	[0, 1]	real	yes

Examples

```
Eye_Rise = eye_rise_time(vout,12 ps,10 GHz, m1, 0.2)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

cross_hist(), eye(), eye_amplitude(), eye_closure(), eye_fall_time(), eye_height()

Notes/Equations

Computes 20% – 80% rise time of a time domain waveform.

spec_power()

Returns the integrated signal power (dBm) of a spectrum

Syntax

y = spec_power(spectralData{, lowerFrequencyLimit, upperFrequencyLimit})

Arguments

Name	Description	Range	Type	Default	Required
spectralData	spectral data in dBm		real		yes
lowerFrequency Limit	lower frequency limit to be used in calculating the integrated power	(-∞, ∞)	real	min(indep(spectralData))	no
upperFrequency Limit	upper frequency limit to be used in calculating the integrated power	[lowerFrequencyLimit, ∞)	real	max(indep(spectralData))	no

Examples

```
total_power = spec_power(dBm(Mod_Spectrum), 60 MHz, 71 MHz)
```
where Mod_Spectrum is the instance name of a SpectrumAnalyzer sink component, will return the integrated power between 60 and 71 MHz.

```
total_power = spec_power(dBm(fs(Vout[1])), indep(m1), indep(m2))
```
where Vout is a named node in a Circuit Envelope simulation, will return the integrated power between markers 1 and 2.

Defined in

$HPEESOF_DIR/expressions/ael/signal_proc_fun.ael

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.

This expression can be used with spectral data of up to 4 dimensions (frequency should be the inner dimension).

The spec_power() function returns the power (in dBm) of a spectrum integrated between the lower and upper frequency limits specified. If no lower (upper) limit is specified, the lowest (highest) frequency in the spectral data is used instead.

The input spectral data must be in dBm. Spectral data can be generated in several different ways, such as applying the fs() expression on voltage or current time domain

data or using the SpectrumAnalyzer sink component. The obsolete components SpecAnalyzer and FFTAnalyzer also generate spectral data.

The fs() expression returns the voltage or current spectrum of the input data and so the dBm() expression should be applied to the fs() output before it is passed to the spec_power() expression. The frequency axis values in the spectral data returned by fs() are in Hz and so if lower and/or upper frequency limits are to be passed to spec_power(), they should be specified in Hz.

Similarly, the SpectrumAnalyzer sink component returns the voltage spectrum of the input signal and so the dBm() expression should be applied to the spectral data generated by SpectrumAnalyzer before it is passed to the spec_power() expression. The frequency axis values in the spectral data generated by SpectrumAnalyzer are in Hz, so if lower and/or upper frequency limits are to be passed to spec_power(), they should be specified in Hz.

On the contrary, the obsolete SpecAnalyzer and FFTAnalyzer sink components can return the spectrum of the input signal in dBm, dBV, or Magnitude (based on the value of their Display parameter). If dBm is selected, then the spectral data generated by these two sinks can be directly passed to spec_power(). Otherwise, the appropriate transformation from dBV to dBm or Magnitude to dBm should be applied first. The frequency axis values in the spectral data generated by SpecAnalyzer and FFTAnalyzer are in the unit specified by their DisplayFreqUnit parameter, so if lower and/or upper frequency limits are to be passed to spec_power(), they should be specified in the same unit.

Chapter 9: S-parameter Analysis Functions

This chapter describes the S-parameter analysis functions in detail. The functions are listed in alphabetical order.

T,U,V,W,Y

Z

abcdtoh()

This measurement transforms the chain (ABCD) matrix of a 2-port network to a hybrid matrix

Syntax

h=abcdtoh(a)

Arguments

Name	Description	Range	Type	Required
A	chain (ABCD) matrix of a 2-port network	(-∞, ∞)	complex	yes

Examples

```
h = abcdtoh(a)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtos(), abcdtoy(), abcdtoz(), htoabcd(), stoabcd(), ytoabcd(), ztoabcd()

abcdtos()

His measurement transforms the chain (ABCD) matrix of a 2-port network to a scattering matrix

Syntax

sp = abcdtos(A, zRef)

Arguments

Name	Description	Range	Type	Default	Required
A	chain (ABCD) matrix of a 2-port network	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	integer, real or complex	50.0	no

Examples

```
sp = abcdtos(a, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh(), abcdtoy(), abcdtoz(), htoabcd(), stoabcd(), ytoabcd(), ztoabcd()

abcdtoy()

This measurement transforms the chain (ABCD) matrix of a 2-port network to an admittance matrix

Syntax

y = abcdtoy(a)

Arguments

Name	Description	Range	Type	Required
A	chain (ABCD) matrix of a 2-port network	(-∞, ∞)	complex	yes

Examples

```
y = abcdtoy(a)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh(), abcdtos(), abcdtoz(), htoabcd(), stoabcd(), ytoabcd(), ztoabcd()

abcdtoz()

This measurement transforms the chain (ABCD) matrix of a 2-port network to impedance matrix

Syntax

z = abcdtoz(a)

Arguments

Name	Description	Range	Type	Required
A	chain (ABCD) matrix of a 2-port network	(-∞, ∞)	complex	yes

Examples

```
z = abcdtoz(a)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh(), abcdtos(), abcdtoy(), htoabcd(), stoabcd(), ytoabcd(), ztoabcd()

bandwidth_func()

Returns the bandwidth at the specified level as a real number. Typically used in filter application to calculate the 1, 3 dB bandwidth

Syntax

bw = bandwidth_func(Data, DesiredValue, Type)

Arguments

Name	Description	Range	Type	Default	Required
Data	data (usually gain) to find the bandwidth	(-∞, ∞)	integer, real		yes
DesiredValue	A single value representing the desired bandwidth level.	(-∞, ∞)	real		yes
Type	Type of response	[0, 3] †	integer	0	no
† Type: 0 - Band-pass 1 - Band-stop 2 - Low-pass 3 - High-pass					

Examples

This example assumes that an S-parameter analysis has been performed.

```
bw3dB = bandwidth_func(db(S21), 3)
```

returns the 3dB bandwidth

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

center_freq()

Notes/Equations

This function returns the bandwidth of a filter response. Bandwidth is defined as the difference between the upper and lower frequency at which the amplitude response is *DesiredValue* dB below the maximum amplitude. It uses an iterative process to find the bandwidth on non-ideal responses for the band limited responses.

Data can be from 1 to 4 dimensions.

center_freq()

Returns the center frequency at the specified level as a real number.

Syntax

fc = center_freq(Data, ReferenceBW)

Arguments

Name	Description	Range	Type	Required
Data	data (usually gain) to find the center frequency	(-∞, ∞)	complex	yes
ReferenceBW	a single value representing the reference bandwidth	(-∞, ∞)	real	yes

Examples

This example assumes that an S-parameter analysis has been performed.

```
fc3 = center_freq(db(S21), 3)
```

returns the center frequency using the 3dB point as reference.

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

bandwidth_func()

Notes/Equations

This function returns the center frequency of a filter response. Linear interpolation is performed between data points. The function uses an iterative process to find the bandwidth on non-ideal responses. The data can be from 1 to 4 dimensions.

dev_lin_gain()

Given a variable sweep over a frequency range, a linear least-squares fit is performed on the gain of the variable, and the deviation from this linear fit is calculated at each frequency point

Syntax

y = dev_lin_gain(voltGain)

Arguments

Name	Description	Range	Type	Required
voltGain	gain as a function of frequency	[0, ∞)	complex	yes

Examples

```
a = dev_lin_gain(volt_gain(S,PortZ(1),PortZ(2)))
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

dev_lin_phase(), diff(), phasedeg(), phaserad(), pwr_gain(), ripple(), unwrap(), volt_gain()

dev_lin_phase()

Given a variable sweep over a frequency range, a linear least-squares fit is performed on the phase of the variable, and the deviation from this linear fit is calculated at each frequency point

Syntax

y = dev_lin_phase(voltGain)

Arguments

Name	Description	Range	Type	Required
voltGain	function of frequency	[0, ∞)	complex	yes

Examples

```
a = dev_lin_phase(S21)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

dev_lin_gain(), diff(), phasedeg(), phaserad(), pwr_gain(), ripple(), unwrap(), volt_gain()

Notes/Equations

In order to use this function, the *Group Delay* option must be enabled in the S-parameter analysis setup. For more information, refer to "*Calculating Group Delay*" in your "*S-Parameter Simulation*" documentation.

ga_circle()

Generates an available gain circle

Syntax

y = ga_circle(S, gain, numOfPts, numCircles, gainStep)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network.	(-∞, ∞)	complex		yes
gain	specified gain in dB	[0, ∞)	integer or real array	†	no
numOfPts	desired number of points per circle	[1, ∞)	integer	51	no
numCircles	number of desired circles. This is used if gain is not specified.	[0, ∞)	integer		no
gainStep	gain step size. This is used if gain is not specified.	[0, ∞)	integer or real	1.0	no
† Default value for gain is min(max_gain(S)) - {1, 2, 3}					

Examples

```
circleData = ga_circle(S, 2, 51)
```

```
circleData = ga_circle(S, {2, 3, 4}, 51)
```
return the points on the circle(s).

```
circleData = ga_circle(S, , 51, 5, 0.5)
```
return the points on the circle(s) for 5 circles at maxGain - {0,0.5,1.0,1.5,2.0}

```
circleData = ga_circle(S, , , 2, 1.0)
```
return the points on the circle(s) for 2 circles at maxGain - {0,1.0}

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

gl_circle(), gp_circle(), gs_circle()

Notes/Equations

This function is used in Small-signal S-parameter simulations.

The function generates the constant available-gain circle resulting from a source mismatch. The circle is defined by the loci of the source-reflection coefficients resulting in the specified gain.

A gain circle is created for each value of the swept variable(s). Multiple gain values can be specified for a scattering parameter that has dimension less than four. This measurement is supported for 2-port networks only.

If gain and numCircles are not specified, gain circles are drawn at min(max_gain(S)) - {0,1,2,3}. That is, gain is calculated at a loss of 0,1,2,3 dB from maxGain.

If gain is not specified and numCircles is given, then numCircles gain circles are drawn at gainStep below max_gain(). Gain is also limited by max_gain(S). That is, if gain > max_gain(S), then the circle is generated at max_gain(S).

$$Ca(Center) = \frac{ga C_1^*}{(1 + ga(|S_{11}|^2 - |\Delta|^2))}$$

$$Ra(radius) = \frac{\sqrt{1 - 2K|S_{12}S_{21}|ga + |S_{12}S_{21}|^2 ga^2}}{|1 + ga(|S_{11}|^2 - |\Delta|^2)|}$$

Where:

K = Stability Factor

$ga(G) = (Ga / |S_{21}|^2)$ (Ga is Desired Gain in absolute terms)

$C_1 = S_{11} - |\Delta|^2 S_{22}^*$

gain_comp()

Returns gain compression

Syntax

y = gain_comp(Sji)

Arguments

Name	Description	Range	Type	Required
Sji	Sji is a power-dependent complex transmission coefficient obtained from large-signal S-parameter simulation.	(-∞, ∞)	complex	yes

Examples

```
gc = gain_comp(S21[::, 0])
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

phase_comp()

Notes/Equations

Used in Large-signal S-parameter simulations.

This measurement calculates the small-signal minus the large-signal power gain, in dB. The first power point (assumed to be small) is used to calculate the small-signal power gain.

gl_circle()

Returns a load-mismatch gain circle

Syntax

y = gl_circle(S, gain, numOfPts, numCircles, gainStep)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network.	(-∞, ∞)	complex		yes
gain	specified gain in dB	[0, ∞)	integer or real array	maxGain - {0, 1, 2, 3} †	no
numOfPts	desired number of points per circle	[1, ∞)	integer	51	no
numCircles	number of desired circles. This is used if gain is not specified.	[0, ∞)	integer		no
gainStep	gain step size. This is used if gain is not specified.	[0, ∞)	integer or real	1.0	no
† Where maxGain = 10*log(1 / (1 - mag(S22)**2))					

Examples

```
circleData = gl_circle(S, 2, 51)
```

```
circleData = gl_circle(S, {2, 3, 4}, 51)
```
return the points on the circle(s).

```
circleData = gl_circle(S, , 51, 5, 0.5)
```
return the points on the circle(s) for 5 circles at maxGain - {0,0.5,1.0,1.5,2.0}

```
circleData = gl_circle(S, , , 2, 1.0)
```
return the points on the circle(s) for 2 circles at maxGain - {0,1.0}

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

ga_circle(), gp_circle(), gs_circle()

Notes/Equations

Used in Small-signal S-parameter simulations.

This function generates the unilateral gain circle resulting from a load mismatch. The circle is defined by the loci of the load-reflection coefficients that result in the specified gain.

A gain circle is created for each value of the swept variable(s). Multiple gain values can be specified for a scattering parameter that has dimension less than four. This measurement is supported for 2-port networks only.

If gain and numCircles are not specified, gain circles are drawn at maxGain - {0,1,2,3}. That is, gain is calculated at a loss of 0,1,2,3 dB from the maximum gain. If gain is not specified and numCircles is given, then numCircles gain circles are drawn at gainStep below maxGain. Gain is also limited by maxGain. That is, if gain > maxGain, then the circle is generated at maxGain.

$$Cl(Center) = \frac{(1-|S_{22}|^2 Gabs)|S_{22}|^*}{1-|S_{22}|^2(1-|S_{22}|^2 Gabs)}$$

$$Rl(radius) = \frac{\sqrt{|1-(1-|S_{22}|^2 Gabs)|}(1-|S_{22}|^2)}{1-|S_{22}|^2(1-|S_{22}|^2 Gabs)}$$

Where *Gabs* is the absolute gain, and is given by:

$$Gabs = (10^{dBGain})/10.0$$

gp_circle()

Generates a power gain circle

Syntax

y = gp_circle(S, gain, numOfPts, numCircles, gainStep)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network.	(-∞, ∞)	complex		yes
gain	specified gain in dB	[0, ∞)	integer or real array	†	no
numOfPts	desired number of points per circle	[1, ∞)	integer	51	no
numCircles	number of desired circles. This is used if gain is not specified.	[0, ∞)	integer		no
gainStep	gain step size. This is used if gain is not specified.	[0, ∞)	integer or real	1.0	no
† Default value for gain is min(max_gain(S)) - {1, 2, 3}					

Examples

```
circleData = gp_circle(S, 2, 51)
```

```
circleData = gp_circle(S, {2, 3, 4}, 51)
```
return the points on the circle(s)

```
circleData = gp_circle(S, , 51, 5, 0.5)
```
return the points on the circle(s) for 5 circles at maxGain - {0,0.5,1.0,1.5,2.0}

```
circleData = gp_circle(S, , , 2, 1.0)
```
return the points on the circle(s) for 2 circles at maxGain - {0,1.0}

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

ga_circle(), gl_circle(), gs_circle()

Notes/Equations

Used in Small-signal S-parameter simulations.

This function generates a constant-power-gain circle resulting from a load mismatch. The circle is defined by the loci of the output-reflection coefficients that result in the specified gain.

A gain circle is created for each value of the swept variable(s). Multiple gain values can be specified for a scattering parameter that has dimension less than four. This measurement is supported for 2-port networks only.

If gain and numCircles are not specified, gain circles are drawn at min(max_gain(S)) - {0,1,2,3}. That is, gains are calculated at a loss of 0,1,2,3 dB from the maximum gain. If gain is not specified and numCircles is given, then numCircles gain circles are drawn at gainStep below max_gain(). Gain is also limited by max_gain(S). That is, if gain > max_gain(S), then the circle is generated at max_gain(S)).

$$Cp(Center) = \frac{gpC_2^*}{(1 + gp(|S_{22}|^2 - |\Delta|^2))}$$

$$Rp(radius) = \frac{\sqrt{1 - 2K|S_{12}S_{21}|gp + |S_{12}S_{21}|^2 gp^2}}{|1 + gp(|S_{22}|^2 - |\Delta|^2)|}$$

Where:

K = Stability Factor

$gp(G) = (Gp / |S_{21}|^2)$ (Gp is Desired Gain in absolute terms)

$C_2 = S_{22} - |\Delta|^2 S_{11}^*$

gs_circle()

Returns a source-mismatch gain circle

Syntax

y = gs_circle(S, gain, numOfPts, numCircles, gainStep)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network.	(-∞, ∞)	complex		yes
gain	specified gain in dB	[0, ∞)	integer or real array	maxGain - {0, 1, 2, 3} †	no
numOfPts	desired number of points per circle	[1, ∞)	integer	51	no
numCircles	number of desired circles. This is used if gain is not specified.	[0, ∞)	integer		no
gainStep	gain step size. This is used if gain is not specified.	[0, ∞)	integer or real	1.0	no
† Where maxGain = 10*log(1 / (1 - mag(S11)**2))					

Examples

```
circleData = gs_circle(S, 2, 51)
```

```
circleData = gs_circle(S, {2, 3, 4}, 51)
```
return the points on the circle(s)

```
circleData = gs_circle(S, , 51, 5, 0.5)
```
return the points on the circle(s) for 5 circles at maxGain - {0,0.5,1.0,1.5,2.0}

```
circleData = gs_circle(S, , , 2, 1.0)
```
return the points on the circle(s) for 2 circles at maxGain - {0,1.0}

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

ga_circle(), gl_circle(), gp_circle()

Notes/Equations

Used in Small-signal S-parameter simulations.

This function generates the unilateral gain circle resulting from a source mismatch. The circle is defined by the loci of the source-reflection coefficients that result in the specified gain. A gain circle is created for each value of the swept variable(s). Multiple gain values can be specified for a scattering parameter that has dimension less than four. This measurement is supported for 2-port networks only.

If gain and numCircles are not specified, gain circles are drawn at maxGain - {0,1,2,3}. That is, gain values are calculated at a loss of 0,1,2,3 dB from the maximum gain. If gain is not specified and if numCircles is given, then numCircles gain circles are drawn at gainStep below maxGain. Gain is also limited by maxGain. That is, if gain > maxGain, then the circle is generated at maxGain.

$$Cs(Center) = \frac{(1-|S_{11}|^2 Gabs)|S_{11}|^*}{1-|S_{11}|^2(1-|S_{11}|^2 Gabs)}$$

$$Rs(radius) = \frac{\sqrt{|1-(1-|S_{11}|^2 Gabs)|}(1-|S_{11}|^2)}{1-|S_{11}|^2(1-|S_{11}|^2 Gabs)}$$

Where *Gabs* is the absolute gain, and is given by:

$$Gabs = (10^{dBGain})/10.0$$

htoabcd()

This measurement transforms the hybrid matrix of a 2-port network to a chain (ABCD) matrix

Syntax

a = htoabcd(H)

Arguments

Name	Description	Range	Type	Required
H	hybrid matrix of a 2-port network	(-∞, ∞)	complex	yes

Examples

```
a = htoabcd(H)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh(), htoz(), ytoh()

htos()

This measurement transforms the hybrid matrix of a 2-port network to a scattering matrix

Syntax

sp = htos(h, zRef)

Arguments

Name	Description	Range	Type	Default	Required
H	hybrid matrix of a 2-port network	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	integer, real or complex	50.0	no

Examples

```
s = htos(h, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htoy(), htoz(), stoh()

htoy()

This measurement transforms the hybrid matrix of a 2-port network to an admittance matrix

Syntax

y = htoy(H)

Arguments

Name	Description	Range	Type	Required
H	hybrid matrix of a 2-port network	(-∞, ∞)	complex	yes

Examples

```
y = htoy(H)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htos(), htoz(), ytoh()

htoz()

This measurement transforms the hybrid matrix of a 2-port network to an impedance matrix

Syntax

z = htoz(H)

Arguments

Name	Description	Range	Type	Required
H	hybrid matrix of a 2-port network	(-∞, ∞)	complex	yes

Examples

```
z = htoz(H)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htos(), htoy(), ytoh()

ispec()

Returns the current frequency spectrum

Syntax

y = ispec(current)

Arguments

Name	Description	Range	Type	Required
current	current	(-∞, ∞)	real, complex	yes

Examples

```
a = ispec(i1)
```

Defined In

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pspec(), vspec()

Notes/Equations

This measurement gives a current frequency spectrum. The measurement gives a set of RMS currents at each frequency.

l_stab_circle()

Returns load (output) stability circles

Syntax

y = l_stab_circle(S, numOfPts)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network.	(-∞, ∞)	complex		yes
numOfPts	desired number of points per circle	[1, ∞)	integer	51	no

Examples

```
circleData = l_stab_circle(S, 51)
```

returns the points on the circle(s)

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle_center_radius(), l_stab_region(), s_stab_circle(), s_stab_region()

Notes/Equations

Used in Small-signal S-parameter simulations.

The function generates a load stability circle. The circle is defined by the loci of load-reflection coefficients where the magnitude of the source-reflection coefficient is 1.

A circle is created for each value of the swept variable(s). This measurement is supported for 2-port networks only.

Use the function *l_stab_circle_center_radius()* to find the center and radius of the stability circle.

Use the function *l_stab_region(S)* to determine the region of stability.

l_stab_circle_center_radius()

Returns the center and radius of the load (output) stability circle

Syntax

l_cr = l_stab_circle_center_radius(S, Type)

Arguments

Name	Description	Range	Type	Default	Required
S	2-port S-Parameters		complex		yes
Type	Type of parameter to return	["both"|"center"|"radius"]	string	"both"	no

Examples

```
l_cr = l_stab_circle_center_radius(S)
```
returns a 1X2 matrix containing center and radius

```
lCirc = expand(circle(l_cr(1), real(l_cr(2)), 51))
```
returns data for the load stability circle

```
l_radius = l_stab_circle_center_radius(S, "radius")
```
returns the radius

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle(), s_stab_circle(), s_stab_circle_center_radius()

Notes/Equations

Used in Small-signal S-parameter simulations.

If the argument *Type* is not specified, the function returns complex data. Although radius is of type real, the values are returned as complex. Therefore, when using the returned radius, use the real part. To obtain the radius as a real number, set the *Type* argument to *radius*.

l_stab_region()

This expression returns a string identifying the region of stability of the corresponding load stability circle

Syntax

y = l_stab_region(S)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of a 2-port network.	(-∞, ∞)	complex	yes

Examples

```
region = l_stab_region(S)
```

returns “Outside” or “Inside”

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle(), s_stab_circle(), s_stab_region()

Notes/Equations

Used in Small-signal S-parameter simulations.

map1_circle()

Returns source-mapping circles from port 1 to port 2

Syntax

circleData=map1_circle(S, numOfPts)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network.	(-∞, ∞)	complex		yes
numOfPts	desired number of points per circle	[1, ∞)	integer	51	no

Examples

```
circleData = map1_circle(S, 51)
```
returns the points on the circle(s)

Defined in

$HPEESOF_DIR/expressions/ael/circles_fun.ael

See Also

map2_circle()

Notes/Equations

Used in Small-signal S-parameter simulations

The expression maps the set of terminations with unity magnitude at port 1 to port 2. The circles are defined by the loci of terminations on one port as seen at the other port. A source-mapping circle is created for each value of the swept variable(s). This measurement is supported for 2-port networks only.

map2_circle()

Returns load-mapping circles from port 2 to port 1

Syntax

circleData=map2_circle(S, numOfPts)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network.	(-∞, ∞)	complex		yes
numOfPts	desired number of points per circle	[1, ∞)	integer	51	no

Examples

```
circleData = map2_circle(S, 51)
```
returns the points on the circle(s)

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

map1_circle()

Notes/Equations

Used in Small-signal S-parameter simulations

The function maps the set of terminations with unity magnitude at port 2 to port 1. The circles are defined by the loci of terminations on one port as seen at the other port. A source-mapping circle is created for each value of the swept variable(s). This measurement is supported for 2-port networks only.

max_gain()

Given a 2 x 2 scattering matrix, this measurement returns the maximum available and stable gain (in dB) between the input and the measurement ports

Syntax

y = max_gain(S)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of 2-port network	(-∞, ∞)	complex	yes

Examples

```
y = max_gain(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

sm_gamma1(), sm_gamma2(), stab_fact(), stab_meas()

mu()

Returns the geometrically derived stability factor for the load

Syntax

y = mu(S)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of a 2-port network	(-∞, ∞)	complex	yes

Examples

```
x = mu(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

mu_prime()

Notes/Equations

This measurement gives the distance from the center of the Smith chart to the nearest output (load) stability circle. This stability factor is given by:

$$mu = \{1-|S_{11}|^2\} / \{|S_{22} - conj(S_{11})*Delta| + |S_{12}*S_{21}|\}$$

where Delta is the determinant of the S-parameter matrix. Having mu > 1 is the single necessary and sufficient condition for unconditional stability of the 2-port network.

Reference

[1] M. L. Edwards and J. H. Sinsky, “A new criterion for linear 2-port stability using geometrically derived parameters", IEEE Transactions on Microwave Theory and Techniques, Vol. 40, No. 12, pp. 2303-2311, Dec. 1992.

mu_prime()

Returns the geometrically derived stability factor for the source

Syntax

y = mu_prime(S)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of a 2-port network	(-∞, ∞)	complex	yes

Examples

```
a = mu_prime(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

mu()

Notes/Equations

This measurement gives the distance from the center of the Smith chart to the nearest unstable-input (source) stability circle. This stability factor is given by:

mu_prime = $\{1-|S_{22}|^2\} / \{|S_{11} - \text{conj}(S_{22})*\text{Delta}| + |S_{21}*S_{12}|\}$

where Delta is the determinant of the S-parameter matrix. Having mu_prime > 1 is the single necessary and sufficient condition for unconditional stability of the 2-port network.

Reference

M. L. Edwards and J. H. Sinsky, “A new criterion for linear 2-port stability using geometrically derived parameters”, IEEE Transactions on Microwave Theory and Techniques, Vol. 40, No. 12, pp. 2303-2311, Dec. 1992.

ns_circle()

Returns noise-figure circles

Syntax

y = ns_circle(nf2, NFmin, Sopt, rn, numOfPts, numCircles, NFStep)

Arguments

Name	Description	Range	Type	Default	Required
nf2	specified noise figure	(-∞, ∞)	real	†	no
NFmin	minimum noise figure	[0, ∞)	integer or real		yes
Sopt	optimum mismatch	[0, ∞)	complex		yes
rn	equivalent normalized noise resistance of a 2-port network ‡	[0, ∞)	complex		yes
numOfPts	desired number of points per circle	[1, ∞)	integer	51	no
numCircles	number of desired circles. This is used if nf2 is not specified.	[0, ∞)	integer		no
NFStep	nf step size. This is used if nf2 is not specified.	[0, ∞)	integer or real	1.0	no

† If nf2 is NULL or not specified the default is max(NFmin)+{0,1,2,3}.
‡ rn = Rn/zRef where Rn is the equivalent noise resistance and zRef is the reference impedance.

Examples

```
circleData = ns_circle(0+NFmin, NFmin, Sopt, Rn/50, 51)
```

```
circleData = ns_circle(NULL, NFmin, Sopt, Rn/50, 51)
```
return the points on the circle for 4 circles at max(NFmin)+(0,1,2,3)

```
circleData = ns_circle({0, 1}+NFmin, NFmin, Sopt, Rn/50, 51)
```
returns the points on the circle(s)

```
circleData = ns_circle(, NFmin, Sopt, Rn/50, 51, 3, 0.5)
```
returns the points on the circle(s) for 3 circles at max(NFmin) + {0, 0.5, 1.0}

```
circleData = ns_circle(, NFmin, Sopt, Rn/50, , 3)
```
returns the points on the circle(s) for 3 circles at max(NFmin) + {0, 1, 2.0}

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

Notes/Equations

Used in Small-signal S-parameter simulations and Harmonic Balance analysis.

The expression generates constant noise-figure circles. The circles are defined by the loci of the source-reflection coefficients that result in the specified noise figure. NFmin, Sopt, and Rn are generated from noise analysis.

A circle is created for each value of the swept variable(s).

If both nf2 and numCircles are specified, then circles are drawn at nf2 values (numCircles is not used).

If nf2 and numCircles are not specified, then nf2 circles are drawn at max(NFmin) + {0,1,2,3}.

If nf2 is not specified, and numCircles is given, then numCircles nf2 circles are drawn at NFStep above max(NFmin).

If nf2 is specified, and numCircles is not specified, then circles are drawn at nf2 values.

ns_pwr_int()

Returns the integrated noise power

Syntax

y = ns_pwr_int(Sji, nf, resBW, stop)

Arguments

Name	Description	Range	Type	Required
Sji	complex transmission coefficient	(-∞, ∞)	complex	yes
nf	noise figure at the output port (in dB)	(-∞, ∞)	complex	yes
resBW	user-defined resolution bandwidth	(-∞, ∞)	complex	yes
stop	stop value (works for nonuniform delta frequency)	[0, ∞)	real	no

Examples

```
Y = ns_pwr_int(S21, nf2, 1MHz)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ns_pwr_ref_bw(), snr()

Notes/Equations

Used in Small-signal S-parameter simulation

This is the integrated noise power (in dBm) calculated by integrating the noise power over the entire frequency sweep. The noise power at each frequency point is calculated by multiplying the noise spectral density by a user-defined resolution bandwidth.

ns_pwr_ref_bw()

Returns noise power in a reference bandwidth

Syntax

y = ns_pwr_ref_bw(Sji, nf, resBW)

Arguments

Name	Description	Range	Type	Required
Sji	complex transmission coefficient	(-∞, ∞)	complex	yes
nf	noise figure at the output port (in dB)	(-∞, ∞)	complex	yes
resBW	user-defined resolution bandwidth	(-∞, ∞)	complex	yes

Examples

```
Y = ns_pwr_ref_bw(S21, nf2, 1MHz)
```

returns the noise power with respect to the reference bandwidth

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ns_pwr_int(), snr()

Notes/Equations

Used in Small-signal S-parameter simulation.

This is the noise power calculated by multiplying the noise spectral density at a frequency point by a user-defined resolution bandwidth. Unlike NsPwrInt, this gives the noise power (in dB) at each frequency sweep.

phase_comp()

Returns the phase compression (phase change)

Syntax

y = phase_comp(Sji)

Arguments

Name	Description	Range	Type	Required
Sji	power-dependent parameter obtained from large-signal S-parameters simulation	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = phase_comp(S21[::, 0])
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_systems_fun.ael

See Also

gain_comp()

Notes/Equations

Used in Large-signal S-parameter simulations

This measurement calculates the small-signal minus the large-signal phase, in degrees. The first power point (assumed to be small) is used to calculate the small-signal phase. Phase compression (change) is only available for 1-D power sweep.

pwr_gain()

This measurement is used to determine the transducer power gain (in dB) and is defined as the ratio of the power delivered to the load, to the power available from the source. (where power is in dBm)

Syntax

y = pwr_gain(S, Zs, Zl, Zref)

Arguments

Name	Description	Range	Type	Required
S	2 X 2 scattering matrix	(-∞, ∞)	complex	yes
Zs	input impedance	(-∞, ∞)	real, complex	yes
Zl	Output impedance	(-∞, ∞)	real, complex	yes
Zref	reference impedance	(-∞, ∞)	real, complex	yes

Examples

```
pGain = pwr_gain(S,50,75)
```
Zref defaults to 50 ohms

```
pGain1 = pwr_gain(S, 50, 75, 75)
```
Zref = 75 ohms

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

stos(), volt_gain(), volt_gain_max()

ripple()

This function measures the deviation of x from the average of x

Syntax

y = ripple(x, fstart, fstop)

Arguments

Name	Description	Range	Type	Required
x	can be a gain or group delay data over a given frequency range	(-∞, ∞)	real	yes
fstart	start frequency	(0, ∞)	real	no
fstop	stop frequency	(0, ∞)	real	no

Examples

```
a = ripple(pwr_gain(S21))
```
returns the ripple of the tranducer power gain of S21

```
a1 = ripple(pwr_gain(S21), 1GHz, 3GHz)
```
returns the ripple frequency between the start frequency (1GHz) and stop frequency(3GHz)

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

See Also

dev_lin_gain(), dev_lin_phase(), diff(), mean(), phasedeg(), phaserad(), unwrap()

Notes/Equations

In order to use this function, the *Group Delay* option must be enabled in the S-parameter analysis setup. For more information, refer to "*Calculating Group Delay*" in your "*S-Parameter Simulation*" documentation.

This function supports data up to four dimensions.

s_stab_circle()

Returns source (input) stability circles

Syntax

y = s_stab_circle(S, numOfPts)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network.	(-∞, ∞)	complex		yes
numOfPts	desired number of points per circle	[1, ∞)	integer	51	no

Examples

```
circleData = s_stab_circle(S, 51)
```

returns the points on the circle(s)

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle(), l_stab_region(), s_stab_circle_center_radius(), s_stab_region()

Notes/Equations

Used in Small-signal S-parameter simulations.

This expression generates source stability circles. The circles are defined by the loci of source-reflection coefficients where the magnitude of the load-reflection coefficient is 1. A circle is created for each value of the swept variable(s). This measurement is supported for 2-port networks only.

To find the center and radius of the stability circle use the following expressions:

```
cir=s_stab_circle(S,2)
cir_center=sum((cir[0::1]) /2)
cir_radius=abs(cir[1]-cir[0]) /2
```

Alternately, the function s_stab_circle_center_radius() can be used. Use the function s_stab_region(S) to determine the region of stability.

s_stab_circle_center_radius()

Returns the center and radius of the source stability circle

Syntax

s_cr = s_stab_circle_center_radius(S, Type)

Arguments

Name	Description	Range	Type	Default	Required
S	2-port S-Parameters		complex		yes
Type	Type of parameter to return	["both"|"center"|"radius"]	string	"both"	no

Examples

```
s_cr = s_stab_circle_center_radius(S)
```
returns a 1X2 matrix containing center and radius

```
sCirc = expand(circle(s_cr(1), real(s_cr(2)), 51))
```
returns data for the source stability circle

```
s_radius = s_stab_circle_center_radius(S, "radius")
```
returns the radius

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle(), s_stab_circle(), l_stab_circle_center_radius()

Notes/Equations

Used in Small-signal S-parameter simulations

If the argument *Type* is not specified, the function returns complex data. Although radius is of type real, the values are returned as complex. Therefore, when using the returned radius, use the real part. To obtain the radius as a real number, set *Type* to *radius*.

s_stab_region()

This expression returns a string identifying the region of stability of the corresponding source stability circle

Syntax

y = s_stab_region(S)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of a 2-port network.	(-∞, ∞)	complex	yes

Examples

```
region = s_stab_region(S)
```
returns “Outside” or “Inside”

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle(), l_stab_region(), s_stab_region()

Notes/Equations

Used in Small-signal S-parameter simulations

sm_gamma1()

Returns the simultaneous-match input-reflection coefficient

Syntax

y = sm_gamma1(S)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of 2-port network	(-∞, ∞)	complex	yes

Examples

```
a = sm_gamma1(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

max_gain(), sm_gamma2(), stab_fact(), stab_meas()

Notes/Equations

This complex measurement determines the reflection coefficient that must be presented to the input (port 1) of the network to achieve simultaneous input and output reflections. If the Rollett stability factor stab_fact(S) is less than unity for the analyzed circuit, then sm_gamma1(S) returns 1+j*0.

sm_gamma2()

Returns the simultaneous-match output-reflection coefficient

Syntax

y = sm_gamma2(S)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of 2-port network	(-∞, ∞)	complex	yes

Examples

```
a = sm_gamma2(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

max_gain(), sm_gamma1(), stab_fact(), stab_meas()

Notes/Equations

This complex measurement determines the reflection coefficient that must be presented to the output (port 2) of the network to achieve simultaneous input and output reflections. If the Rollett stability factor stab_fact(S) is less than unity for the analyzed circuit, then sm_gamma2(S) returns1+j*0.

sm_y1()

This complex measurement determines the admittance that must be presented to the input (port 1) of the network to achieve simultaneous input and output reflections

Syntax

y = sm_y1(S, Z)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of a 2-port network	(-∞, ∞)	complex	yes
Z	port impedance	(-∞, ∞)	integer, real or complex	yes

Examples

```
a = sm_y1(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

sm_y2()

sm_y2()

This complex measurement determines the admittance that must be presented to the input (port 1) of the network to achieve simultaneous input and output reflections

Syntax

y = sm_y2(S, Z)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of a 2-port network	(-∞, ∞)	complex	yes
Z	port impedance	(-∞, ∞)	integer, real or complex	yes

Examples

```
a = sm_y2(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

sm_y1()

sm_z1()

This complex measurement determines the impedance that must be presented to the input (port 1) of the network to achieve simultaneous input and output reflections

Syntax

y = sm_z1(S, Z)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of a 2-port network	(-∞, ∞)	complex	yes
Z	port impedance	(-∞, ∞)	integer, real or complex	yes

Examples

```
a = sm_z1(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

sm_z2()

sm_z2()

This complex measurement determines the impedance that must be presented to the output (port 2) of the network to achieve simultaneous input and output reflections

Syntax

y = sm_z2(S, Z)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of a 2-port network	(-∞, ∞)	complex	yes
Z	port impedance	(-∞, ∞)	integer, real or complex	yes

Examples

```
a = sm_z2(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

sm_z1()

stab_fact()

Returns the Rollett stability factor

Syntax

k = stab_fact(S)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of 2-port network	(-∞, ∞)	complex	yes

Examples

```
k = stab_fact(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

max_gain(), sm_gamma1(), sm_gamma2(), stab_meas()

Notes/Equations

Given a 2 x 2 scattering matrix between the input and measurement ports, this function calculates the stability factor. The Rollett stability factor is given by:

$$k = \{1- |S_{11}|^2 - |S_{22}|^2 + |S_{11}*S_{22} - S_{12}*S_{21}|^2\} / \{2*|S_{12}*S_{21}|\}$$

The necessary and sufficient conditions for unconditional stability are that the stability factor is greater than unity and the stability measure is positive.

Reference

[1] Guillermo Gonzales, Microwave Transistor Amplifiers, second edition, Prentice-Hall, 1997.

stab_meas()

Returns the stability measure

Syntax

k = stab_meas(S)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of 2-port network	(-∞, ∞)	complex	yes

Examples

```
b = stab_meas(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

max_gain(), sm_gamma1(), sm_gamma2(), stab_fact()

Notes/Equations

Given a 2 x 2 scattering matrix between the input and measurement ports, this function calculates the stability measure. The stability measure is given by:

$$b = 1 + |S_{11}|^2 - |S_{22}|^2 - |S_{11}*S_{22} - S_{12}*S_{21}|^2$$

The necessary and sufficient conditions for unconditional stability are that the stability factor is greater than unity and the stability measure is positive.

Reference

[1] Guillermo Gonzales, Microwave Transistor Amplifiers, second edition, Prentice-Hall, 1997.

stoabcd()

This measurement transforms the scattering matrix of a 2-port network to a chain (ABCD) matrix

Syntax

y = stoabcd(S, zRef)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	integer, real or complex	50.0	no

Examples

```
a = stoabcd(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh(), stoh(), stoy()

stoh()

This measurement transforms the scattering matrix of a 2-port network to a hybrid matrix

Syntax

y = stoh(S, zRef)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	integer, real or complex	50.0	no

Examples

```
h = stoh(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htos(), stoabcd(), stoy()

stos()

Changes the normalizing impedance of a scattering matrix

Syntax

y = stos(S, zRef, zNew, zy)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix	(-∞, ∞)	complex		yes
zRef	normalizing impedance	(-∞, ∞)	integer, real or complex	50.0	no
zNew	new normalizing impedance	(-∞, ∞)	integer, real or complex	50.0	no
zy	directs the conversion through the Z- or Y-matrix. †	[0, 1]	integer	1 (Y-matrix)	no
† f zy=0, the S-to-S conversion is performed through the Z-matrix. If zy=1, the S-to-S conversion is performed through the Y-matrix					

Examples

```
a = stos(S, 50, 75, 1)
```
converts the 50 ohm terminated S-parameters to 75 ohm terminated S-parameters through the Y-matrix

```
a = stos(S, 75)
```
converts the 75 ohm terminated S-parameters to 50 ohm terminated S-parameters through the Y-matrix

Assume that a two-port S-parameter analysis has been done with port 1terminated in50 ohms, and port 2 in 75 ohms. The expression below converts the S-parameters at a 50 ohm impedance termination at both ports:

```
S50 = stos(S, PortZ, 50, 1)
```
The converted S-parameters can then be written to a S2P file using the function write_snp()

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoy(), stoz()

stot()

This function transforms the scattering matrix of a 2-port network to a chain scattering matrix

Syntax

t = stot(S)

Arguments

Name	Description	Range	Type	Required
S	scattering matrix of a 2-port network	(-∞, ∞)	complex	yes

Examples

```
Tparams = stot(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

ttos()

stoy()

This measurement transforms a scattering matrix to an admittance matrix

Syntax

y = stoy(S, zRef)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	integer, real or complex	50.0	no

Examples

```
y = stoy (S, 50.0)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoh(), stoz(), ytos()

stoz()

This measurement transforms a scattering matrix to an impedance matrix

Syntax

z = stoz(S, zRef)

Arguments

Name	Description	Range	Type	Default	Required
S	scattering matrix of a 2-port network	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	integer, real or complex	50.0	no

Examples

```
z = stoz(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoh(), stoy(), ztos()

tdr_sp_gamma()

Returns step response. This function calculates time domain response from the S-parameter measurement directly. Normalization is taken into account

Syntax

y = tdr_sp_gamma(Sii, delay, Tstart, Tstop, NumPts, window)

Arguments

Name	Description	Range	Type	Default	Required
Sii	Complex reflection coefficient	(-∞, ∞)	complex		yes
delay	delay value	[0, ∞)	real		yes
Tstart	Start Time	[0, ∞)	real	0	no
Tstop	Stop Time	[0, ∞)	real	2 cycles	no
NumPts	Number of points	[0, ∞)	real ot string	101	no
Window	Windowing to be applied	[0, 9] †	real	0	no

† The window types allowed and their default constants are:

0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the frequency-to-time transformation with normal gate window setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13

Examples

```
x = tdr_sp_gamma(S(1,1), 0.05ns, -0.2 ns, 3.8 ns, 401, "Hamming")
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

tdr_sp_imped(), tdr_step_imped()

Notes/Equations

This function takes an S-parameter dataset and changes it to a step response. The step response is normalized and used to calculate reflection coefficient vs. time.

tdr_sp_imped()

Returns step response. This function calculates time domain response from the S-parameter measurement directly. Normalization is taken into account

Syntax

y = tdr_sp_imped(Sii, delay, zRef, Tstart, Tstop, NumPts, window)

Arguments

Name	Description	Range	Type	Default	Required
Sii	Complex reflection coefficient	(-∞, ∞)	complex		yes
delay	delay value	[0, ∞)	real		yes
zRef	Reference impedance	[0, ∞)	real		yes
Tstart	Start Time	[0, ∞)	real	0	no
Tstop	Stop Time	[0, ∞)	real	2 cycles	no
NumPts	Number of points	[0, ∞)	real ot string	101	no
Window	Windowing to be applied	[0, 7] †	real	0	no

† The window types allowed and their default constants are:

0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the frequency-to-time transformation with normal gate window setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13

Examples

```
x = tdr_sp_imped(S(1,1), 0.05ns, 50, -0.2 ns, 3.8 ns, 401, "Hamming")
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

tdr_sp_gamma(), tdr_step_imped()

Notes/Equations

This function takes an S-parameter dataset and changes it to a step response. The step response is normalized and used to calculate reflection coefficient vs. time. This gamma is then used to calculate impedance versus time.

tdr_step_imped()

Returns time domain Impedance. This function essentially takes

Syntax

y = tdr_step_imped(time_waveform, zRef)

Arguments

Name	Description	Range	Type	Required
time_waveform	time domain pulse TDR waveform	(-∞, ∞)	complex	yes
zRef	Reference impedance	[0, ∞)	real	yes

Examples

```
x = tdr_step_imped(vout, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

tdr_sp_gamma(), tdr_sp_imped()

Notes/Equations

This function takes a time domain pulse TDR waveform, and computes the impedance versus time. The function also assumes that a step impulse was applied to the DUT, since it normalizes the impedance data to the last time point.

ttos()

This function transforms the chain scattering matrix of a 2-port network to a scattering matrix

Syntax

sp = ttos(T)

Arguments

Name	Description	Range	Type	Required
T	scattering matrix of a 2-port network	(-∞, ∞)	complex	yes

Examples

```
Sparams = ttos(T)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stot()

unilateral_figure()

Returns the unilateral figure as a real number

Syntax

U = uniltaeral_figure(SParam)

Arguments

Name	Description	Type	Required
SParam	2-Port S-Parameters	complex	yes

Examples

Form an *S* matrix at a single frequency

```
sMat={{polar(0.55,-50), polar(0.02,10)}, {polar(3.82,80),polar(0.15,-20)}}
U = uniltaeral_figure(sMat)
```

returns 0.009

```
U_plus = 10*log(1/(1-U)**2)
```

returns 0.081

```
U_minus = 10*log(1/(1+U)**2)
```

returns -0.08

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

Not applicable

Notes/Equations

Used in Small-signal S-parameter simulations.

This function is used to calculate the Unilateral Figure of Merit, which determines whether the simplification can be made in neglecting the effect of S_{12} (unilateral behavior of device). It is calculated as:

$$U = \frac{|S_{11}||S_{12}||S_{21}||S_{22}|}{(1-|S_{11}|^2)(1-|S_{22}|^2)}$$

The error limit on unilateral figure or merit, U is:

$$\frac{1}{(1+U)^2} < \frac{GT}{GTU} < \frac{1}{(1-U)^2}$$

where:

GT is Transducer Gain

GTU is Unilateral Transducer Gain

This function can be used only with 2-Port S-parameters and works only on 1-dimensional or single swept parameter data.

unwrap()

This measurement unwraps a phase by changing an absolute jump greater than jump to its 2*jump complement

Syntax

y = unwrap(phase, jump)

Arguments

Name	Description	Range	Type	Default	Required
phase	swept real variable	(-∞, ∞)	real		yes
jump	absolute jump	(-∞, ∞)	integer, real	180.0	no

Examples

```
a = unwrap(phase(S21))
a = unwrap(phaserad(S21), pi)
```

Defined in

Built-in

See Also

dev_lin_phase(), diff(), phase(), phasedeg(), phaserad(), ripple()

Notes/Equations

The unwrap function requires that the difference between two succesive data should be less than or greater than 2*jump. Otherwise no jump is made for that phase and the original data is maintained. And, if the number of phase data points is one, no phase is unwrapped and the original data is maintained.

v_dc()

Returns the voltage difference

Syntax

y = v_dc(vPlus, vMinus)

Arguments

Name	Description	Range	Type	Required
vPlus	voltage at the positive output terminal	(-∞, ∞)	real, complex	yes
vMinus	voltage at the negative output terminal	(-∞, ∞)	real, complex	yes

Examples

```
y = v_dc(vp, vm)
```

Defined In

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

volt_gain()

Returns the voltage gain

Syntax

y= volt_gain(S, Zs, Zl, Zref)

Arguments

Name	Description	Range	Type	Required
S	2 2 scattering matrix measured with equal terminations of Zref		complex	yes
Zs	input impedance	(-∞, ∞)	real, complex	yes
Zl	Output impedance	(-∞, ∞)	real, complex	yes
Zref	reference impedance	(-∞, ∞)	real, complex	yes

Examples

```
a = volt_gain(S, 50, 75)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pwr_gain(), volt_gain_max()

Notes/Equations

This function calculates the ratio of the voltage across the load impedance to the voltage applied at the input port of the network. The network-parameter transformation function *stos()* can be used to change the normalizing impedance of the scattering matrix.

$$\text{volt\_gain} = \frac{S_{21}}{2} \cdot \frac{(1-\Gamma_S)\cdot(1+\Gamma_L)}{(S_{11}\cdot\Gamma_S-1)\cdot(S_{22}\cdot\Gamma_L-1)-S_{12}\cdot S_{21}\cdot\Gamma_S\cdot\Gamma_L} \qquad (9\text{-}1)$$

Figure 9-1 illustrates the volt_gain measurement.

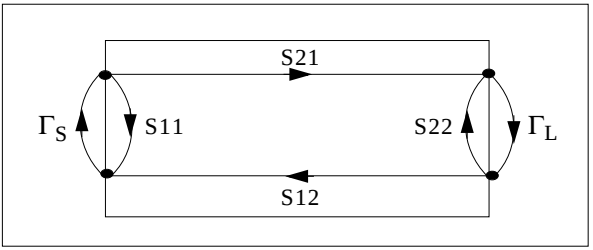


Figure 9-1. Signal Flow Diagram used for volt_gain Calculation

1. In the S-parameter simulation setup, the source and load impedances must be identical.
2. For a case of unequal source and load impedances, S-parameter analysis should be performed with identical source and load impedances. Voltage gain can then be computed with the actual source and load impedances as the second and third arguments.

   For example, compute voltage gain with Zs=100 and Zl=50. Perform an S-parameter analysis with both the Zs=Zl=50 ohms. The voltage gain is computed as follows:

   volt_gain(S, 100, 50, 50)

   This expression gives the voltage gain when the source impedance is 100 ohms and the load impedance is 50 ohms. The fourth argument in volt_gain is the reference impedance, which is the value of the Z parameter of the Term components used in the S-parameter analysis.

volt_gain_max()

This measurement determines the ratio of the voltage across the load to the voltage available from the source at maximum power transfer

Syntax

y = volt_gain_max(S, Zs, Zl, Zref)

Arguments

Name	Description	Range	Type	Required
S	2 X 2 scattering matrix	(-∞, ∞)	complex	yes
Zs	input impedance	(-∞, ∞)	real, complex	yes
Zl	Output impedance	(-∞, ∞)	real, complex	yes
Zref	reference impedance	(-∞, ∞)	real, complex	yes

Examples

```
vGain = volt_gain_max(S, 50, 75)
```
Zref defaults to 50 ohms

```
vGain1 = volt_gain(S, 50, 75, 75)
```
Zref = 75 ohms

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

pwr_gain(), volt_gain()

Note/Equations

The network-parameter transformation function stos() can be used to change the normalizing impedance of the scattering matrix.

vswr()

Given a complex reflection coefficient, this measurement returns the voltage standing wave ratio

Syntax

y = stab_meas(Sii)

Arguments

Name	Description	Range	Type	Required
Sii	complex reflection coefficient	(-∞, ∞)	complex	yes

Examples

```
a = vswr(S11)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

yin(), zin()

write_snp()

Write S-Parameters in Touchstone SnP file format. Returns True or False

Syntax

y = write_snp(FileName, S, Comment, FreqUnit, DataFormat, Zref, Znorm, ZorY, Precision, Delimiter)

Arguments

Name	Description	Range	Type	Default	Required
FileName	Name or full-path of the S-Parameter file.		string		yes
S	S-parameter matrix variable		real, complex		yes
Comment	Text that is to be written at the top of file.		string	""	no
FreqUnit	Frequency Unit	"Hz", "KHz", "MHz", "GHz", "THz"	string	GHz	no
DataFormat	Format of S-Parameter that is to be output	"MA", "DB", "RI" †	string	"MA"	no
Zref	Reference impedance (scalar or vector)	(0, ∞)	real	50	no
Znorm	Normalizing impedance (a scalar value)	(0, ∞)	real	50	no
ZorY	Directs the conversion through Z or Y transform ‡	[0, 1]	integer	1 (Y-matrix)	no
Precision	precision of the data	[1, 64]	integer	6	no
Delimiter	Delimiter that separates the data		string	""	no

† "MA" = magnitude-phase, "DB" = dB-phase, "RI" = real-imaginary
‡ If ZorY=0, the S-to-S conversion is performed through the Z-matrix. If ZorY=1, the S-to-S conversion is performed through the Y-matrix

Examples

These examples assume that an S-parameter Analysis has been performed.

```
write_snp("spar_ts.s2p", S, "S-par simulation data", "GHz", "MA", 50)
```
writes the S-parameters to the file spar_ts.s2p in mag-phase format

```
write_snp("spar_ts_1.s2p", S, "S-par simulation data")
```
writes the S-parameters to the file spar_ts_1.s2p in default "GHz", mag-phase format and reference impedance of 50.0

Assuming that a 2-port S-parameter analysis has been performed with source terminated in 60 ohms and load terminated in 70 ohms:

```
write_snp("spar_norm_ts.s2p", S, "S-par simulation data", "GHz", "MA", PortZ,
50, 1, 9, "  ")
```
writes the S-parameters to the file spar_norm_ts.s2p in "GHz", mag-phase format with 9 digit precision and delimited by " " and normalized impedance of 50.0

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stos()

Notes/Equations

The function supports only 1-dimensional S-parameter data. S-parameters can be from 1 to 99 ports. The S-parameters to be written can be normalized by a different impedance through the arguments *Znorm* and *ZorY*. If the argument Znorm is not specified, then the S-parameters are normalized to 50 ohms.

When using this function from the schematic page, the output file will be written in the project's data directory. When used from the Data Display, the file will be written to the project's directory.

yin()

Given a reflection coefficient and the reference impedance, this measurement returns the input admittance looking into the measurement ports

Syntax

y = yin(Sii, Z)

Arguments

Name	Description	Range	Type	Default	Required
Sii	complex reflection coefficient	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	integer, real or complex	50.0	no

Examples

```
yIN = yin(S11, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

vswr(), zin()

yopt()

Returns optimum admittance for noise match

Syntax

y = yopt(gammaOpt, zRef)

Arguments

Name	Description	Range	Type	Default	Required
gammaOpt	optimum reflection coefficient	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	real, complex	50.0	no

Examples

```
a = yopt(Sopt, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

zopt()

Notes/Equations

Used in Small-signal S-parameter simulations

This complex measurement produces the optimum source admittance for noise matching. *gammaOpt* is the optimum reflection coefficient that must be presented at the input of the network to realize the minimum noise figure (NFmin).

ytoabcd()

This measurement transforms an admittance matrix of a 2-port network into a hybrid matrix

Syntax

a = ytoabcd(Y)

Arguments

Name	Description	Range	Type	Required
Y	2-port admittance matrix	(-∞, ∞)	complex	yes

Examples

```
a = ytoabcd(Y)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh(), htoabcd()

ytoh()

This measurement transforms an admittance matrix of a 2-port network into a hybrid matrix

Syntax

h = ytoh(Y)

Arguments

Name	Description	Range	Type	Required
Y	2-port admittance matrix	(-∞, ∞)	complex	yes

Examples

```
h = ytoh(Y)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htoy(), ytoabcd()

ytos()

This measurement transforms an admittance matrix into a scattering matrix

Syntax

z = ytos(Y, zRef)

Arguments

Name	Description	Range	Type	Default	Required
Y	admittance matrix	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	integer, real or complex	50.0	no

Examples

```
s = ytos(Y, 50.0)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoy(), ytoz()

ytoz()

This measurement transforms an admittance matrix to an impedance matrix

Syntax

Z = ytoz(Y)

Arguments

Name	Description	Range	Type	Required
Y	admittance matrix	(-∞, ∞)	complex	yes

Examples

```
a = ytoz(Y)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

ytos(), ztoy()

zin()

Given a reflection coefficient and the reference impedance, this measurement returns the input impedance looking into the measurement ports

Syntax

z = zin(Sii, Z)

Arguments

Name	Description	Range	Type	Default	Required
Sii	complex reflection coefficient.	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	integer, real or complex	50.0	no

Examples

```
zIN = zin(S11, 50.0)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

vswr(), yin()

zopt()

Returns optimum impedance for noise match

Syntax

y = zopt(gammaOpt, zRef)

Arguments

Name	Description	Range	Type	Default	Required
gammaOpt	optimum reflection coefficient	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	real, complex	50.0	no

Examples

```
a = zopt(Sopt, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

yopt()

Notes/Equations

Used in Small-signal S-parameter simulations.

This complex measurement produces the optimum source impedance for noise matching. *gammaOpt* is the optimum reflection coefficient that must be presented at the input of the network to realize the minimum noise figure (NFmin).

ztoabcd()

This measurement transforms an impedance matrix of a 2-port network into a chain (ABCD) matrix

Syntax

a = ztoabcd(Z)

Arguments

Name	Description	Range	Type	Required
Z	2-port impedance matrix	(-∞, ∞)	complex	yes

Examples

```
a = ztoabcd(Z)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoz(), ytoabcd(), ztoh()

ztoh()

This measurement transforms an impedance matrix of a 2-port network into a hybrid matrix

Syntax

h = ztoh(Z)

Arguments

Name	Description	Range	Type	Required
Z	2-port impedance matrix	(-∞, ∞)	complex	yes

Examples

```
h = ztoh(Z)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htoz(), ytoh(), ztoabcd()

ztos()

This measurement transforms an impedance matrix to a scattering matrix

Syntax

sp = ztos(Z, zRef)

Arguments

Name	Description	Range	Type	Default	Required
Z	impedance matrix	(-∞, ∞)	complex		yes
zRef	reference impedance	(-∞, ∞)	integer, real or complex	50.0	no

Examples

```
s = ztos(Z, 50.0)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoz(), ytoz(), ztoy()

ztoy()

This measurement transforms an impedance matrix to an admittance matrix

Syntax

y = ztoy(Z)

Arguments

Name	Description	Range	Type	Required
Z	impedance matrix	(-∞, ∞)	complex	yes

Examples

```
a = ztoy(Z)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoz(), ytos(), ztoy()

Chapter 10: Statistical Analysis Functions

This chapter describes the statistical analysis functions in detail. The functions are listed in alphabetical order.

C,F,H,L,M,N,P,S,U,Y

cdf()

Returns the cumulative distribution function (CDF)

Syntax

y = cdf(data, numBins, minBin, maxBin)

Arguments

Name	Description	Range	Type	Default	Required
data	the signal	(-∞, ∞)	real		yes
numBins	number of subintervals or bins used to measure CDF	[1, ∞)	real	log(numOfPts)/log(2.0)	no
minBin	beginning of the evaluation of the CDF	(-∞, ∞)	real	minimum value of the data	no
maxBin	end of the evaluation of the CDF	(-∞, ∞)	real	maximum value of the data	no

Examples

```
y = cdf(data)
y = cdf(data, 20)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

histogram(), pdf(), yield_sens()

Notes/Equations

This function measures the cumulative distribution function of a signal. This function can only be used by entering an equation (Eqn) in the Data Display window.

cross_corr()

Returns the cross-correlation

Syntax

y = cross_corr(v1, v2)

Arguments

Name	Description	Range	Type	Required
v1	one-dimensional data	(-∞, ∞)	real	yes
v2	one-dimensional data	(-∞, ∞)	real	yes

Examples

```
v1 = qpsk..videal[1]
v2 = qpsk..vout[1]
x_corr_v1v2 = cross_corr(v1, v2)
auto_corr_v1 = cross_corr(v1, v1)
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

fun_2d_outer()

Applies a function to the outer dimension of two-dimensional data

Syntax

y = fun_2d_outer(data, fun)

Arguments

Name	Description	Range	Type	Required
data	two-dimensional data	(-∞, ∞)	integer, real, complex	yes
fun	name of function (usually mean, max, or min) that will be applied to the outer dimension of the data		string	yes

Examples

```
y = fun_2d_outer(data, min)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistcal_fun.ael

See Also

max_outer(), mean_outer(), min_outer()

Notes/Equations

Used in max_outer(), mean_outer(), min_outer() functions.

Functions such as mean, max, and min operate on the inner dimension of two-dimensional data. The function fun_2d_outer enables these functions to be applied to the outer dimension. As an example, assume that a Monte Carlo simulation of an amplifier was run, with 151 random sets of parameter values, and that for each set the S-parameters were simulated over 26 different frequency points. S21 becomes a [151 Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being frequency, and the outer dimension being Monte Carlo index. Now, assume that it is desired to know the mean value of the S-parameters at each frequency. Inserting an equation mean(S21) computes the mean value of S21 at each Monte Carlo iteration. If S21 is simulated from 1 to 26 GHz, it computes the mean value over this frequency range, which usually is not very useful. The function

fun_2d_outer allows the mean to be computed over each element in the outer dimension.

histogram()

Generates a histogram representation. This function creates a histogram that represents data

Syntax

y = histogram(data, numBins, minBin, maxBin)

Arguments

Name	Description	Range	Type	Default	Required
data	signal(must be one-dimensional)	(-∞, ∞)	integer, real		yes
numBins	number of subintervals or bins used to measure the histogram.	[1, ∞)	integer	log(numOfPts)/log(2.0)	no
minBin	beginning of the evaluation of the histogram	(-∞, ∞)	real	minimum value of the data	no
maxBin	end of the evaluation of the histogram	(-∞, ∞)	real	maximum value of the data	no

Examples

```
y = histogram(data)
y = histogram(data, 20)
```

If you have performed a parameter sweep such that the first argument (*data*) in the histogram function is a function of two independent variables, then you must reduce the dimensionality of *data* before using it in the histogram function. For example, if you run a Monte Carlo simulation on the S-parameters of a circuit, S_{21} would be a function of both the Monte Carlo index and the frequency (assuming you have swept frequency). So, you could plot the histogram of S_{21} at the 100th frequency in the sweep by using:

```
y =  histogram(dB(S21[::,99]))
```

Defined in

Built in

See Also

cdf(), pdf(), yield_sens(), histogram_multiDim(), histogram_stat()

Notes/Equations

This function can only be used by entering an equation (Eqn) in the Data Display window. Use the *histogram_multiDim()* function for multi-dimensional data.

histogram_multiDim()

Collapses the multi-dimensional data down to one-dimensional data and applies the histogram to the one-dimensional data

Syntax

y = histogram_multiDim(data, normalized, numBins, minBin, maxBin)

Arguments

Name	Description	Range	Type	Default	Required
data	the signal	(-∞, ∞)	real		yes
normalized	sets normalization of data †	"no", "yes"	string	no	no
numBins	number of subintervals or bins used to measure CDF	[1, ∞)	real	log(numOfPts)/log(2.0)	no
minBin	beginning of the evaluation of the CDF	(-∞, ∞)	real	minimum value of the data	no
maxBin	end of the evaluation of the CDF	(-∞, ∞)	real	maximum value of the data	no
† When normalized is set to "yes", the histogram is generated with percent on the Y-axis instead of the number of outcomes					

Examples

Given monte carlo analysis results for the S_{12}. It is two-dimensional data: the outer sweep is mcTrial; the inner sweep is the frequency from 100 MHz to 500 MHz.

```
Histogram_multiDim_S12 = histogram_multiDim(S12)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

collapse(), histogram()

histogram_sens()

Produces the yield sensitivity histogram displaying the sensitivity of a measurement statistical response to a selected statistical variable. The function is mainly applied to the statistical analysis (Monte Carlo/Yield/YieldOpt) results

Syntax

y = histogram_sens(data, sensitivityVar, goalMin, goalMax, innermostIndepLow, innermostIndepHigh, numBins)

Arguments

Name	Description	Range	Type	Default	Required
data	statistical response	(-∞, ∞)	real		yes
sensitivityVar	selected statistical variable		string		yes
goalMin	specifies the performance range †	(-∞, ∞)	real		no
goalMax	specifies the performance range †	(-∞, ∞)	real		no
innermostIndepLow	specifies the low value of the subrange of data with the inner most sweep variable	(-∞, ∞)	real		yes
innermostIndepHigh	specifies the high value of the subrange of data with the inner most sweep variable	(-∞, ∞)	real		yes
numBins	number of sub-intervals or bins used to measure the histogram.	(1, ∞)	real	log(numOfPts)/log(2.0)	no

† The yield is 1 inside the range, and the yield is zero outside the range. Note that while goalMin and goalMax are optional arguments, at least one of them must be specified

Examples

Given Monte Carlo analysis results for the S_{11}. It is two-dimensional data: the outer sweep is mcTrial; the inner sweep is the frequency from 100 MHz to 500 MHz.

The design requires the maximum of db(S11), -18.0dB in the frequency range of 200 MHz to 400 MHz. The yield sensitivity of such performance to statistical variable "C1v" can be calculated as:

```
Histogram_sens_S11 = histogram_sens(dB(S11),C1v,,-18.0,200MHz,400MHz)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

histogram(), histogram_multiDim(), histogram_stat(), build_subrange(), yield_sens()

histogram_stat()

Reduces the histogram of a subrange of the multi-dimension data. It first calls build_subrange() to build the subrange of a mulit-dimension data, then calls histogram_multiDim() to produce the histogram

Syntax

y = histogram_stat(data, normalized, innermostIndepLow, innermostIndepHigh, numBins, minBin, maxBin)

Arguments

Name	Description	Range	Type	Default	Required
data	statistical data to be analyzed	(-∞, ∞)	real		yes
normalized	sets normalization of data †	"no", "yes"	string	no	no
innermostIndepLow	specifies the low value of the subrange of data with the inner most sweep variable	(-∞, ∞)	real		yes
innermostIndepHigh	specifies the high value of the subrange of data with the inner most sweep variable	(-∞, ∞)	real		yes
numBins	number of subintervals or bins used to measure histogram	[1, ∞)	real	log(numOfPts)/log(2.0)	no
minBin	beginning of the evaluation of the histogram	(-∞, ∞)	real	minimum value of the data	no
maxBin	end of the evaluation of the histogram	(-∞, ∞)	real	maximum value of the data	no
† When normalized is set to "yes", the histogram is generated with percent on the Y-axis instead of the number of outcomes					

Examples

Given monte carlo analysis results for the S_{12}. It is two-dimensional data: the outer sweep is mcTrial; the inner sweep is the frequency from 100 MHz to 500 MHz.

```
Histogram_stat_S12 = histogram_stat(S12,,200MHz,400MHz)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

histogram(), histogram_multiDim(), build_subrange()

lognorm_dist_inv1D()

Returns the inverse of the cumulative distribution function (cdf) for a lognormal distribution

Syntax

y = lognorm_dist_inv1D(data, m, s, absS)

Arguments

Name	Description	Range	Type	Default	Required
data	real number representing the cumulative probability	[0, 1]	real		yes
m	mean for the lognormal distribution	(-∞, ∞)	integer, real		yes
s	standard deviation for the lognormal distribution	(-∞, ∞)	integer, real		yes
absS	specifies absolute or relative standard deviation	[0, ∞) †	integer	1	no
† absS is not equal to "0", s is the absolute standard deviation; otherwise, s is the relative standard deviation					

Examples

```
X_cpf = 0.5
X = lognorm_dist_inv1D(X_cpf, 2.0, 0.2, 1)
XX_cpf = [0.0::0.01::1.0]
XX = lognorm_dist_inv1D(XX_cpf, 2.0, 0.2, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

lognorm_dist1D()

Returns a lognormal distribution: either its probability density function (pdf), or cumulative distribution function (cdf)

Syntax

y = lognorm_dist1D(data, m, s, absS, is_cpf)

Arguments

Name	Description	Range	Type	Default	Required
data	number or a one-dimensional data	(-∞, ∞)	real		yes
m	mean for the lognormal distribution	(-∞, ∞)	integer, real		yes
s	standard deviation for the lognormal distribution	(-∞, ∞)	integer, real		yes
absS	specifies absolute or relative standard deviation	[0, ∞) †	integer	1	no
is_cpf	specifies cdf or pdf og lognormal distribution	[0, ∞) † †	integer	0	no

† When absS is not equal to "0", s is the absolute standard deviation; otherwise, s is the relative standard deviation
† † When is_cdf is not equal to "0", the function returns the cdf of the lognormal distribution. Otherwise, it returns the pdf of the lognormal distribution.

Examples

```
X = 0.5
X_pdf = lognorm_dist1D(X, 2.0,0.2, 1, 0)
X_cdf = lognorm_dist1D(X,2.0,0.1, 0, 1)

XX = [-3.9::0.1::3.9]
XX_pdf = lognorm_dist1D(XX,2.0, 0.2,1,0)
XX_cdf = lognorm_dist1D(XX,2.0,0.2,0,1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

mean()

Returns the mean

Syntax

y = mean(x)

Arguments

Name	Description	Range	Type	Required
x	data to find mean	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = mean([1, 2, 3])
```
returns 2

Defined in

Built in

See Also

cum_prod(), cum_sum(), max(), min(), prod(), sum()

mean_outer()

Computes the mean across the outer dimension of two-dimensional data

Syntax

y = mean_outer(data)

Arguments

Name	Description	Range	Type	Required
data	2-dimensional data to find mean	(-∞, ∞)	integer, real, complex	yes

Examples

```
a = mean_outer(data)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistcal_fun.ael

See Also

fun_2d_outer(), max_outer(), min_outer()

Notes/Equations

The mean function operates on the inner dimension of two-dimensional data. The mean_outer function just calls the fun_2d_outer function, with mean being the applied operation. As an example, assume that a Monte Carlo simulation of an amplifier was run, with 151 random sets of parameter values, and that for each set the S-parameters were simulated over 26 different frequency points. S21 becomes a [151 Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being frequency, and the outer dimension being Monte Carlo index. Now, assume that it is desired to know the mean value of the S-parameters at each frequency. Inserting an equation mean(S21) computes the mean value of S21 at each Monte Carlo iteration. If S21 is simulated from 1 to 26 GHz, it computes the mean value over this frequency range, which usually is not very useful. Inserting an equation mean_outer(S21) computes the mean value of S21 at each Monte Carlo frequency.

median()

Returns the median

Syntax

y = median(x)

Arguments

Name	Description	Range	Type	Required
data	data to find the median	(-∞, ∞)	real	yes

Examples

```
a = median([1, 2, 3, 4])
```

returns 2.5

Defined in

$HPEESOF_DIR/expressions/ael/statistcal_fun.ael

See Also

mean(), sort()

Notes/Equations

This function can only be used by entering an equation (Eqn) in the Data Display window.

moving_average()

Returns the moving_average of a sequence of data

Syntax

y = moving_average(Data, NumPoints)

Arguments

Name	Description	Range	Type	Required
Data	one or multi-dimensional sequence of numbers	(-∞, ∞)	integer, real or complex	yes
NumPoints	Number of points to be averaged together	[1, ∞) †	integer	yes
† NumPoints must be an odd number. If even, the value is increased to the next odd number. If greater or equal to the number of data points, the value is set to number of data points - 1 for even number of data points. For odd number of data points, NumPoints is set to number of data points				

Examples

```
a = moving_average([1, 2, 3, 7, 5, 6, 10], 3)
```
returns [1, 2, 4, 5, 6, 7, 10]

Defined in

Built in

Notes/Equations

The first value of the smoothed sequence is the same as the original data. The second value is the average of the first three. The third value is the average of data elements 2, 3, and 4, etc. If NumPoints were set to 7, for example, then the first value of the smoothed sequence would be the same as the original data. The second value would be the average of the first three original data points. The third value would be the average of the first five data points, and the fourth value would be the average of the first seven data points. Subsequent values in the smoothed array would be the average of the seven closest neighbors. The last points in the smoothed sequence are computed in a way similar to the first few points. The last point is just the last point in the original sequence. The second from last point is the average of the last three points in the original sequence. The third from the last point is the average of the last five points in the original sequence, etc.

norm_dist_inv1D()

Returns the inverse of the cumulative distribution function (cdf) for a normal distribution

Syntax

y = norm_dist_inv1D(data, m, s, absS)

Arguments

Name	Description	Range	Type	Default	Required
data	represents the cumulative probability	[0, 1]	real		yes
m	mean for the normal distribution	(-∞, ∞)	integer, real		yes
s	standard deviation for the normal distribution	(-∞, ∞)	integer, real		yes
absS	specifies absolute or relative standard deviation	[0, ∞) †	integer	1	no
† When absS is not equal to "0", s is the absolute standard deviation; otherwise, s is the relative standard deviation					

Examples

```
X_cpf = 0.5
X = norm_dist_inv1D(X_cpf, 0, 1, 1)
```

will be equal to 0.0

```
XX_cpf = [0.0::0.01::1.0]
XX = norm_dist_inv1D(XX_cpf, 5.0, 0.5, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist1D(), norms_dist_inv1D(), norms_dist1D(), lognorm_dist_inv1D(), lognorm_dist1D(), uniform_dist_inv1D(), uniform_dist1D()

norm_dist1D()

Returns a normal distribution: either its probability density function (pdf), or cumulative distribution function (cdf)

Syntax

y = norm_dist1D(data, m, s, absS, is_cdf)

Arguments

Name	Description	Range	Type	Default	Required
data	number or a one-dimensional data	(-∞, ∞)	real		yes
m	mean for the normal distribution	(-∞, ∞)	integer, real		yes
s	standard deviation for the normal distribution	(-∞, ∞)	integer, real		yes
absS	specifies absolute or relative standard deviation	[0, ∞) †	integer	1	no
is_cdf	specifies cdf or pdf of normal distribution	[0, ∞) † †	integer	0	no

† When absS is not equal to "0", s is the absolute standard deviation; otherwise, s is the relative standard deviation
† † When is_cdf is not equal to "0", the function returns the cdf of the normal distribution. Otherwise, it returns the pdf of the normal distribution.

Examples

```
X = 0.5
X_pdf = norm_dist1D(X, 0, 1, 1, 0)
X_cdf = norm_dist1D(X,0, 1, 1, 1)
```

```
XX = [-3.9::0.1::3.9]
XX_pdf = norm_dist1D(XX,5.0, 0.5,1,0)
XX_cdf = norm_dist1D(XX,5.0,0.1,0,1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist_inv1D(), norms_dist_inv1D(), norms_dist1D(), lognorm_dist_inv1D(),

lognorm_dist1D(), uniform_dist_inv1D(), uniform_dist1D()

norms_dist_inv1D()

Returns the inverse of the cumulative distribution function (cdf) for a standard normal distribution

Syntax

y = norms_dist_inv1D(data)

Arguments

Name	Description	Range	Type	Required
data	number represents the cumulative probability	[0, 1]	integer, real	yes

Examples

```
X_cpf = 0.5
X = norms_dist_inv1D(X_cpf)
```

will be equal to 0.0

```
XX_cpf = [0.0::0.01::1.0]
XX = norms_dist_inv1D(XX_cpf)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist1D(), norm_dist_inv1D(), norms_dist1D(), lognorm_dist_inv1D(), lognorm_dist1D(), uniform_dist_inv1D(), uniform_dist1D()

norms_dist1D()

Returns the standard normal distribution: either its probability density function (pdf), or cumulative distribution function (cdf)

Syntax

y = norms_dist1D(data, is_cdf)

Arguments

Name	Description	Range	Type	Default	Required
data	number or a one-dimensional data	(-∞, ∞)	real		yes
is_cdf	specifies cdf or pdf of standard normal distribution	[0, ∞) †	integer	0	no
† When is_cdf is not equal to "0", the function returns the cdf of the standard normal distribution. Otherwise, it returns the pdf of the standard normal distribution					

Examples

```
X = 0.5
X_pdf = norms_dist1D(X, 0)
X_cdf = norms_dist1D(X,1)
```

```
XX = [-3.9::0.1::3.9]
XX_pdf = norms_dist1D(XX,0)
XX_cdf = norms_dist1D(XX,1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist1D(), norm_dist_inv1D(), norms_dist_inv1D(), lognorm_dist_inv1D(), lognorm_dist1D(), uniform_dist_inv1D(), uniform_dist1D()

pdf()

Returns a probability density function (PDF)

Syntax

y = pdf(data, numBins, minBin, maxBin)

Arguments

Name	Description	Range	Type	Default	Required
data	the signal	(-∞, ∞)	real		yes
numBins	number of subintervals or bins used to measure PDF	[1, ∞)	real	log(numOfPts)/log(2.0)	no
minBin	beginning of the evaluation of the PDF	(-∞, ∞)	real	minimum value of the data	no
maxBin	end of the evaluation of the PDF	(-∞, ∞)	real	maximum value of the data	no

Examples

```
y = pdf(data)
y = pdf(data, 20)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

cdf(), histogram(), yield_sens()

Notes/Equations

This function measures the probability density function of a signal. This function can only be used by entering an equation (Eqn) in the Data Display window.

stddev()

This function calculates the standard deviation of the data

Syntax

y = stddev(data, flag)

Arguments

Name	Description	Range	Type	Default	Required
data	data to find the stddev	(-∞, ∞)	real		yes
flag	indicates how stddev normalizes	[0, 1] †	integer	0	no
† When flag equals 0, the stddev normalizes by N-1, where N is the length of the data sequence. Otherwise, stddev normalizes by N					

Examples

```
a = stddev(data)
a = stddev(data, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

mean()

Notes/Equations

This function can only be used by entering an equation (Eqn) in the Data Display window.

stddev_outer()

Computes the stddev across the outer dimension of two-dimensional data

Syntax

y = stddev_outer(x, flag)

Arguments

Name	Description	Range	Type	Default	Required
data	data to find the stddev	(-∞, ∞)	real		yes
flag	indicates how stddev normalizes	[0, 1] †	integer	0	no
† When flag equals 0, the stddev normalizes by N-1, where N is the length of the data sequence. Otherwise, stddev normalizes by N					

Examples

```
a = stddev_outer(data)
a = stddev_outer(data, 1)
```

Defined In

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

fun_2d_outer(), max_outer(), mean_outer(), min_outer()

Notes/Equations

The stddev_outer() function operates on the inner dimension of two-dimensional data. This function just calls the fun_2d_outer function, with stddev being the applied operation. As an example, assume that a Monte Carlo simulation of an amplifier was run, with 151 random sets of parameter values, and that for each set the S-parameters were simulated over 26 different frequency points. S21 becomes a [151 Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being frequency, and the outer dimension being Monte Carlo index. Now, assume that it is desired to know the stddev value of the S-parameters at each frequency. Inserting an equation stddev(S21) computes the stddev value of S21 at each Monte Carlo iteration. If S21 is simulated from 1 to 26 GHz, it computes the stddev value over this frequency range, which usually is not very useful. Inserting an equation stddev_outer(S21) computes the stddev value of S21 at each Monte Carlo frequency.

uniform_dist_inv1D()

Returns the inverse of the cumulative distribution function (cdf) for a uniform distribution

Syntax

y = uniform_dist_inv1D(data, A, B)

Arguments

Name	Description	Range	Type	Required
data	represents the cumulative probability	[0, 1]	integer, real	yes
A	uniform distributed range	[0, ∞)	integer, real	yes
B	uniform distributed range	[0, ∞)	integer, real	yes

Examples

```
X_cpf = 0.5
X = uniform_dist_inv1D(X_cpf, 0.0, 1.5)
```

```
XX_cpf = [0.0::0.01::1.0]
XX = uniform_dist_inv1D(XX_cpf, 0.0, 1.5)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist1D(), norm_dist_inv1D(), norms_dist_inv1D(), norms_dist1D(), lognorm_dist_inv1D(), lognorm_dist1D(), uniform_dist1D()

uniform_dist1D()

Returns a uniform distribution: either its probability density function (pdf), or cumulative distribution function (cdf)

Syntax

y = uniform_dist1D(data, A, B, is_cdf)

Arguments

Name	Description	Range	Type	Default	Required
data	number or a one-dimensional data	(-∞, ∞)	integer, real		yes
A	uniform distributed range	[0, ∞)	integer, real		yes
B	uniform distributed range	[0, ∞)	integer, real		yes
is_cdf	specifies cdf or pdf of standard uniform distribution	[0, ∞) †	integer	0	no
† When is_cdf is not equal to "0", the function returns the cdf of the uniform distribution. Otherwise, it returns the pdf of the uniform distribution					

Examples

```
X = 0.5
X_pdf = uniform_dist1D(X, 0.0,1.0, 0)
X_cdf = uniform_dist1D(X,0.0,1.0, 1)
```

```
XX = [-3.9::0.1::3.9]
XX_pdf = uniform_dist1D(XX,0.0,5.0,0)
XX_cdf = uniform_dist1D(XX,0.0,5.0,1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist1D(), norm_dist_inv1D(), norms_dist_inv1D(), norms_dist1D(), lognorm_dist_inv1D(), lognorm_dist1D(), uniform_dist_inv1D()

yield_sens()

Returns the yield as a function of a design variable

Syntax

y = yield_sens(pf_data, numBins)

Arguments

Name	Description	Range	Type	Default	Required
pf_data	binary-valued scalar data set indicating the pass/fail status of each value of a companion independent variable	[0-1]	integer		yes
numBins	number of subintervals or bins used to measure yield_sens	[1, ∞)	real	log(numOfPts)/log(2.0)	no

Examples

```
a = yield_sens(pf_data)
a = yield_sens(pf_data, 20)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

cdf(), histogram(), pdf()

Notes/Equations

Used in Monte Carlo simulation.

This function measures the yield as a function of a design variable. For more information and an example refer to "*Creating a Sensitivity Histogram*" in the *Optimization and Statistical Design* documentation.

This function can only be used by entering an equation (Eqn) in the Data Display window.

Chapter 11: Transient Analysis Functions

This chapter describes the transient analysis functions in detail. The functions are listed in alphabetical order.

C,F,I,P,S,V

Working with Transient Data

Transient analysis produces real voltages and currents as a function of time. A single analysis produces 1-dimensional data. Sections of time-domain waveforms can be indexed by using a sequence within “[ ]”.

constellation()

Generates the constellation diagram from Circuit Envelope, Transient, or Ptolemy simulation I and Q data

Syntax

Const = constellation(i_data, q_data, symbol_rate, delay)

Arguments

Name	Description	Range	Type	Required
i_data	in-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †	(-∞, ∞)	complex	yes
q_data	quadrature-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †	(-∞, ∞)	real	yes
symbol_rate	symbol rate of the modulation signal	[0, ∞)	real	yes
delay	delay value † †	[0, ∞)	real	no

† this could be a baseband signal instead, but in either case it must be real-valued versus time.
† † (if nonzero) throws away the first delay = N seconds of data from the constellation plot. It is also used to interpolate between simulation time points, which is necessary if the optimal symbol-sampling instant is not exactly at a simulation time point. Usually this parameter must be nonzero to generate a constellation diagram with the smallest grouping of sample points

Examples

```
Rotation = -0.21
Vfund =vOut[1] * exp(j * Rotation)
delay =1/sym_rate[0, 0] - 0.5 * tstep[0, 0]
Vimag = imag(Vfund)
Vreal = real(Vfund)
Const = constellation(Vreal, Vimag, sym_rate[0, 0], delay)
```

where Rotation is a user-selectable parameter that rotates the constellation by that

many radians, and vOut is the named connection at a node. The parameter delay can be a numeric value, or in this case an equation using sym_rate, the symbol rate of the modulated signal, and tstep, the time step of the simulation. If these equations are to be used in a Data Display window, sym_rate and tstep must be defined by means of a variable (VAR) component, and they must be passed into the dataset as follows: Make the parameter Other visible on the Envelope simulation component, and edit it so that,

Other = OutVar = sym_rate OutVar = tstep

In some cases, it may be necessary to experiment with the value of delay to get the constellation diagram with the tightest points.

Note vOut is a named connection on the schematic. Assuming that a Circuit Envelope simulation was run, vOut is output to the dataset as a two-dimensional matrix. The first dimension is time, and there is a value for each time point in the simulation. The second dimension is frequency, and there is a value for each fundamental frequency, each harmonic, and each mixing term in the analysis, as well as the baseband term.

vOut[1] is the equivalent of vOut[::, 1], and specifies all time points at the lowest non-baseband frequency (the fundamental analysis frequency, unless a multitone analysis has been run and there are mixing products). For former MDS users, the notation "vOut[*, 2]" in MDS corresponds to the notation of "vOut[1]".

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

const_evm()

Notes/Equations

Used in Constellation diagram generation.

The I and Q data do not need to be baseband waveforms. For example, they could be the in-phase (real or I) and quadrature-phase (imaginary or Q) part of a modulated carrier. The user must supply the I and Q waveforms versus time, as well as the symbol rate. A delay parameter is optional.

The i_data and q_data must be of the same dimension, and up to 5-dimensional data is supported.

cross()

Computes the zero crossings of a signal, interval between successive zero crossings or slope at the crossing

Syntax

yCross = cross(signal, direction, slope)

Arguments

Name	Description	Range	Type	Default	Required
signal	the signal for which the zero crossing is to be found	(-∞, ∞)	real		yes
direction	type of zero crossing †	[-1, 1]	integer		yes
slope	specifies if slope is to be calculated, rather than interval between zero crossing	[0, 1]	integer	0 (no slope)	no

† If direction = +1, compute positive going zero crossings.
If direction = -1, compute negative going zero crossings.
If direction = 0, compute all zero crossings

Examples

```
period = cross(vosc-2.0, 1)
```
this computes the period of each cycle of the vosc signal. The period is measured from each positive-going transition through 2.0V

```
period = cross(vosc-2.0, 1, 1)
```
returns the zero crossings and the slope at the zero crossings.

Defined In

Built in

Notes/Equations

The independent axis returns the time when the crossing occurred. If the third argument is set to 1, the dependent axis returns the slope at zero crossing. Otherwise the dependent axis returns the time interval since the last crossing (default behavior).

fspot()

Performs a single-frequency time-to-frequency transform

Syntax

y = fspot(x, fund, harm, windowType, windowConst, interpOrder, tstart)

Arguments

Name	Description	Range	Type	Default	Required
x	time domain signal	(-∞, ∞)	real		yes
fund	period 1/fund for the Fourier transform	[1, ∞)	real	period that matches the length of the independent axis of x	no
harm	harmonic number †	[1, ∞)	integer	1	no
windowType	type of window to apply to the data	[0, 9] ‡	integer, string	0	no
windowConst	window constant ‡ †	[0, ∞)	real	1	no
interpOrder	interpolation scheme	[1, 3] ‡ ‡	integer		no

Name	Description	Range	Type	Default	Required
tstart	start time	[0, ∞)	real	1	no

† harm=0 will compute the dc component of x.
‡ The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13
windowType can be specified either by the number or by the name.

‡ † windowConst is not used if windowType is 8510

‡ ‡ If the tranorder variable is not present, or if the user wishes to override the interpolation scheme, then interpOrder may be set to a nonzero value:
1 = use only linear interpolation
2 = use quadratic interpolation
3 = use cubic polynomial interpolation

Examples

The following example equations assume that a transient simulation was performed from 0 to 5 ns on a 1-GHz-plus-harmonics signal called vOut.

```
y = fspot(vOut)
```
returns the 200-MHz component, integrated from 0 to 5 ns

```
y = fspot(vOut, , 5)
```
returns the 1-GHz component, integrated from 0 to 5 ns

```
y = fspot(vOut, 1GHz, 1)
```
returns the 1-GHz component, integrated from 4 to 5 ns

```
y = fspot(vOut, 0.5GHz, 2, , , , 2.5ns)
```
returns the 1-GHz component, integrated from 2.5 to 4.5 ns

```
y = fspot(vOut, 0.25GHz, 4, "Kaiser")
```
returns the 1-GHz component, integrated from 1 to 5 ns, after applying the default Kaiser window to this range of data

```
y = fspot(vOut, 0.25GHz, 4, 3, 2.0)
```
returns the 1-GHz component, integrated from 1 to 5 ns, after applying a Gaussian window with a constant of 2.0 to this range of data

Defined in

Built in

See Also

fft(), fs()

Notes/Equations

fspot(x) returns the discrete Fourier transform of the vector x evaluated at one specific frequency. The value returned is the peak component, and it is complex. The harmth harmonic of the fundamental frequency fund is obtained from the vector x. The Fourier transform is applied from time tstop-1/fund to tstop, where tstop is the last timepoint in x.

When x is a multidimensional vector, the transform is evaluated for each vector in the specified dimension. For example, if x is a matrix, then fspot(x) applies the transform to every row of the matrix. If x is three dimensional, then fspot(x) is applied in the lowest dimension over the remaining two dimensions. The dimension over which to apply the transform may be specified by dim; the default is the lowest dimension (dim=1). x must be numeric. It will typically be data from a transient, signal processing, or envelope analysis.

By default, the transform is performed at the end of the data from tstop-1/fund to tstop. By using tstart, the transform can be started at some other point in the data. The transform will then be performed from tstart to tstart+1/fund.

Unlike with fft() or fs(), the data to be transformed are not zero padded or resampled. fspot() works directly on the data as specified, including non-uniformly sampled data from a transient simulation.

Transient simulation uses a variable timestep and variable order algorithm. The user sets an upper limit on the allowed timestep, but the simulator will control the timestep so the local truncation error of the integration is controlled. If the Gear integration algorithm is used, the order can also be changed during simulation. fspot() can use all of this information when performing the Fourier transform. The time data are not resampled; the Fourier integration is performed from timestep to timestep of the original data.

When the order varies, the Fourier integration will adjust the order of the polynomial it uses to compute the shape of the data between timepoints.

This variable order integration depends on the presence of a special dependent variable, tranorder, which is output by the transient simulator. If this variable is not present, or if the user wishes to override the interpolation scheme, then interpOrder may be set to a nonzero value.

Only polynomials of degree one to three are supported. The polynomial is fit because time domain data are obtained by integrating forward from zero; previous data are used to determine future data, but future data can never be used to modify past data.

ifc_tran()

Returns frequency-selective current in Transient analysis

Syntax

y = ifc_tran(iOut, fundFreq, harmNum)

Arguments

Name	Description	Range	Type	Required
iOut	current through a branch	(-∞, ∞)	real, complex	yes
fundFreq	fundamental frequency	[0, ∞)	real	yes
harmNum	harmonic number of the fundamental frequency	[0, ∞)	integer	yes

Examples

```
y = ifc_tran(I_Probe1.i, 1GHz, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pfc_tran(), vfc_tran()

Notes/Equations

This measurement gives RMS current, in current units, for a specified branch at a particular frequency of interest. *fundFreq* determines the portion of the time-domain waveform to be converted to the frequency domain. This is typically one full period corresponding to the lowest frequency in the waveform. *harmNum* is the harmonic number of the fundamental frequency at which the current is requested.

ispec_tran()

Returns current spectrum

Syntax

y = ispec_tran(iOut, fundFreq, numHarm, windowType, windowConst)

Arguments

Name	Description	Range	Type	Default	Required
iOut	current through a branch	(-∞, ∞)	real, complex		yes
fundFreq	fundamental frequency	[0, ∞)	real		yes
numHarm	number of harmonics of fundamental frequency	[0, ∞)	integer		yes
windowType	type of window to be applied to the data	[0, 9] †	integer, string	0	no
windowConst	window constant ‡ †	[0, ∞)	integer, real	0	no

† The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the time-to-frequency transformation with normalgate shape setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13

Examples

```
y = ispec_tran(I_Probe1.i, 1GHz, 8)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pspec_tran(), vspec_tran()

Notes/Equations

This measurement gives a current spectrum for a specified branch. The measurement gives a set of RMS current values at each frequency. The *fundFreq* argument determines the portion of the time-domain waveform to be converted to frequency domain. This is typically one full period corresponding to the lowest frequency in the waveform. The *numHarm* argument is the number of harmonics of fundamental frequency to be included in the currents spectrum.

pfc_tran()

Returns frequency-selective power

Syntax

y = pfc_tran(vPlus, vMinus, iOut, fundFreq, harmNum)

Arguments

Name	Description	Range	Type	Required
vPlus	voltage at the positive terminal	(-∞, ∞)	real, complex	yes
vMinus	voltage at the negative terminal	(-∞, ∞)	real, complex	yes
iOut	current through a branch measured for power calculation	(-∞, ∞)	real, complex	yes
fundFreq	fundamental frequency	[0, ∞)	real	yes
harmNum	harmonic number of the fundamental frequency	[0, ∞)	integer	yes

Examples

```
a = pfc_tran(v1, v2, I_Probe1.i, 1GHz, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ifc_tran(), vfc_tran()

Notes/Equations

This measurement gives RMS power, delivered to any part of the circuit at a particular frequency of interest. *fundFreq* determines the portion of the time-domain waveform to be converted to frequency domain. This is typically one full period corresponding to the lowest frequency in the waveform. *harmNum* is the harmonic number of the fundamental frequency at which the power is requested.

pt_tran()

This measurement produces a transient time-domain power waveform for specified nodes

Syntax

y = pt_tran(vPlus, vMinus, current, fundFreq)

Arguments

Name	Description	Range	Type	Required
vPlus	voltage at the positive terminal	(-∞, ∞)	real, complex	yes
vMinus	voltage at the negative terminal	(-∞, ∞)	real, complex	yes
current	current	(-∞, ∞)	real, complex	yes
fundFreq	fundamental frequency	[0, ∞)	real	yes

Examples

```
a = pt_tran(v1, v2, i1, 1GHz)
```

Defined In

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

vt(), vt_tran()

Notes/Equations

DC-to-RF efficiency is based on HB analysis.

pspec_tran()

Returns transient power spectrum

Syntax

y = pspec_tran(vPlus, vMinus, iOut, fundFreq, numHarm, windowType, windowConst)

Arguments

Name	Description	Range	Type	Default	Required
vPlus	voltage at the positive terminal	(-∞, ∞)	real, complex		yes
vMinus	voltage at the negative terminal	(-∞, ∞)	real, complex		yes
iOut	current through a branch measured for power calculation	(-∞, ∞)	real, complex		yes
fundFreq	fundamental frequency	[0, ∞)	real		yes
numHarm	number of harmonics of fundamental frequency	[0, ∞)	integer		yes
windowType	type of window to be applied to the data	[0, 9] †	integer, string	0	no
windowConst	window constant ‡ †	[0, ∞)	integer, real	0	no

† The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the time-to-frequency transformation with normalgate shape setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13

Examples

```
a = pspec_tran(v1, v2, I_Probe1.i, 1GHz, 8)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ispec_tran(), vspec_tran()

Notes/Equations

This measurement gives a power spectrum, delivered to any part of the circuit. The measurement gives a set of RMS power values at each frequency. The *fundFreq* argument is the fundamental frequency that determines the portion of the time-domain waveform to be converted to frequency domain (typically one full period corresponding to the lowest frequency in the waveform). The *numHarm* argument is the number of harmonics of the fundamental frequency to be included in the power spectrum.

vfc_tran()

Returns the transient frequency-selective voltage

Syntax

y = vfc_tran(vPlus, vMinus, fundFreq, harmNum)

Arguments

Name	Description	Range	Type	Required
vPlus	voltage at the positive terminal	(-∞, ∞)	real, complex	yes
vMinus	voltage at the negative terminal	(-∞, ∞)	real, complex	yes
fundFreq	fundamental frequency	[0, ∞)	real	yes
harmNum	harmonic number of the fundamental frequency	[0, ∞)	integer	yes

Examples

```
a = vfc_tran(vOut, 0, 1GHz, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ifc_tran(), pfc_tran()

Notes/Equations

This measurement gives the RMS voltage across any two nodes at a particular frequency of interest. The fundamental frequency determines the portion of the time-domain waveform to be converted to frequency domain. This is typically one full period corresponding to the lowest frequency in the waveform. The harmonic number is the fundamental frequency at which the voltage is requested (positive integer value only).

vspec_tran()

Returns the transient voltage spectrum

Syntax

y = vspec_tran(vPlus, vMinus, fundFreq, numHarm, windowType, windowConst)

Arguments

Name	Description	Range	Type	Default	Required
vPlus	voltage at the positive terminal	(-∞, ∞)	real, complex		yes
vMinus	voltage at the negative terminal	(-∞, ∞)	real, complex		yes
fundFreq	fundamental frequency	[0, ∞)	real		yes
numHarm	number of harmonics of fundamental frequency	[0, ∞)	integer		yes
windowType	type of window to be applied to the data	[0, 9] †	integer, string	0	no
windowConst	window constant	[0, ∞)	integer, real	0	no

† The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the time-to-frequency transformation with normalgate shape setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13

Examples

```
a = vspec_tran(v1, v2, 1GHz, 8)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ispec_tran(), pspec_tran()

Notes/Equations

This measurement gives a voltage spectrum across any two nodes. The measurement gives a set of RMS voltages at each frequency. The fundamental frequency determines the portion of the time-domain waveform to be converted to the frequency domain. This is typically one full period corresponding to the lowest frequency in the waveform. The *numHarm* argument is the number of harmonics of the fundamental frequency to be included in the voltage spectrum.

vt_tran()

This measurement produces a transient time-domain voltage waveform for specified nodes. vPlus and vMinus are the nodes across which the voltage is measured

Syntax

y = vt_tran(vPlus, vMinus)

Arguments

Name	Description	Range	Type	Required
vPlus	voltage at the positive terminal	(-∞, ∞)	real, complex	yes
vMinus	voltage at the negative terminal	(-∞, ∞)	real, complex	yes

Examples

```
a = vt_tran(v1, v2)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

vt()

Index

N

O

P

R

S