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, Inc. 1983-2005
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.

SystemC® is a registered trademark of Open SystemC Initiative, Inc. in the United States and other countries and is used with permission.
## Contents

1 **Introduction to Measurement Expressions**  
   - Measurement Expressions Syntax ................................................................. 1-3  
   - Case Sensitivity .......................................................................................... 1-4  
   - Variable Names ......................................................................................... 1-4  
   - Built-in Constants ................................................................................... 1-5  
   - Operator Precedence .............................................................................. 1-5  
   - Conditional Expressions ........................................................................ 1-6  
   - Manipulating Simulation Data with Expressions ........................................ 1-7  
     - Simulation Data ..................................................................................... 1-7  
     - Measurements and Expressions ............................................................ 1-8  
     - Generating Data .................................................................................... 1-8  
     - Simple Sweeps and Using “[ ]” ................................................................. 1-9  
     - S-Parameters and Matrices .................................................................... 1-9  
     - Matrices ................................................................................................. 1-10  
     - Multidimensional Sweeps and Indexing ................................................ 1-10  
   - User-Defined Functions .......................................................................... 1-10  
   - Functions Reference Format .................................................................... 1-12

2 **Using Measurement Expressions in Advanced Design System**  
   - MeasEqn (Measurement Equations Component) .......................................... 2-2  
   - Pre-Configured Measurements in ADS ...................................................... 2-4  
   - Measurement Expressions Examples in ADS ........................................... 2-8  
   - Alphabetical Listing of Measurement Functions ......................................... 2-11

3 **Circuit Budget Functions**  
   - Budget Measurement Analysis ................................................................ 3-1  
     - Frequency Plan ....................................................................................... 3-2  
     - Reflection and Backward-Travelling Wave Effects .................................. 3-3

4 **Circuit Envelope Functions**  
   - Working with Envelope Data .................................................................... 4-1

5 **Data Access Functions**

6 **Harmonic Balance Functions**  
   - Working with Harmonic Balance Data ...................................................... 6-1
7 Math Functions
8 Signal Processing Functions
9 S-Parameter Analysis Functions
10 Statistical Analysis Functions
11 Transient Analysis Functions
   Working with Transient Data ................................................................. 11-1
12 FrontPanel Eye Diagram Functions
   Working with the Eye Diagram FrontPanel ........................................... 12-2
Index
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.
Introduction to Measurement Expressions

- 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

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

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
Introduction to Measurement Expressions

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>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>function call, matrix indexer</td>
<td>foo(expr_list) X(expr,expr)</td>
</tr>
<tr>
<td>[]</td>
<td>sweep indexer, sweep generator</td>
<td>X[expr_list] [expr_list]</td>
</tr>
<tr>
<td>{}</td>
<td>matrix generator</td>
<td>{expr_list}</td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
<td>expr**expr</td>
</tr>
<tr>
<td>!</td>
<td>not</td>
<td>!expr</td>
</tr>
<tr>
<td>*</td>
<td>multiply</td>
<td>expr * expr</td>
</tr>
<tr>
<td>/</td>
<td>divide</td>
<td>expr / expr</td>
</tr>
<tr>
<td>.*</td>
<td>element-wise multiply</td>
<td>expr .* expr</td>
</tr>
<tr>
<td>./</td>
<td>element-wise divide</td>
<td>expr ./ expr</td>
</tr>
<tr>
<td>+</td>
<td>add</td>
<td>expr + expr</td>
</tr>
<tr>
<td>-</td>
<td>subtract</td>
<td>expr - expr</td>
</tr>
<tr>
<td>::</td>
<td>sequence operator</td>
<td>exp::expr::expr start::inc::stop ::</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>expr &lt; expr</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>expr &lt;= expr</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>expr &gt; expr</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>expr &gt;= expr</td>
</tr>
<tr>
<td>==, EQUALS</td>
<td>equal</td>
<td>expr == expr</td>
</tr>
<tr>
<td>!=, NOTEQUALS</td>
<td>not equal</td>
<td>expr != expr</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
<td>expr &amp;&amp; expr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

\[
A = \begin{cases} \text{true_expression} & \text{if (condition) then} \\ \text{false_expression} & \text{else} \end{cases}
\]

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.

```plaintext
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  andOp1 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  orOp1 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 6 of the Using Circuit Simulators documentation.

**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
Introduction to Measurement Expressions

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:

\[
\begin{align*}
  v1 &= \{1,2,3,4,5\} \text{ five-element vector } \\
  v2 &= \{1::5\} \text{ five-element vector using the sequence operator } \\
  v3 &= \{\{1,0\}, \{0,1\}\} \text{ 2X2 identity matrix }
\end{align*}
\]

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

\[
a = X[\text{start}::\text{increment}::\text{stop}]
\]

Because increment defaults to one,

\[
a = X[\text{start}::\text{stop}]
\]

is equivalent to

\[
a = X[\text{start}::1::\text{stop}]
\]

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 Smn.
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:
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.
Introduction to Measurement Expressions

**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 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.
Using Measurement Expressions in Advanced Design System

**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.

---

2-2 MeasEqn (Measurement Equations Component)
The if-then-else construct can be used in a MeasEqn component on a schematic. It has the following syntax: 

\[ A = \text{if} ( \text{condition} ) \text{ then true_expression } \text{ else false_expression} \]
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

<table>
<thead>
<tr>
<th>Simulator Palette</th>
<th>Pre-configured Measurement</th>
<th>Measurement Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation-AC</td>
<td>BudFreq</td>
<td>“bud_freq()” on page 3-4</td>
</tr>
<tr>
<td></td>
<td>BudGain</td>
<td>“bud_gain()” on page 3-7</td>
</tr>
<tr>
<td></td>
<td>BudGainComp</td>
<td>“bud_gain_comp()” on page 3-10</td>
</tr>
<tr>
<td></td>
<td>BudGamma</td>
<td>“bud_gamma()” on page 3-13</td>
</tr>
<tr>
<td></td>
<td>BudIP3Deg</td>
<td>“bud_ip3_deg()” on page 3-15</td>
</tr>
<tr>
<td></td>
<td>BudNF</td>
<td>“bud_nf()” on page 3-17</td>
</tr>
<tr>
<td></td>
<td>BudNFDeg</td>
<td>“bud_nf_deg()” on page 3-19</td>
</tr>
<tr>
<td></td>
<td>BudNoisePwr</td>
<td>“bud_noise_pwr()” on page 3-21</td>
</tr>
<tr>
<td></td>
<td>BudPwrInc</td>
<td>“bud_pwr_inc()” on page 3-25</td>
</tr>
<tr>
<td></td>
<td>BudPwrRefl</td>
<td>“bud_pwr_refl()” on page 3-27</td>
</tr>
<tr>
<td></td>
<td>BudSNR</td>
<td>“bud_snr()” on page 3-29</td>
</tr>
<tr>
<td></td>
<td>BudTN</td>
<td>“bud_tn()” on page 3-31</td>
</tr>
<tr>
<td></td>
<td>BudVSWR</td>
<td>“bud_vswr()” on page 3-33</td>
</tr>
</tbody>
</table>
Table 2-1. Pre-configured Measurements Available in ADS

<table>
<thead>
<tr>
<th>Simulator Palette</th>
<th>Pre-configured Measurement</th>
<th>Measurement Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation-S_Param</td>
<td>MaxGain</td>
<td>“max_gain()” on page 9-31</td>
</tr>
<tr>
<td></td>
<td>PwrGain</td>
<td>“pwr_gain()” on page 9-39</td>
</tr>
<tr>
<td></td>
<td>VoltGain</td>
<td>“volt_gain()” on page 9-69</td>
</tr>
<tr>
<td></td>
<td>VSWR</td>
<td>“vswr()” on page 9-72</td>
</tr>
<tr>
<td></td>
<td>GainRipple</td>
<td>“ripple()” on page 9-40</td>
</tr>
<tr>
<td></td>
<td>Mu</td>
<td>“mu()” on page 9-32</td>
</tr>
<tr>
<td></td>
<td>MuPrime</td>
<td>“mu_prime()” on page 9-33</td>
</tr>
<tr>
<td></td>
<td>StabFact</td>
<td>“stab_fact()” on page 9-50</td>
</tr>
<tr>
<td></td>
<td>StabMeas</td>
<td>“stab_meas()” on page 9-51</td>
</tr>
<tr>
<td></td>
<td>SmGamma1</td>
<td>“sm_gamma1()” on page 9-44</td>
</tr>
<tr>
<td></td>
<td>SmGamma2</td>
<td>“sm_gamma2()” on page 9-45</td>
</tr>
<tr>
<td></td>
<td>SmY1</td>
<td>“sm_y1()” on page 9-46</td>
</tr>
<tr>
<td></td>
<td>SmY2</td>
<td>“sm_y2()” on page 9-47</td>
</tr>
<tr>
<td></td>
<td>SmZ1</td>
<td>“sm_z1()” on page 9-48</td>
</tr>
<tr>
<td></td>
<td>SmZ2</td>
<td>“sm_z2()” on page 9-49</td>
</tr>
<tr>
<td></td>
<td>Yin</td>
<td>“yin()” on page 9-75</td>
</tr>
<tr>
<td></td>
<td>Zin</td>
<td>“zin()” on page 9-81</td>
</tr>
<tr>
<td></td>
<td>Yopt</td>
<td>“yopt()” on page 9-76</td>
</tr>
<tr>
<td></td>
<td>Zopt</td>
<td>“zopt()” on page 9-82</td>
</tr>
<tr>
<td></td>
<td>NsPwrInt</td>
<td>“ns_pwr_int()” on page 9-36</td>
</tr>
<tr>
<td></td>
<td>NsPwrRefBW</td>
<td>“ns_pwr_ref_bw()” on page 9-37</td>
</tr>
<tr>
<td></td>
<td>DevLinPhase</td>
<td>“dev_lin_phase()” on page 9-11</td>
</tr>
<tr>
<td></td>
<td>GrpDelayRipple</td>
<td>“ripple()” on page 9-40</td>
</tr>
<tr>
<td></td>
<td>GaCircle</td>
<td>“ga_circle()” on page 9-12</td>
</tr>
<tr>
<td></td>
<td>GICircle</td>
<td>“gl_circle()” on page 9-15</td>
</tr>
<tr>
<td></td>
<td>GpCircle</td>
<td>“gp_circle()” on page 9-17</td>
</tr>
<tr>
<td>Simulator Palette</td>
<td>Pre-configured Measurement</td>
<td>Measurement Expression</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td><strong>Simulation-S_Param</strong> (cont.)</td>
<td>GsCircle</td>
<td>“gs_circle()” on page 9-19</td>
</tr>
<tr>
<td></td>
<td>S_StabCircle</td>
<td>“s_stab_circle()” on page 9-41</td>
</tr>
<tr>
<td></td>
<td>L_StabCircle</td>
<td>“l_stab_circle()” on page 9-26</td>
</tr>
<tr>
<td></td>
<td>Map1Circle</td>
<td>“map1_circle()” on page 9-29</td>
</tr>
<tr>
<td></td>
<td>Map2Circle</td>
<td>“map2_circle()” on page 9-30</td>
</tr>
<tr>
<td></td>
<td>NsCircle</td>
<td>“ns_circle()” on page 9-34</td>
</tr>
<tr>
<td><strong>Simulation-LSSP</strong></td>
<td>MaxGain</td>
<td>“max_gain()” on page 9-31</td>
</tr>
<tr>
<td></td>
<td>PwrGain</td>
<td>“pwr_gain()” on page 9-39</td>
</tr>
<tr>
<td></td>
<td>VoltGain</td>
<td>“volt_gain()” on page 9-69</td>
</tr>
<tr>
<td></td>
<td>VSWR</td>
<td>“vswr()” on page 9-72</td>
</tr>
<tr>
<td></td>
<td>GainRipple</td>
<td>“ripple()” on page 9-40</td>
</tr>
<tr>
<td></td>
<td>StabFact</td>
<td>“stab_fact()” on page 9-50</td>
</tr>
<tr>
<td></td>
<td>StabMeas</td>
<td>“stab_meas()” on page 9-51</td>
</tr>
<tr>
<td></td>
<td>Yin</td>
<td>“yin()” on page 9-75</td>
</tr>
<tr>
<td></td>
<td>Zin</td>
<td>“zin()” on page 9-81</td>
</tr>
<tr>
<td></td>
<td>DevLinPhase</td>
<td>“dev_lin_phase()” on page 9-11</td>
</tr>
<tr>
<td></td>
<td>GainComp</td>
<td>“gain_comp()” on page 9-14</td>
</tr>
<tr>
<td></td>
<td>PhaseComp</td>
<td>“phase_comp()” on page 9-38</td>
</tr>
<tr>
<td><strong>Simulation-XDB</strong></td>
<td>CDRRange</td>
<td>“cdrange()” on page 6-3</td>
</tr>
<tr>
<td><strong>Simulation-Transient</strong></td>
<td>IfcTran</td>
<td>“ifc_tran()” on page 11-9</td>
</tr>
<tr>
<td></td>
<td>IspecTran</td>
<td>“ispec_tran()” on page 11-10</td>
</tr>
<tr>
<td></td>
<td>PfcTran</td>
<td>“pfc_tran()” on page 11-12</td>
</tr>
<tr>
<td></td>
<td>PspecTran</td>
<td>“pspec_tran()” on page 11-14</td>
</tr>
<tr>
<td></td>
<td>VfcTran</td>
<td>“vfc_tran()” on page 11-16</td>
</tr>
<tr>
<td></td>
<td>VspecTran</td>
<td>“vspec_tran()” on page 11-17</td>
</tr>
</tbody>
</table>
Table 2-1. Pre-configured Measurements Available in ADS

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

<table>
<thead>
<tr>
<th>Example Design/Data Display</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>express_meas_prj/networks/simple_meas_1.dsn</td>
<td>“Measurements and Expressions” on page 1-8</td>
</tr>
<tr>
<td>express_meas_prj/variable.dds</td>
<td>“Generating Data” on page 1-8</td>
</tr>
<tr>
<td>express_meas_prj/if_then_else_1.dds</td>
<td>“Conditional Expressions” on page 1-6</td>
</tr>
<tr>
<td>express_meas_prj/gen_1.dds</td>
<td>“Generating Data” on page 1-8</td>
</tr>
<tr>
<td>express_meas_prj/sweep.dds</td>
<td>“Simple Sweeps and Using “[ ]”” on page 1-9</td>
</tr>
<tr>
<td>express_meas_prj/sparam_1.dsn &amp; analysis.dds (see S-parameter 1 page)</td>
<td>“S-Parameters and Matrices” on page 1-9</td>
</tr>
<tr>
<td>express_meas_prj/Matrix.dds</td>
<td>“Matrices” on page 1-10</td>
</tr>
<tr>
<td>express_meas_prj/multidim_1.dsn &amp; multidim_1.dds</td>
<td>“Multidimensional Sweeps and Indexing” on page 1-10</td>
</tr>
<tr>
<td>express_meas_prj/analysis.dds (see Harmonic Balance page)</td>
<td>“Working with Harmonic Balance Data” on page 6-1</td>
</tr>
<tr>
<td>express_meas_prj/tran_1.dsn &amp; tran_1.dds</td>
<td>“Working with Transient Data” on page 11-1</td>
</tr>
<tr>
<td>express_meas_prj/env_1.dds</td>
<td>“Working with Envelope Data” on page 4-1</td>
</tr>
<tr>
<td>DataAccess_prj/Truth_MonteCarlo.dds</td>
<td>These functions can be applied to the data in the example: “max_outer()” on page 7-62 “mean_outer()” on page 10-16 “min_outer()” on page 7-65</td>
</tr>
</tbody>
</table>
Additional measurement expression examples can be found in the ADS DesignGuides:

### Table 2-2. ADS Examples Using Measurement Expressions

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

### Table 2-3. Examples Using the ADS DesignGuides

<table>
<thead>
<tr>
<th>DesignGuide</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refer to the S-Parameter to Time Transform &amp; Single Ended TDR/ TDT Impulse Simulation in the Signal Integrity Applications under the DesignGuide menu. SP_measVSmod_NEW.dds</td>
<td>“tdr_sp_gamma()” on page 9-59</td>
</tr>
<tr>
<td></td>
<td>“tdr_sp_imped()” on page 9-61</td>
</tr>
<tr>
<td></td>
<td>“tdr_step_imped()” on page 9-63</td>
</tr>
</tbody>
</table>
### Table 2-3. Examples Using the ADS DesignGuides

<table>
<thead>
<tr>
<th>DesignGuide</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refer to the <strong>Eye Diagram Jitter Histogram Measurement</strong> in the Signal Integrity Applications under the DesignGuide menu</td>
<td>“cross_hist()” on page 4-15</td>
</tr>
<tr>
<td>Refer to the <strong>Eye Closure Measurements</strong> in the Signal Integrity Applications under the DesignGuide menu</td>
<td>“eye_amplitude()” on page 8-8</td>
</tr>
<tr>
<td></td>
<td>“eye_closure()” on page 8-10</td>
</tr>
<tr>
<td></td>
<td>“eye_fall_time()” on page 8-12</td>
</tr>
<tr>
<td></td>
<td>“eye_height()” on page 8-14</td>
</tr>
<tr>
<td></td>
<td>“eye_rise_time()” on page 8-16</td>
</tr>
</tbody>
</table>
Alphabetical Listing of Measurement Functions

Consult the Index for an alternate method of accessing measurement functions.
For information on simulator functions, refer to the Simulator Expressions documentation.

A
“abcdtoh()” on page 9-3 “acpr_vi()” on page 4-3
“abcdtos()” on page 9-4 “acpr_vr()” on page 4-5
“abcdtoy()” on page 9-5 “add_rf()” on page 8-2
“abcdtoz()” on page 9-6 “asin()” on page 7-9
“abs()” on page 7-4 “asinh()” on page 7-10
“acos()” on page 7-5 “atan()” on page 7-11
“acosh()” on page 7-6 “atan2()” on page 7-12
“acot()” on page 7-7 “atanh()” on page 7-13
“acoth()” on page 7-8

B
“bandwidth_func()” on page 9-7 “bud_nf_deg()” on page 3-19
“ber_pi4dqpsk()” on page 8-3 “bud_noise_pwr()” on page 3-21
“ber_qpsk()” on page 8-5 “bud_pwr()” on page 3-23
“bud_freq()” on page 3-4 “bud_pwr_incl()” on page 3-25
“bud_gain()” on page 3-7 “bud_pwr_refl()” on page 3-27
“bud_gain_comp()” on page 3-10 “bud_snr()” on page 3-29
“bud_gamma()” on page 3-13 “bud_tn()” on page 3-31
“bud_ip3_deg()” on page 3-15 “bud_vswr()” on page 3-33
“bud_nf()” on page 3-17 “build_subrange()” on page 5-2

C
Using Measurement Expressions in Advanced Design System

“carr_to_im()” on page 6-2
“cdf()” on page 10-2
“cdrange()” on page 6-3
“ceil()” on page 7-14
“center_freq()” on page 9-9
“channel_power_vi()” on page 4-8
“channel_power_vr()” on page 4-10
“chop()” on page 5-3
“chr()” on page 5-4
“cint()” on page 7-15
“circle()” on page 5-5
“cmplx()” on page 7-16
“collapse()” on page 5-6
“complex()” on page 7-17
“conj()” on page 7-18
“const_evm()” on page 4-12
“constellation()” on page 11-2
“contour()” on page 5-8

D
“db()” on page 7-29
“dbm()” on page 7-30
“dbmtow()” on page 7-32
“dc_to_rf()” on page 6-4
“deg()” on page 7-33
“delay_path()” on page 4-17

E
“erf()” on page 7-36

“contour_polar()” on page 5-10
“convBin()” on page 7-19
“convHex()” on page 7-20
“convInt()” on page 7-21
“convOct()” on page 7-22
“copy()” on page 5-12
“cos()” on page 7-23
“cosh()” on page 7-24
“cot()” on page 7-25
“coth()” on page 7-26
“create()” on page 5-13
“cross()” on page 11-4
“cross_corr()” on page 10-3
“cross_hist()” on page 4-15
“cum_prod()” on page 7-27
“cum_sum()” on page 7-28
“cot()” on page 7-25

“delete()” on page 5-16
“dev_lin_gain()” on page 9-10
“dev_lin_phase()” on page 9-11
“diff()” on page 7-35
“diagonal()” on page 7-34

“expand()” on page 5-17

2-12  Alphabetical Listing of Measurement Functions
“erfc()” on page 7-37  “eye_amplitude()” on page 8-8
“erfcinv()” on page 7-38  “eye_binning()” on page 12-3
“erfinv()” on page 7-39  “eye_closure()” on page 8-10
“evm_wlan_dsss_cck_pbcc()” on page 4-18  “eye_density()” on page 12-4
“evm_wlan_ofdm()” on page 4-28  “eye_fall_time()” on page 8-12
“exp()” on page 7-40  “eye_height()” on page 8-14
“eye()” on page 8-7  “eye_rise_time()” on page 8-16

F
“fft()” on page 7-41  “FrontPanel_eye_risefall_marker()” on page 12-20
“find()” on page 5-19  “FrontPanel_eye_topbase()” on page 12-22
“find_index()” on page 5-21  “FrontPanel_get_histogram_mean_st ddev()” on page 12-24
“fix()” on page 7-42  “FrontPanel_pp_rms_jitter()” on page 12-26
“float()” on page 7-43  “FrontPanel_wave_1st_falling_edge_period()” on page 12-27
“floor()” on page 7-44  “FrontPanel_wave_1st_rising_edge_period()” on page 12-29
“fmod()” on page 7-45  “FrontPanel_wave_1st_transition_fall_time()” on page 12-31
“FrontPanel_eye()” on page 12-5  “FrontPanel_wave_1st_transition_ris e_time()” on page 12-33
“FrontPanel_eye_2d indepvar maximum inner()” on page 12-7
“FrontPanel_eye_amplitude_histogram()” on page 12-8
“FrontPanel_eye_crossings()” on page 12-9  “FrontPanel_wave_datarate()” on page 12-35
“FrontPanel_wave_negative_pulse_width()” on page 12-36
“FrontPanel_wave positive pulse width()” on page 12-38
Using Measurement Expressions in Advanced Design System

“FrontPanel_eye_delay()” on page 12-11
“FrontPanel_wave_topbase()” on page 12-40
“FrontPanel_eye_fall_trace()” on page 12-13
“fs()” on page 4-37
“FrontPanel_eye_horizontal_histogram()” on page 12-15
“fspot()” on page 11-5
“FrontPanel_eye_regular()” on page 12-17
“fun_2d_outer()” on page 10-4
“FrontPanel_eye_rise_trace()” on page 12-18

G
“ga_circle()” on page 9-12
“get_indep_values()” on page 5-24
“gain_comp()” on page 9-14
“gl_circle()” on page 9-15
“generate()” on page 5-22
“gp_circle()” on page 9-17
“get_attr()” on page 5-23
“gs_circle()” on page 9-19

H
“histogram()” on page 10-6
“htos()” on page 9-22
“histogram_multiDim()” on page 10-8
“htoy()” on page 9-23
“histogram_sens()” on page 10-9
“htoz()” on page 9-24
“histogram_stat()” on page 10-11
“hypot()” on page 7-46
“htoabcd()” on page 9-21

I
“identity()” on page 7-47
“interpolate()” on page 7-54
“ifc()” on page 6-5
“inverse()” on page 7-55
“ifc_tran()” on page 11-9
“ip3_in()” on page 6-6
“im()” on page 7-48
“ip3_out()” on page 6-8
“imag()” on page 7-49
“ipn()” on page 6-10
“indep()” on page 5-26  “ispec()” on page 9-25
“int()” on page 7-50  “ispec_tran()” on page 11-10
“integrate()” on page 7-51  “it()” on page 6-12
“interp()” on page 7-53

J
“jn()” on page 7-56

L
“l_stab_circle()” on page 9-26  “log()” on page 7-58
“l_stab_circle_center_radius()” on page 9-27  “log10()” on page 7-59
“l_stab_region()” on page 9-28  “lognorm_dist_inv1D()” on page 10-13
“ln()” on page 7-57  “lognorm_dist1D()” on page 10-14

M
“mag()” on page 7-60  “median()” on page 10-17
“map1_circle()” on page 9-29  “min()” on page 7-64
“map2_circle()” on page 9-30  “min_index()” on page 5-28
“max()” on page 7-61  “min_outer()” on page 7-65
“max_gain()” on page 9-31  “min2()” on page 7-66
“max_index()” on page 5-27  “mix()” on page 6-13
“max_outer()” on page 7-62  “moving_average()” on page 10-18
“max2()” on page 7-63  “mu()” on page 9-32
“mean()” on page 10-15  “mu_prime()” on page 9-33
“mean_outer()” on page 10-16

N
“norm_dist_inv1D()” on page 10-19  “ns_circle()” on page 9-34
Using Measurement Expressions in Advanced Design System

“norm_dist1D()” on page 10-20
“norms_dist_inv1D()” on page 10-22
“norms_dist1D()” on page 10-23
“ns_pwr_int()” on page 9-36
“ns_pwr_ref_bw()” on page 9-37
“num()” on page 7-67

O
“ones()” on page 7-68

P
“pae()” on page 6-15
“pdf()” on page 10-24
“peak_pwr()” on page 4-42
“peak_to_avg_pwr()” on page 4-44
“permute()” on page 5-29
“pfc()” on page 6-17
“pfc_tran()” on page 11-12
“phase()” on page 7-69
“phase_comp()” on page 9-38
“phase_gain()” on page 6-19
“phasedeg()” on page 7-70
“phaserad()” on page 7-71
“plot_vs()” on page 5-31
“polar()” on page 7-72
“pow()” on page 7-73
“power_ccdf()” on page 4-46
“power_ccdf_ref()” on page 4-48
“prod()” on page 7-74
“pspec()” on page 6-20
“pspec_tran()” on page 11-14
“pt()” on page 6-21
“pt_tran()” on page 11-13
“pwr_gain()” on page 9-39
“pwr_vs_t()” on page 4-50

R
“rad()” on page 7-75
“re()” on page 7-76
“real()” on page 7-77
“relative_noise_bw()” on page 4-51
“remove_noise()” on page 6-22
“ripple()” on page 9-40
“rms()” on page 7-78
“round()” on page 7-79

S
“s_stab_circle()” on page 9-41
“spec_power()” on page 8-18
“s_stab_circle_center_radius()” on page 9-42
“s_stab_region()” on page 9-43
“sample_delay_pi4dqpsk()” on page 4-53
“sample_delay_qpsk()” on page 4-54
“set_attr()” on page 5-33
“sfdr()” on page 6-23
“sgn()” on page 7-80
“sin()” on page 7-81
“sinc()” on page 7-82
“sinh()” on page 7-83
“size()” on page 5-34
“sm_gamma1()” on page 9-44
“sm_gamma2()” on page 9-45
“sm_y1()” on page 9-46
“sm_y2()” on page 9-47
“sm_z1()” on page 9-48
“sm_z2()” on page 9-49
“snr()” on page 6-25
“sort()” on page 5-35
“sqr()” on page 7-84
“sqrt()” on page 7-85
“stab_fact()” on page 9-50
“stab_meas()” on page 9-51
“stddev()” on page 10-25
“stddev_outer()” on page 10-26
“step()” on page 7-86
“stoabcd()” on page 9-52
“stoh()” on page 9-53
“stos()” on page 9-54
“stot()” on page 9-56
“stoy()” on page 9-57
“stoz()” on page 9-58
“sum()” on page 7-87
“sweep_dim()” on page 5-36
“sweep_size()” on page 5-37
“total_pwr()” on page 4-61
“trajectory()” on page 4-62
“transpose()” on page 7-90
“ts()” on page 6-32
“ttos()” on page 9-64
“type()” on page 5-39
Using Measurement Expressions in Advanced Design System

<table>
<thead>
<tr>
<th>Alphabetical Listing of Measurement Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>U</strong></td>
</tr>
<tr>
<td>“uniform_dist_inv1D()” on page 10-27</td>
</tr>
<tr>
<td>“uniform_dist1D()” on page 10-28</td>
</tr>
<tr>
<td>“unwrap()” on page 9-67</td>
</tr>
<tr>
<td><strong>V</strong></td>
</tr>
<tr>
<td>“v_dc()” on page 9-68</td>
</tr>
<tr>
<td>“vfc()” on page 6-35</td>
</tr>
<tr>
<td>“vfc_tran()” on page 11-16</td>
</tr>
<tr>
<td>“volt_gain()” on page 9-69</td>
</tr>
<tr>
<td>“volt_gain_max()” on page 9-71</td>
</tr>
<tr>
<td>“vs()” on page 5-40</td>
</tr>
<tr>
<td><strong>W</strong></td>
</tr>
<tr>
<td>“what()” on page 5-41</td>
</tr>
<tr>
<td>“write_snp()” on page 9-73</td>
</tr>
<tr>
<td>“wtodbm()” on page 7-91</td>
</tr>
<tr>
<td><strong>X</strong></td>
</tr>
<tr>
<td>“xor()” on page 7-92</td>
</tr>
<tr>
<td><strong>Y</strong></td>
</tr>
<tr>
<td>“yield_sens()” on page 10-29</td>
</tr>
<tr>
<td>“yin()” on page 9-75</td>
</tr>
<tr>
<td>“yopt()” on page 9-76</td>
</tr>
<tr>
<td>“ytoabcd()” on page 9-77</td>
</tr>
<tr>
<td><strong>Z</strong></td>
</tr>
<tr>
<td>“zeros()” on page 7-93</td>
</tr>
<tr>
<td>“zin()” on page 9-81</td>
</tr>
<tr>
<td><strong>-</strong></td>
</tr>
</tbody>
</table>

2-18       Alphabetical Listing of Measurement Functions
“zopt()” on page 9-82
“ztoabcd()” on page 9-83
“ztoy()” on page 9-86
Using Measurement Expressions in Advanced Design System
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

bud_freq() on page 3-4  bud_gain() on page 3-7
bud_gain_comp() on page 3-10  bud_pwr() on page 3-23
bud_gamma() on page 3-13  bud_pwr_inc() on page 3-25
bud_ip3_deg() on page 3-15  bud_pwr_refl() on page 3-27
bud_nf() on page 3-17  bud_snr() on page 3-29
bud_nf_deg() on page 3-19  bud_tn() on page 3-31
bud_vswr() on page 3-33
Circuit Budget Functions

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 Using Circuit Simulators 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: $S_{12} = S_{21} = S_{31} = 1$, and all other $S_{ij} = 0$. 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.
Circuit Budget Functions

**bud_freq()**

Returns the frequency plan of a network

**Syntax**

\[
y = \text{bud}_\text{freq}(\text{freqIn}, \text{pinNumber}, \text{"simName"})\text{ for AC analysis or } y = \text{bud}_\text{freq}(\text{planNumber}, \text{pinNumber})\text{ for HB analysis}
\]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>freqIn</td>
<td>input source frequency</td>
<td>((0, \infty))</td>
<td>real</td>
<td>no</td>
</tr>
<tr>
<td>planNumber</td>
<td>represents the chosen frequency plan and is required when using the \text{bud}_\text{freq}() function with HB data.</td>
<td>([1, \infty))</td>
<td>integer</td>
<td>yes</td>
</tr>
<tr>
<td>pinNumber</td>
<td>used to choose which pins of each network element are referenced †</td>
<td>([1, \infty))</td>
<td>integer</td>
<td>no</td>
</tr>
<tr>
<td>simName</td>
<td>simulation instance name, such as &quot;AC1&quot; or &quot;HB1&quot;, used to qualify the data when multiple simulations are performed.</td>
<td>string</td>
<td></td>
<td>no</td>
</tr>
</tbody>
</table>

† 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**

\[
x = \text{bud}_\text{freq}()
\]

returns frequency plan for AC analysis

\[
x = \text{bud}_\text{freq}(1\text{MHz})
\]

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

3-4  bud_freq()
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
Circuit Budget Functions

variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.
**bud_gain()**

Returns budget transducer-power gain

**Syntax**

\[
y = \text{bud_gain}(vIn, iIn, Zs, Plan, pinNumber, \text{"simName"}) \text{ or } y = \text{bud_gain}(\text{"SourceName"}, SrcIndx, Zs, Plan, \text{budgetPath})
\]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vIn</td>
<td>voltage flowing into the input port</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>iIn</td>
<td>current flowing into the input port</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>SourceName</td>
<td>component name at the input port</td>
<td></td>
<td>string</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>SrcIndx</td>
<td>frequency index that corresponds to the source frequency to determine which frequency to use from a multitone source as the reference signal</td>
<td>[1, \infty)</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>Zs</td>
<td>input source port impedance</td>
<td>[0, \infty)</td>
<td>real</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>Plan</td>
<td>number of the selected frequency plan(needed only for HB)</td>
<td></td>
<td>string</td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>pinNumber</td>
<td>Used to choose which pins of each network element are referenced ‡</td>
<td>[1, \infty)</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>budgetPath</td>
<td>Selects the budget path, specified as an array eg. [&quot;PORT1.t1&quot;,&quot;Tee1.t3&quot;,&quot;Term3.t1&quot;]</td>
<td></td>
<td>string array</td>
<td></td>
<td>no</td>
</tr>
</tbody>
</table>
Circuit Budget Functions

Examples

\[
x = \text{bud\_gain}(\text{PORT1.t1.v}, \text{PORT1.t1.i})
\]\n
or

\[
x = \text{bud\_gain}(\text{"PORT1"})
\]\n
\[
y = \text{bud\_gain}(\text{PORT1.t1.v}, \text{PORT1.t1.i}, 75)
\]\n
or

\[
y = \text{bud\_gain}(\text{"PORT1"}, , 75., 1)
\]\n
\[
z = \text{bud\_gain}(\text{PORT1.t1.v}[3], \text{PORT1.t1.i}[3], , 1)
\]\n
or

\[
z = \text{bud\_gain}(\text{"PORT1"}, 3, , 1)
\]

Defined in

\text{$HPEESOF\_DIR/expressions/ael/budget\_fun.ael$}

See Also

\text{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 \text{vIn} and \text{iIn} are passed directly, one may want to use the index of the frequency sweep explicitly to reference the input source frequency.

† 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.
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), must be entered as the budgetPath argument. Use the alternate syntax for budget path measurements as shown below.

bud_gain("SourceName", SrcIndx, Zs, Plan, budgetPath)
Circuit Budget Functions

bud_gain_comp()

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

Syntax

\[ y = \text{bud\_gain\_comp}(vI, iI, Zs, Plan, freqIndex, pinNumber, "simName") \]  

or \[ y = \text{bud\_gain\_comp}("SourceName", SrcIndx, Zs, Plan, freqIndex, pinNumber, "simName") \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vI</td>
<td>voltage flowing into the input port</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>iI</td>
<td>current flowing into the input port</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>SourceName</td>
<td>component name at the input port</td>
<td></td>
<td>string</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>SrcIndx</td>
<td>frequency index that corresponds to the</td>
<td>([1, \infty))</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>source frequency to determine which</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>frequency to use from a multitone source</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>as the reference signal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zs</td>
<td>input source port impedance</td>
<td>([0, \infty))</td>
<td>real</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>freqIndex</td>
<td>index of harmonic frequency</td>
<td>((-\infty, \infty))</td>
<td>integer</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Plan</td>
<td>number of the selected frequency plan(needed only for HB)</td>
<td>string</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pinNumber</td>
<td>Used to choose which pins of each network element are referenced</td>
<td>([1, \infty))</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
</tbody>
</table>
Examples

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

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

Defined in

\$\text{HPESOF_DIR/expressions/ael/budget_fun.ael}\$

See Also

\text{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.
Circuit Budget Functions

**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**

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

**Arguments**

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

† 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**

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

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

**Defined in**

$HPEESOF_DIR/expressions/ael/budget_fun.ael
Circuit Budget Functions

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

\[ y = \text{bud_ip3_deg}(vOut, \text{LinearizedElement}, \text{fundFreq}, \text{imFreq}, \text{zRef}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vOut</td>
<td>signal voltage at the output</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>LinearizedElement</td>
<td>variable containing the names of the linearized components</td>
<td>string</td>
<td></td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>fundFreq</td>
<td>harmonic frequency indices for the fundamental frequency</td>
<td>(-\infty, \infty)</td>
<td>integer</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>imFreq</td>
<td>harmonic frequency indices for the intermodulation frequency</td>
<td>(-\infty, \infty)</td>
<td>integer</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>Zref</td>
<td>input source port impedance</td>
<td>[0, \infty)</td>
<td>real</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

Examples

\[ y = \text{bud_ip3_deg}(vOut, \text{LinearizedElement}, \{1, 0\}, \{2, -1\}) \]
returns the budget third-order intercept point degradation

Defined in

$HPEESOF\_DIR/expressions/acl/budget\_fun.acl$

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.
Circuit Budget Functions

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

Arguments

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

† 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")
In an AC analysis, `bud_nf()` can be used as follows:

```
BudNF1 = bud_nf("PORT1")
BudNF2 = bud_nf("PORT1", 50.0, 1 Hz, 2, "AC1")
BudNF3 = bud_nf("PORT1", ..., budget_path)
BudNF4 = bud_nf(,,, budget_path)
```

where `budget_path` can be defined as:

```
budget_path = ["PORT1.t1","b2_AMP1.t2","b3_MIX1.t2","b4_AMP2.t2",
              "b5_BPF2.t2","b6.t1"]
```

**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()
```
bud_nf_deg()

Returns budget noise figure degradation

Syntax

\[ y = \text{bud_nf_deg}(vIn, iIn, vOut, iOut, vOut.NC.vnc, vOut.NC.name, Zs, BW) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vIn</td>
<td>voltage flowing into the input port</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>iIn</td>
<td>current flowing into the input port</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>vOut</td>
<td>voltage flowing into the output port</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>iOut</td>
<td>current flowing into the output port</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>vOut.NC.vnc</td>
<td>noise contributions at the output port</td>
<td>string</td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>vOut.NC.name</td>
<td>noise contributions component names at the output port</td>
<td>string</td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Zs</td>
<td>input source port impedance</td>
<td>[0, \infty)</td>
<td>real</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>BW †</td>
<td>bandwidth</td>
<td>[1, \infty)</td>
<td>real</td>
<td>1</td>
<td>no</td>
</tr>
</tbody>
</table>

† BW must be set as the value of Bandwidth used on the noise page of the AC controller

Examples

\[ x = \text{bud_nf_deg}(\text{PORT1.t1.v}, \text{PORT1.t1.i}, \text{Term1.t1.v}, \text{Term1.t1.i}, \text{vOut.NC.vnc}, \text{vOut.NC.name}, \text{Zs}, \text{BW}) \]

\[ x = \text{bud_nf_deg}(\text{"PORT1"}, \text{"Term1"}, \text{"vOut"}) \]

Defined in

$\text{HP EESOF_DIR/expressions/ael/budget_fun.ael}$

See Also

bud_nf(), bud_tn()
Circuit Budget Functions

**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**

\[ y = \text{bud\_noise\_pwr}(Z\text{ref}, \text{Plan}, \text{pinNumber}, \text{"simName"}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zref</td>
<td>input source port impedance</td>
<td>([0, \infty))</td>
<td>real</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>Plan</td>
<td>number of the selected frequency plan(needed only for HB)</td>
<td>string</td>
<td>string</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>pinNumber</td>
<td>Used to choose which pins of each network element are referenced†</td>
<td>([1, \infty))</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>simName</td>
<td>simulation instance name, such as &quot;AC1&quot; or &quot;HB1&quot;, used to qualify the data when multiple simulations are performed.</td>
<td>string</td>
<td>string</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

† 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**

\[ x = \text{bud\_noise\_pwr}() \]

returns the noise power at all frequencies

\[ y = \text{bud\_noise\_pwr}(75, 1) \]

returns the noise power at reference frequencies in plan 1

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/budget\_fun.ael}$

**See Also**

bud_pwr()
Circuit Budget Functions

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 \( Z_{\text{ref}} \) 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.

\textbf{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

\[ y = \text{bud}_pwr(\text{Plan}, \text{pinNumber}, \text{"simName"}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>number of the selected frequency plan (needed only for HB)</td>
<td>[1, ∞)</td>
<td>string</td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>pinNumber</td>
<td>Used to choose which pins of each network element are referenced †</td>
<td>[1, ∞)</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>simName</td>
<td>simulation instance name, such as &quot;AC1&quot; or &quot;HB1&quot;, used to qualify the data when multiple simulations are performed.</td>
<td>[1, ∞)</td>
<td>string</td>
<td></td>
<td>no</td>
</tr>
</tbody>
</table>

† 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

\[ x = \text{bud}_pwr() \]
returns the signal power at all frequencies when used in AC or HB simulations

\[ y = \text{bud}_pwr(50, 1) \]
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()
Circuit Budget Functions

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**

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

† 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()
Circuit Budget Functions

**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**

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

**Arguments**

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

† 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**

```matlab
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()`
Circuit Budget Functions

**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_sntr()**

Returns the budget signal-to-noise-power ratio

**Syntax**

\[ y = \text{bud_sntr}(\text{Plan}, \text{pinNumber}, \text{"simName"}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>number of the selected frequency plan (needed only for HB)</td>
<td></td>
<td>string</td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>pinNumber</td>
<td>Used to choose which pins of each network element are referenced †</td>
<td>[1, ∞)</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>simName</td>
<td>simulation instance name, such as &quot;AC1&quot; or &quot;HB1&quot;, used to qualify the data when multiple simulations are performed.</td>
<td></td>
<td>string</td>
<td></td>
<td>no</td>
</tr>
</tbody>
</table>

† 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**

\[ x = \text{bud_sntr()} \]

returns the SNR at all frequencies

\[ y = \text{bud_sntr}(1) \]

returns the SNR at reference frequencies in plan 1

**Defined in**

$\text{HPEESOF_DIR/expressions/ael/budget_fun.ael}$

**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
Circuit Budget Functions

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

\[ y = \text{bud}_\text{tn}(\text{vIn}, \text{iIn}, \text{noisevIn}, \text{Zs}, \text{BW}, \text{pinNumber}, \text{"simName"}) \text{ or } y = \text{bud}_\text{tn}(\text{"SourceName"}) \]

Arguments

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

† 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

\[ x = \text{bud}_\text{tn}(\text{PORT1}\text{.t1}.\text{v}, \text{PORT1}\text{.t1}.\text{i}, \text{PORT1}\text{.t1}.\text{v}.\text{noise}) \]

\[ x = \text{bud}_\text{tn}(\text{"PORT1"}) \]
Circuit Budget Functions

Defined in
$\text{HPEE\_SOF\_DIR/expressions/ael/budget\_fun.ael}$

See Also
bud_nf(), bud_nf_deg()

Notes/Equations
Used in AC simulation

This is an equivalent output-noise temperature (in 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

\[
y = \text{bud_vswr}(Z\text{ref}, \text{Plan}, \text{pinNumber}, "\text{simName}")
\]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zref</td>
<td>input source port impedance</td>
<td>( [0, \infty) )</td>
<td>real</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>Plan</td>
<td>number of the selected frequency plan (needed only for HB)</td>
<td></td>
<td>string</td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>pinNumber</td>
<td>Used to choose which pins of each network element are referenced †</td>
<td>( [1, \infty) )</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>simName</td>
<td>simulation instance name, such as &quot;AC1&quot; or &quot;HB1&quot;, used to qualify the data when multiple simulations are performed.</td>
<td></td>
<td>string</td>
<td></td>
<td>no</td>
</tr>
</tbody>
</table>

† 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

\[
x = \text{bud_vswr}()
\]

returns the vswr at all frequencies

\[
y = \text{bud_vswr}(75, 1)
\]

returns the vswr at reference frequencies in plan 1

Defined in

$HPEESOF\_DIR/expressions/ael/budget\_fun.ael$

See Also

bud_gamma()
Circuit Budget Functions

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.

A,B,C,D
“acpr_vi()” on page 4-3
“acpr_vr()” on page 4-5
“ber_pi4dqpsk()” on page 8-3
“ber_qpsk()” on page 8-5
“channel_power_vi()” on page 4-8
“channel_power_vr()” on page 4-10
“const_evm()” on page 4-12
“constellation()” on page 11-2
“cross_hist()” on page 4-15
“delay_path()” on page 4-17

E,F,P,R,S,T
“evm_wlan_dsss_cck_pbcc()” on page 4-18
“evm_wlan_ofdm()” on page 4-28
“eye()” on page 8-7
“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
“fs()” on page 4-37
“peak_pwr()” on page 4-42
“peak_to_avg_pwr()” on page 4-44
“power_ccdf()” on page 4-46
“power_ccdf_ref()” on page 4-48
“pwr_vs_t()” on page 4-50
“relative_noise_bw()” on page 4-51
“sample_delay_pi4dqpsk()” on page 4-53
“sample_delay_qpsk()” on page 4-54
“spectrum_analyzer()” on page 4-55
“total_pwr()” on page 4-61
“trajectory()” on page 4-62
“ts()” on page 6-32

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.
Circuit Envelope Functions

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

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

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

$HPE ESOF_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**

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

**Arguments**

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

Examples

Vmain_fund=vOut[1]
mainlimits={-16.4 kHz,16.4 kHz}
UpChlimits={mainlimits+30 kHz}
LoChlimits={mainlimits-30 kHz}
acpr_ey=acpr_vr(Vmain_fund,50,mainlimits,UpChlimits,LoChlimits)

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

$HPE/SOF_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.

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>winConst</td>
<td>window constant that affects the shape of the applied window.</td>
<td>[0, ∞)</td>
<td>real</td>
<td>0.75</td>
<td>no</td>
</tr>
</tbody>
</table>

† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"
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.
Circuit Envelope Functions

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

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

† 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")

4-8 channel_power_vi()
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.

\[
\text{Main\_Channel\_Power\_dBm} = 10 \times \log(\text{Main\_Channel\_Power}) + 30
\]

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

Defined in

$\text{HPE\_ESOF\_DIR/expressions/ael/digital\_wireless\_fun.ael}$

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.
Circuit Envelope Functions

**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**

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

† 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**

```python
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.

\[
\text{Power in dBm} = 10 \log_{10}(\text{Power in Watts})
\]

4-10  channel_power_vr()
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/acl/digital_wireless_fun.acl

**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.
Circuit Envelope Functions

**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**

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

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vfund_ideal</td>
<td>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().</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>vfund_dist</td>
<td>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()</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>symbol_rate</td>
<td>symbol rate of the modulation signal</td>
<td>[0, ∞)</td>
<td>integer, real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>sampling_delay</td>
<td>sampling delay †</td>
<td>[0, ∞)</td>
<td>integer, real</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>rotation</td>
<td>parameter that rotates the constellations by that many radian ‡</td>
<td>[0, ∞)</td>
<td>integer, real</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>
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.
Circuit Envelope Functions

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

\[
\begin{align*}
data[0] &= \text{ideal constellation} \\
data[1] &= \text{ideal trajectory} \\
data[2] &= \text{distorted constellation} \\
data[3] &= \text{distorted trajectory} \\
data[4] &= \text{error vector magnitude versus time} \\
data[5] &= \text{percent error vector magnitude}
\end{align*}
\]

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

Defined in

\$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

collection(), 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

\[
y = \text{cross_hist}(\text{Vout\_time}, \text{time\_start}, \text{time\_stop}, \text{level\_low}, \text{level\_high}, \text{number\_of\_bins}, \text{BitRate}, \text{No\_of\_Eye}, \text{Delay}, \text{steps})
\]

Arguments

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay</td>
<td>used to remove initial transient in the eye diagram and is expressed in time units</td>
<td>$[0, \infty)$</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>steps</td>
<td>represents the number of sampling points between level_low and level_high and is used for controlling the density of jitter histogram.</td>
<td>$[1, \infty)$</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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**

```plaintext
y = delay_path(vin, vout)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vin</td>
<td>envelope ((I + j \cdot Q)) at the input node</td>
<td>((−\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>vout</td>
<td>((I + j \cdot Q)) at the output node</td>
<td>((−\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
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`-constellation and `vout`-constellation.

or

```plaintext
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.
Circuit Envelope Functions

**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**

```matlab
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltage</td>
<td>complex envelope WLAN DSSS/CCK/PBCC voltage signal</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>mirrorSpectrum</td>
<td>specifies whether the input signal should be mirrored or not</td>
<td>See Notes</td>
<td>string</td>
<td>NO</td>
<td>no</td>
</tr>
<tr>
<td>start</td>
<td>specifies the start time for the EVM analysis</td>
<td>[0, max(indep(voltage))]</td>
<td>real</td>
<td>first point of the input data</td>
<td>no</td>
</tr>
<tr>
<td>averageType</td>
<td>specifies what type of averaging is done</td>
<td>See Notes</td>
<td>string</td>
<td>OFF</td>
<td>no</td>
</tr>
<tr>
<td>burstsToAverage</td>
<td>number of bursts over which the results will be averaged</td>
<td>(0, ∞)</td>
<td>integer</td>
<td>20</td>
<td>no</td>
</tr>
<tr>
<td>modulationFormat</td>
<td>modulation format</td>
<td>See Notes</td>
<td>string</td>
<td>&quot;Auto Detect&quot;</td>
<td>no</td>
</tr>
<tr>
<td>searchTime</td>
<td>search time</td>
<td>[0, max(indep(voltage))]</td>
<td>real</td>
<td>550 usec</td>
<td>no</td>
</tr>
<tr>
<td>resultLengthType</td>
<td>specifies how the result length is determined</td>
<td>See Notes</td>
<td>string</td>
<td>&quot;Auto Select&quot;</td>
<td>no</td>
</tr>
<tr>
<td>resultLength</td>
<td>result length in chips</td>
<td>[1, 108344]</td>
<td>integer</td>
<td>2816</td>
<td>no</td>
</tr>
<tr>
<td>measurementOffset</td>
<td>measurement offset in chips</td>
<td>(0, ∞)</td>
<td>integer</td>
<td>22</td>
<td>no</td>
</tr>
</tbody>
</table>
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, , , , , , , , , , “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>
<thead>
<tr>
<th>Measurement Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg_WLAN_80211b_1000_chip_Pk_EVM_pct</td>
<td>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</td>
</tr>
<tr>
<td>WLAN_80211b_1000_chip_Pk_EVM_pct</td>
<td>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</td>
</tr>
<tr>
<td>Avg_EVMrms_pct</td>
<td>average EVM rms in percentage as defined in the Agilent 89600 VSA</td>
</tr>
<tr>
<td>EVMrms_pct</td>
<td>EVM rms in percentage as defined in the Agilent 89600 VSA versus burst</td>
</tr>
<tr>
<td>EVM_Pk_pct</td>
<td>peak EVM in percentage versus burst</td>
</tr>
<tr>
<td>EVM_Pk_chip_idx</td>
<td>peak EVM chip index versus burst</td>
</tr>
</tbody>
</table>
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

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

Table 4-1. Available Measurement Results for evm_wlan_dsssdss_cck_pbcc()
Circuit Envelope Functions

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.
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

<table>
<thead>
<tr>
<th>resultLengthType</th>
<th>resultLength</th>
<th>Measurement Result Length Actually Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Select</td>
<td>2200</td>
<td>2200</td>
</tr>
<tr>
<td>Auto Select</td>
<td>2816</td>
<td>2816</td>
</tr>
<tr>
<td>Auto Select</td>
<td>3300</td>
<td>2816</td>
</tr>
<tr>
<td>Manual Override</td>
<td>2200</td>
<td>2200</td>
</tr>
<tr>
<td>Manual Override</td>
<td>2816</td>
<td>2816</td>
</tr>
<tr>
<td>Manual Override</td>
<td>3300</td>
<td>3300</td>
</tr>
</tbody>
</table>

Table 4-2. Measurement result length setting
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

<table>
<thead>
<tr>
<th>WLAN_802_11b Parameter</th>
<th><code>evm_wlan_dsss_cck_pbcc()</code> Parameter</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataRate (default is 11 Mbps)</td>
<td>searchTime (default is 550 µsec)</td>
<td>The recommended searchTime is 2 × BurstLength.</td>
</tr>
<tr>
<td>PreambleFormat (default is Long)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PwrRamp (default is None)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IdleInterval (default is 10 µsec)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataLength (default is 100)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MirrorSpectrum (default is NO)</td>
<td>mirrorSpectrum (default is NO)</td>
<td>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.</td>
</tr>
<tr>
<td>FilterType (default is Gaussian)</td>
<td>referenceFilter (default is Gaussian)</td>
<td>When FilterType is set to “Gaussian”, the recommended setting of referenceFilter is “Gaussian”; otherwise, the recommended setting is “Rectangular”</td>
</tr>
<tr>
<td>GaussianFilter_bT (default is 0.3)</td>
<td>referenceFilterBT (default is 0.3)</td>
<td>The recommended setting of referenceFilterBT is the same value as GaussianFilter_bT. This parameter is only used when referenceFilter is set to “Gaussian”.</td>
</tr>
</tbody>
</table>

BurstLength = tRamp + tPLCP + tPSDU + IdleInterval, where,

\[
tRamp = \begin{cases} 
0, & \text{PwrRamp = None} \\
4 \mu\text{sec}, & \text{otherwise} 
\end{cases}
\]

\[
tPLCP = \begin{cases} 
192 \mu\text{sec}, & \text{PreambleFormat = Long} \\
96 \mu\text{sec}, & \text{PreambleFormat = Short} 
\end{cases}
\]

IPS DU = DataLength × 8 / DataRate
Circuit Envelope Functions

**evm\_wlan\_ofdm()**

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

**Syntax**

```plaintext
evm = evm\_wlan\_ofdm(voltage{, mirrorSpectrum, start, averageType, burstsToAverage, subcarrierModulation, guardInterval, searchTime, resultLengthType, resultLength, measurementOffset, measurementInterval, subcarrierSpacing, symbolTimingAdjust, sync, output})
```

**Arguments**

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

4-28  evm\_wlan\_ofdm()
Examples

devRMS = 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.

devOffset = 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

ev_mlan_ofdm()
Circuit Envelope Functions

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()

<table>
<thead>
<tr>
<th>Measurement Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVMrms_percent</td>
<td>average EVM rms in percentage</td>
</tr>
<tr>
<td>EVM_db</td>
<td>average EVM in dB</td>
</tr>
<tr>
<td>PilotEVM_db</td>
<td>average pilot EVM in dB</td>
</tr>
<tr>
<td>CPErrms_percent</td>
<td>average Common Pilot Error rms in percentage</td>
</tr>
<tr>
<td>IQ_Offset_db</td>
<td>average IQ offset in dB</td>
</tr>
<tr>
<td>SyncCorrelation</td>
<td>average sync correlation</td>
</tr>
</tbody>
</table>

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.
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 sets 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>
<thead>
<tr>
<th>resultLengthType</th>
<th>resultLength</th>
<th>Measurement Result Length Actually Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Select</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Auto Select</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>Auto Select</td>
<td>30</td>
<td>26</td>
</tr>
<tr>
<td>Manual Override</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Manual Override</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>Manual Override</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

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.
Circuit Envelope Functions

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.](image)

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 not to 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.

\[
T = T_{GI} + T_{FFT}
\]

\[
T_{GI} = \text{Guard Interval}
\]

\[
T_{FFT} = \text{FFT/IFFT Time Period}
\]

\[
T_{TR} = \text{Symbol Transition Time}
\]

\[
T_{STA} = \text{Symbol Timing Adjust (\%)}
\]

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>
<thead>
<tr>
<th>WLAN_802_11a</th>
<th>evm_wlan_ofdm()</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataRate (default is 54 Mbps)</td>
<td>searchTime (default is 80 µsec)</td>
<td>The recommended searchTime is $2 \times \text{BurstLength}$.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BurstLength$= (5 + (1 + \text{OFDSymbolsPerBurst}) \times (1 + \text{GuardInterval})) \times 64 / \text{Bandwidth} + \text{IdleInterval}$, where, OFDSymbolsPerBurst$= \text{ceil}[(22 + 8 \times \text{DataLength}) \times 250000 / \text{DataRate}]$</td>
</tr>
<tr>
<td>Bandwidth (default is 20 MHz)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IdleInterval (default is 4 µsec)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataLength (default is 100)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GuardInterval (default is 0.25)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MirrorSpectrum (default is NO)</td>
<td>mirrorSpectrum (default is NO)</td>
<td>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.</td>
</tr>
<tr>
<td>Bandwidth (default is 20 MHz)</td>
<td>subcarrierSpacing (default is 312.5 kHz)</td>
<td>subcarrierSpacing must be set to Bandwidth/64.</td>
</tr>
<tr>
<td>GuardInterval (default is 0.25)</td>
<td>guardInterval (default is 0.25)</td>
<td>guardInterval must be set to the same value as GuardInterval.</td>
</tr>
</tbody>
</table>
fs()
Performs a time-to-frequency transform

Syntax
\[ y = \text{fs}(x, \text{fstart}, \text{fstop}, \text{numfreqs}, \text{dim}, \text{windowType}, \text{windowConst}, \text{tstart}, \text{tstop}, \text{interpOrder}, \text{transformMethod}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Time-domain data to be transformed</td>
<td>(-(\infty, \infty))</td>
<td>real</td>
<td>0 (\dagger)</td>
<td>yes</td>
</tr>
<tr>
<td>fstart</td>
<td>Starting frequency</td>
<td>[0, (\infty)]</td>
<td>real</td>
<td>0 (\dagger)</td>
<td>no</td>
</tr>
<tr>
<td>fstop</td>
<td>Stopping frequency</td>
<td>[0, (\infty)]</td>
<td>real</td>
<td>1/(2*newdeltat)</td>
<td>no</td>
</tr>
<tr>
<td>numfreqs</td>
<td>Number of frequencies</td>
<td>[1, (\infty)]</td>
<td>integer</td>
<td>(fstop-fstart)*(tstop-tstart)+1</td>
<td>no</td>
</tr>
<tr>
<td>dim</td>
<td>Dimension to be transformed (not used currently)</td>
<td>[1, (\infty)]</td>
<td>integer</td>
<td>highest dimension</td>
<td>no</td>
</tr>
<tr>
<td>windowType</td>
<td>Type of window to be applied to the data</td>
<td>[0, 9] (\dagger)</td>
<td>integer, string</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>windowConst</td>
<td>Window constant (\dagger) (\dagger)</td>
<td>[0, (\infty)]</td>
<td>integer, real</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>tstart</td>
<td>Start time (\dagger) (\dagger)</td>
<td>[0, (\infty)]</td>
<td>integer, real</td>
<td>first time point in given data</td>
<td>no</td>
</tr>
<tr>
<td>tstop</td>
<td>Stop time (\dagger) (\dagger)</td>
<td>[0, (\infty)]</td>
<td>integer, real</td>
<td>last time point in given data</td>
<td>no</td>
</tr>
<tr>
<td>interpOrder</td>
<td>Interpolation Order</td>
<td>[1, 3] (\dagger) (\dagger)</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
</tbody>
</table>
Circuit Envelope Functions

### 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:

\[
y = \text{fs}(\text{vOut})
\]

returns the spectrum \((0, 0.2\text{GHz}, \ldots, 17.6\text{GHz})\), evaluated from 0 to 5 ns.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>transformMethod</td>
<td>Transformation method</td>
<td>[1, 3] ‡ ‡ ‡ integer</td>
<td>1</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

† If data is real, \(f_{\text{start}} = 0, f_{\text{stop}} = 1/(2*\text{newdeltat})\). If data is complex, \(f_{\text{start}} = -1/(\text{time}[\text{startIndex}+2] - \text{time}[\text{stopIndex}])\) and \(f_{\text{stop}} = 1/(\text{time}[\text{startIndex}+2] - \text{time}[\text{stopIndex}])\). 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
\[ y = fs(vOut, 0, 10GHz) \]
returns the spectrum \((0, 0.2GHz, \ldots, 10.0GHz)\), evaluated from 0 to 5 ns.

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

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

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

\[ y = fs(vOut, 0, 10GHz, 11, , "Blackman") \]
returns the spectrum \((0, 1.0GHz, \ldots, 10.0GHz)\), 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 $\Delta F$ equals 0. Where $\Delta F = (F_{stop} - F_{start})/(\text{numFREQs})$.

2. When $1.0/\Delta F > T_{stop} - T_{start}$. In this case there are not enough time data for requested frequency resolution.

3. When $2 \times \text{numFREQs} > \text{numDataPts}$. In this case there are not enough data points for frequencies.
Circuit Envelope Functions

**peak_pwr()**

Returns the peak power of the input voltage data

**Syntax**

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

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltage</td>
<td>baseband or complex envelope voltage signal</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>refR</td>
<td>reference resistance in Ohms</td>
<td>(0, ∞)</td>
<td>real</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>percent</td>
<td>percentage of time the returned power value is exceeded</td>
<td>[0, 100]</td>
<td>real</td>
<td>0.0</td>
<td>no</td>
</tr>
<tr>
<td>unit</td>
<td>power unit to be used</td>
<td>&quot;W&quot;</td>
<td>&quot;dBm&quot;</td>
<td>&quot;dBW&quot;</td>
<td>string</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
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.

```plaintext
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 (\(-\ln(0.05) / 0.032^2 / 50\)).

**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.
Circuit Envelope Functions

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltage</td>
<td>baseband or complex envelope voltage signal</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>percent</td>
<td>percentage of time the returned power value is exceeded</td>
<td>[0, 100]</td>
<td>real</td>
<td>0.0</td>
<td>no</td>
</tr>
<tr>
<td>unit</td>
<td>unit to be used</td>
<td>&quot;dB&quot;</td>
<td>&quot;ratio&quot;</td>
<td>&quot;dB&quot;</td>
<td>no</td>
</tr>
</tbody>
</table>

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.
Circuit Envelope Functions

**power_ccdf()**

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

**Syntax**

```plaintext
pCCDF = power_ccdf(data, numBins)
```

**Arguments**

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

**Examples**

```plaintext
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.

```plaintext
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.
Circuit Envelope Functions

**power_ccdf_ref()**

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

**Syntax**

```
ccdfRef = power_ccdf_ref(indepPowerRatioValues)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>indepPowerRatioValues</td>
<td>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</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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
Circuit Envelope Functions

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltage</td>
<td>baseband or complex envelope voltage signal</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>refR</td>
<td>reference resistance in Ohms</td>
<td>(0, ∞)</td>
<td>real</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>unit</td>
<td>power unit to be used</td>
<td>string</td>
<td>“W”</td>
<td>“dBm”</td>
<td>“dBW”</td>
</tr>
</tbody>
</table>

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**

```plaintext
y = relative_noise_bw(winType, winConst)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>winType</td>
<td>window type</td>
<td>†</td>
<td>string</td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>winConst</td>
<td>window constant that affects the shape of the applied window.</td>
<td>[0, ∞)</td>
<td>real</td>
<td>0.75</td>
<td>no</td>
</tr>
</tbody>
</table>

† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"

**Examples**

```plaintext
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
Circuit Envelope Functions

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

Defined in
$HPE E SOF_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

\[
y = \text{sample_delay_pi4dqpsk}(\text{vlQ}, \text{symbolRate}, \text{path_delay}, \text{timeResolution})
\]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vlQ</td>
<td>complex envelope ((I + jQ)) of a pi/4 DQPSK signal</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>symbolRate</td>
<td>symbol rate of the pi/4 DQPSK signal</td>
<td>((0, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>path_delay</td>
<td>time delay on the waveform before the sampling starts †</td>
<td>((0, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>timeResolution</td>
<td>time step (typically one-tenth of a symbol time or less) used to search for the best sampling point in a given symbol period</td>
<td>((0, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

† 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

\[
a = \text{sample_delay_pi4dqpsk}(\text{vout}[1], 25e3, 1.5e-6, 0.15e-6)
\]

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vlQ</td>
<td>complex envelope ((I + jQ)) of a QPSK signal</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>symbolRate</td>
<td>symbol rate of the QPSK signal</td>
<td>((0, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>path_delay</td>
<td>time delay on the waveform before the sampling starts †</td>
<td>((0, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>timeResolution</td>
<td>time step (typically one-tenth of a symbol time or less) used to search for the best sampling point in a given symbol period</td>
<td>((0, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

† 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

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

† 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 \([f\text{Carrier}-0.5/T\text{Step}, f\text{Carrier}+0.5/T\text{Step}]\)
- for baseband signals \([0, 0.5/T\text{Step}]\)

where \(T\text{Step}\) 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:

  \[
  w(kT_s) = \begin{cases} 
  1.0 & 0 \leq k \leq N \\
  0.0 & \text{otherwise}
  \end{cases}
  \]

  where \(N\) is the window size

- Hamming 0.54:

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

  where \(N\) is the window size

- Hanning 0.5:
Circuit Envelope Functions

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

where \(N\) is the window size

• **Gaussian 0.75:**

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

where \(N\) is the window size

• **Kaiser 7.865:**

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

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

• **8510 6.0 (Kaiser 6.0):**

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

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

• **Blackman:**

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

where \(N\) is the window size
• Blackman-Harris:

\[
w(kT) = \begin{cases} 
0.35875 - 0.48829 \cos\left(\frac{2\pi k}{N}\right) + 0.14128 \cos\left(\frac{4\pi k}{N}\right) - 0.01168 \cos\left(\frac{6\pi k}{N}\right) & 0 \leq k \leq N \\
0.0 & \text{otherwise}
\end{cases}
\]

where \( N \) is the window size

The \( \text{resBW} \) 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 \( \text{resBW} \) 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 \( \text{resBW} \) 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 \( \text{resBW} \). 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.

![Input signal segments](image)

**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.

\[ \text{ResBW} = \text{ENBW} = \frac{\text{NENBW}}{T} \]
Circuit Envelope Functions

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.

<table>
<thead>
<tr>
<th>Window</th>
<th>NENBW</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>1.0</td>
</tr>
<tr>
<td>Hamming 0.54</td>
<td>1.363</td>
</tr>
<tr>
<td>Hanning 0.5</td>
<td>1.5</td>
</tr>
<tr>
<td>Gaussian 0.75</td>
<td>1.883</td>
</tr>
<tr>
<td>Kaiser 7.865</td>
<td>1.653</td>
</tr>
<tr>
<td>8510 6.0</td>
<td>1.467</td>
</tr>
<tr>
<td>Blackman</td>
<td>1.727</td>
</tr>
<tr>
<td>Blackman-Harris</td>
<td>2.021</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltage</td>
<td>baseband or complex envelope voltage signal</td>
<td>$(\infty, \infty)$</td>
<td>integer, real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>refR</td>
<td>reference resistance in Ohms</td>
<td>$(0, \infty)$</td>
<td>real</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>unit</td>
<td>power unit to be used</td>
<td>&quot;W&quot;</td>
<td>&quot;dBm&quot;</td>
<td>&quot;dBW&quot;</td>
<td>string</td>
</tr>
</tbody>
</table>

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 (0.032 / 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.
Circuit Envelope Functions

**trajectory()**

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

**Syntax**

```matlab
Traj = trajectory(i_data, q_data)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_data</td>
<td>in-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>q_data</td>
<td>quadrature-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

† This could be a baseband signal instead, but in either case it must be real valued versus time

**Examples**

```matlab
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.
Circuit Envelope Functions
Chapter 5: Data Access Functions

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

B, C, E, F, G, I
“build_subrange()” on page 5-2 “delete()” on page 5-16
“chop()” on page 5-3 “expand()” on page 5-17
“chr()” on page 5-4 “find()” on page 5-19
“circle()” on page 5-5 “find_index()” on page 5-21
“collapse()” on page 5-6 “generate()” on page 5-22
“contour()” on page 5-8 “get_attr()” on page 5-23
“contour_polar()” on page 5-10 “get_indep_values()” on page 5-24
“copy()” on page 5-12 “indep()” on page 5-26
“create()” on page 5-13

M, P, S, T, V, W
“max_index()” on page 5-27 “sweep_dim()” on page 5-36
“min_index()” on page 5-28 “sweep_size()” on page 5-37
“permute()” on page 5-29 “type()” on page 5-39
“plot_vs()” on page 5-31 “vs()” on page 5-40
“set_attr()” on page 5-33 “what()” on page 5-41
“size()” on page 5-34 “write_var()” on page 5-42
“sort()” on page 5-35

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.
Data Access Functions

**build_subrange()**

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

**Syntax**

\[ y = \text{build}\_\text{subrange}(\text{data, innermost}\_\text{IndepLow, innermost}\_\text{IndepHigh}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>analysis results or user input.</td>
<td>(-\infty, \infty)</td>
<td>integer, real, complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>innermostIndepLow</td>
<td>lowest value of innermost independent</td>
<td>(-\infty, \infty)</td>
<td>integer, real, complex</td>
<td>minimum value of the inner most independent variable</td>
<td>no</td>
</tr>
<tr>
<td>innermostIndepHigh</td>
<td>highest value of innermost independent</td>
<td>(-\infty, \infty)</td>
<td>integer, real, complex</td>
<td>maximum value of the inner most independent variable</td>
<td>no</td>
</tr>
</tbody>
</table>

**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.

\[ \text{subrange}_\text{S12} = \text{build}\_\text{subrange}(\text{S12, 200MHz, 400MHz}) \]

**Defined in**

$\$HPEESOF\_DIR/expressions/aelf/statistical\_fun.ael$
chop()
Replace numbers in x with magnitude less than dx with 0

Syntax
y = chop(x, dx)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>numbers to replace</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>dx</td>
<td>value to compare to</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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,

\[
y = x \text{ if mag}(x) \geq \text{mag}(dx) \\
y = 0 \text{ if mag}(x)
\]
Data Access Functions

**chr()**

Returns the character representation of an integer

**Syntax**

```python
y = chr(x)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>valid number representing a ASCII character</td>
<td>[0, 127]</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

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

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

```python
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

\[ a = \text{circle}(\text{center}, \text{radius}, \text{numPts}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td>center coordinate</td>
<td>(−∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>radius</td>
<td>radius</td>
<td>[0, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>numPts</td>
<td>number of points in the circle</td>
<td>[0, ∞)</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ \begin{align*}
x &= \text{circle}(1, 1, 500) \\
y &= \text{circle}(1 + j \times 1, 1, 500)
\end{align*} \]

Defined in

Built in
Data Access Functions

**collapse()**
Collapses the inner independent variable and returns one dimensional data

**Syntax**
y = collapse(x)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>multi-dimensional data to be collapsed (dimension is larger than one and less than four)</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>mcTrial</th>
<th>freq</th>
<th>S11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100MHz</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>200MHz</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>300MHz</td>
<td>0.6</td>
</tr>
<tr>
<td>2</td>
<td>100MHz</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>200MHz</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>300MHz</td>
<td>0.7</td>
</tr>
</tbody>
</table>

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

collapsed_S11 = collapse(S11)

<table>
<thead>
<tr>
<th>mcTrial</th>
<th>S11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
</tr>
<tr>
<td>2</td>
<td>0.3</td>
</tr>
</tbody>
</table>
defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

expand()

Notes/Equations

The collapse() function cannot be used to convert a matrix into an array or vector. Use the expand() function instead.
### contour()

Generates contour levels on surface data

#### Syntax

```markdown
y = contour(data, contour_levels, interpolation_type)
```

#### Arguments

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

† 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

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

or

```markdown
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.

```markdown
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.
Data Access Functions

**contour_polar()**

Generates contour levels on polar or Smith chart surface data

**Syntax**

\[ y = \text{contour} \_\text{polar}(\text{data}, \text{contour} \_\text{levels}, \text{InterpolationType}, \text{DataFormat}) \]

**Arguments**

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

† 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**

\[ a = \text{contour} \_\text{polar} (\text{data} \_\text{polar}, [1:4]) \]

or

\[ a = \text{contour} \_\text{polar} (\text{data} \_\text{polar}, (1, 2, 3, 4)) \]

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

\[ a = \text{contour} \_\text{polar} (\text{data} \_\text{polar}, (1, 2, 3, 4), 2) \]

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

5-10  contour\_polar()
Defined in
$HPEE$OF_DIR/expressions/acl/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.
Data Access Functions

**copy()**

Makes a copy of a multi-dimensional data variable

**Syntax**

\[ y = \text{copy(DataVar)} \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataVar</td>
<td>data variable or array that is to be copied</td>
<td>boolean, integer, real, complex, string</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ \text{result} = \text{copy(S21)} \]

returns the copy of the data stored in the data variable S21

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

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

**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 structures 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**

\[ y = \text{create}(	ext{Dimensionality}, \text{DependDataType}, \text{IndepName}, \text{IndepType}, \text{NumRows}, \text{NumColumns}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensionality</td>
<td>dimensionality of the data variable or array</td>
<td>([1, \infty))</td>
<td>integer</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>DependDataType</td>
<td>Dependent data type</td>
<td>†</td>
<td>string</td>
<td>&quot;Real&quot;</td>
<td>no</td>
</tr>
<tr>
<td>IndepName</td>
<td>Name(s) of independent</td>
<td></td>
<td>string</td>
<td>&quot;__i&quot;</td>
<td>no</td>
</tr>
<tr>
<td>IndepType</td>
<td>Independent data type</td>
<td>†</td>
<td>string</td>
<td>&quot;Real&quot;</td>
<td>no</td>
</tr>
<tr>
<td>NumRows</td>
<td>Number of rows</td>
<td>([0, \infty))</td>
<td>integer</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>NumColumns</td>
<td>Number of columns</td>
<td>([0, \infty))</td>
<td>integer</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† "Boolean", "Integer", "Real", "Complex", "String" or "Byte16"

**Examples**

```plaintext
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:

\[ \text{indepV} = \text{indep}(\text{result}, "\text{Index}"); \]
\[ \text{result}[0] = 1.1; \]
\[ \text{indepV}[0] = 1.0; \]

An example that modifies the input argument and returns the modified result as a data-variable is given below:

1. Copy the code below into the file user_defined_fun.ael in the directory
   \$HOME/hpeesof/expressions/ael.
   ```plaintext
defun datavar_test_create(arg1)
{
    //Create a 1D data-var
```
Data Access Functions

```c
//create(dimension, DependentDataType, IndepName, IndepType, # rows, #columns)
decl result = create(1, "Real", {"Index"}, {"Real"}, 1, 1);

//Do some simple data manipulation to the input data
result = conj(arg1);

//Set the independent value
decl idenpRV = indep(result, "Index");
idenpRV = indep(arg1);
return result;
```

2. Launch Advanced Design System. In a Data Display window, create an equation:

```c
cre2D = datavar_test_create(S11)
```

3. Display the equation cre2D to view the contents.

A 2-dimensional example is given in the Measurement Expression AEL function below.

```c
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

The create() function can only be used from the measurement expression ael code and not from the DDS or the schematic. See the Examples section.

This function is used to create multi-dimensional data variable or arrays. Data Variables in ADS/RFDE are data structures 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.

The number of rows and columns are used in specifying the dimension of matrix data. For a scalar this would be 1, 1.
**delete()**

Deletes the multi-dimensional data variable

**Syntax**

\[ y = \text{delete(DataVar)} \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataVar</td>
<td>data variable or array that is to be deleted</td>
<td>boolean, integer, real, complex, string</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ \text{result} = \text{delete(S21)} \]

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
\[ y = \text{expand}(x) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to be expanded (dimension is larger than one and less than four)</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>X</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>X=1 10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>20</td>
<td>2</td>
<td>X=2 20</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>30</td>
<td>3</td>
<td>X=3 30</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>40</td>
<td>4</td>
<td>X=4 40</td>
</tr>
</tbody>
</table>

Defined in

Built in
See Also

collapse()

Notes/Equations

In addition to the application above, the expand() function can also be used to convert a matrix into an array or vector. For example, an S-parameter matrix can be converted into an array by using the equation \( \text{arr} = \text{expand}(S) \).
find()
Finds the indices of the conditions that are true. Use with all simulation data

Syntax
indices = find(condition)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>condition</td>
<td>string</td>
<td>yes</td>
</tr>
</tbody>
</table>

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()
Data Access Functions

**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
\( \text{index} = \text{find_index}(\text{data}\_\text{sweep}, \text{search}\_\text{value}) \)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_sweep</td>
<td>data to search</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex, string</td>
<td>yes</td>
</tr>
<tr>
<td>search_value</td>
<td>value to search</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex, string</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

\[
\text{index} = \text{find_index}(\text{freq}, 1\text{GHz})
\]
\[ a = S_{11}[\text{index}] \]

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**

\[ y = \text{generate}(\text{start}, \text{stop}, \text{npts}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>start value of sequence</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>stop</td>
<td>stop value of sequence</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>npts</td>
<td>Number of points in the sequence</td>
<td>[2, \infty)</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{generate}(9, 4, 6) \]

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

**Defined in**

$\text{HPEESOF_DIR/expressions/ael/elementary_fun.ael}$
**get_attr()**

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

**Syntax**

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

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>frequency swept variable</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>attr_name</td>
<td>name of the attribute</td>
<td></td>
<td>string</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>eval</td>
<td>specifies whether to</td>
<td></td>
<td>boolean</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>evaluate the attribute</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Examples**

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

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

```matlab
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**

```plaintext
indepVals = get_indep_values(Data, LookupValue)
```

**Arguments**

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

**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.

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

```plaintext
indepVals = get_indep_values(Pout, [m1,m2])  # returns the independent values of the markers m1 and m2
```

**Defined in**

$HPEESOF_DIR/expressions/ael/utility_fun.ael

**See Also**

`indep()`

---

5-24  get_indep_values()
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.
Data Access Functions

\section*{indep()}

\textit{Returns the independent attached to the data}

\textbf{Syntax}

\[ Y = \text{indep}(x, \text{NumberOrName}) \]

\textbf{Arguments}

\begin{tabular}{|l|l|l|l|l|}
\hline
\textbf{Name} & \textbf{Description} & \textbf{Range} & \textbf{Type} & \textbf{Default} & \textbf{Required} \\
\hline
\textit{x} & data to access the independent values & \( (-\infty, \infty) \) & integer, real, complex & & yes \\
\hline
\textit{NumberOrName} & number or name of independent & \([0, 6]\) & integer, string & 0 & no \\
\hline
\end{tabular}

\textbf{Examples}

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

\begin{verbatim}
freq = indep(S, 1)
freq = indep(S, "freq")
power = indep(S, 2)
power = indep(S, "power")
\end{verbatim}

\textbf{Defined in}

Built in

\textbf{See Also}

\textit{find_index()}, \textit{get_indep_values()}

\textbf{Notes/Equations}

The \texttt{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 \texttt{indep()} returns the innermost independent.
max_index()

Returns the index of the maximum

Syntax
max_index(x)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to find maximum index</td>
<td>(-∞, ∞)</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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()
Data Access Functions

**min_index()**

Returns the index of the minimum

**Syntax**
y = min_index(x)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to find the minimum index</td>
<td>(-\infty, \infty)</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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()

Permutates data based on the attached independents

Syntax

\[ y = \text{permute}(\text{data}, \text{permute \_vector}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>any N-dimensional square data (all inner independents must have the same value N)</td>
<td>((\text{-}\infty, \infty))</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>permute _vector</td>
<td>any permutation vector of the numbers 1 through N †</td>
<td>((\text{-}\infty, \infty))</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

† 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:

```plaintext
a = \text{permute}(\text{data})
reverses the (three inner independents of) the data

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

a\text{Orig} = \text{permute}(\text{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:

```plaintext
\text{permutedData} = \text{permute}(\text{IDS}\_i, \{2, 1\})
```
Data Access Functions

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dependent</td>
<td>any N-dimensional square data (all inner independents must have the same value N)</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>independent</td>
<td>independent variable</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

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
Data Access Functions

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>data</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>attr_name</td>
<td>name of the attribute</td>
<td></td>
<td>string</td>
<td>yes</td>
</tr>
<tr>
<td>attribute_value</td>
<td>value of the attribute</td>
<td>(-∞, ∞)</td>
<td>boolean, integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**
a = set_attr(data, "TraceType", "Spectral")
a = set_attr(data, "TraceType", "Histogram")

**Defined in**
Built in

**See Also**
get_attr()
Data Access Functions

**size()**

Returns the row and column size of a vector or matrix

**Syntax**

\[ y = \text{size}(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data</td>
<td>(-(\infty), (\infty))</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

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

\[
Y = \text{size}(S) \\
\text{return} \ \{2, 2\} \\
Y = \text{sweep\_size}(S) \\
\text{return} \ 10
\]

**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**

\[ y = \text{sort}(\text{data}, \text{sortOrder}, \text{indepName}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>data to be sorted (multidimensional scalar variable)</td>
<td>(-\infty, \infty)</td>
<td>integer, real or complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>sortOrder</td>
<td>sorting order</td>
<td>&quot;ascending&quot; or &quot;descending&quot;</td>
<td>string</td>
<td>&quot;ascending&quot;</td>
<td>no</td>
</tr>
<tr>
<td>indepName</td>
<td>specify the name of the independent variable for sorting</td>
<td>string</td>
<td>dependent value †</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

† if indepName not specified, the sorting is done on the dependent

**Examples**

```
a = \text{sort}(\text{data})
a = \text{sort}(\text{data}, "\text{descending}", "\text{freq}")
```

**Defined in**

Built in
Data Access Functions

**sweep_dim()**

Returns the dimensionality of the data

**Syntax**

\[ y = \text{sweep\_dim}(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data</td>
<td>(-\infty, \infty)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
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

\[ y = \text{sweep\_size}(x) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data</td>
<td>(-\infty, \infty)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

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

\[ a = \text{sweep\_size}(S) \]
returns 10

\[ a = \text{size}(S) \]
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:

\[ \text{SwpSz}=\text{sweep\_size(data)} \]

will return:

\[
\begin{array}{c|c|c}
\text{__SIZE} & \text{SwpSz} \\
\hline
\text{Vsrc}=1.0,\text{size}=1.0 & 1 & 20 \\
\text{Vsrc}=1.0,\text{size}=2.0 & 1 & 21 \\
\text{Vsrc}=2.0,\text{size}=1.0 & 1 & 59 \\
\text{Vsrc}=2.0,\text{size}=2.0 & 1 & 61 \\
\text{Vsrc}=3.0,\text{size}=1.0 & 1 & 76 \\
\text{Vsrc}=3.0,\text{size}=2.0 & 1 & 78 \\
\end{array}
\]

where \text{__SIZE} is an independent added by the \text{sweep\_size()} function.
Data Access Functions

\texttt{sweep\_size(SwpSz)} would return the correct size of the two outer variables:

\begin{tabular}{ccc}
(1) & (2) & (3) \\
3 & 2 & 1
\end{tabular}

\textbf{Defined in}

Built in

\textbf{See Also}

\texttt{size()}, \texttt{sweep\_dim()}

\textbf{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

\[ y = \text{type}(x) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to find the type</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex, string</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

```
a = \text{type}(1)
returns "Integer"
```
```
a = \text{type}(1.1)
returns "Real"
```
```
a = \text{type}(1i)
returns "Complex"
```
```
a = \text{type}("type")
returns "String"
```

Defined in

Built in

See Also

what()
vs()
Attaches an independent to dependent data

Syntax
\[ y = vs(\text{dependent}, \text{independent}, \text{indepName}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dependent</td>
<td>dependent values</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>independent</td>
<td>independent values</td>
<td>(-∞, ∞)</td>
<td>integer, real, string, complex</td>
<td>yes</td>
</tr>
<tr>
<td>indepName</td>
<td>independent name</td>
<td></td>
<td>string</td>
<td>no</td>
</tr>
</tbody>
</table>

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 \( I_{ds} \) vs \( V_{gs} \), in a DC analysis data with two independent variables, \( V_{gs} \) and \( V_{ds} \), and a dependent variable \( I_{ds.i} \), use as below:

\[ \text{plot}_\text{vs}(I_{ds.i}, V_{gs}) \]
what()

Returns size and type of data

Syntax
\[ y = \text{what}(x, \text{DisplayBlockName}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data</td>
<td>(-(\infty, \infty))</td>
<td>integer, real, complex, string</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>DisplayBlockName</td>
<td>Displays block name</td>
<td>[0, 1]†</td>
<td>integer</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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

\[ x = [10, 20, 30, 40] \]
\[ y = \text{what}(x) \]

<table>
<thead>
<tr>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency : []</td>
</tr>
<tr>
<td>Num. Points : [4]</td>
</tr>
<tr>
<td>Matrix Size : scalar</td>
</tr>
<tr>
<td>Type : Integer</td>
</tr>
</tbody>
</table>

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).
Data Access Functions

**write_var()**
Writess dataset variables to a file

**Syntax**

\[
y = \text{write_var}(\text{FileName}, \text{WriteMode}, \text{Comment}, \text{Delimiter}, \text{Format}, \text{Precision}, \text{Var1}, \text{Var2}, ..., \text{VarN})
\]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>Name of the output file</td>
<td>string</td>
<td></td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>WriteMode</td>
<td>Describes the write mode - overwrite or append</td>
<td>&quot;W&quot;</td>
<td>&quot;A&quot; †</td>
<td>string</td>
<td>yes</td>
</tr>
<tr>
<td>Comment</td>
<td>Text to be written at the top of the file</td>
<td>string</td>
<td><strong>&quot;&quot;</strong></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Delimiter</td>
<td>Delimiter that separates the data</td>
<td>string</td>
<td><strong>&quot;&quot;</strong></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Format</td>
<td>Format of the data</td>
<td>&quot;f&quot;</td>
<td>&quot;s&quot; ‡</td>
<td>string</td>
<td>&quot;f&quot;</td>
</tr>
<tr>
<td>Precision</td>
<td>precision of the data</td>
<td>[1, 64]</td>
<td>integer</td>
<td>6</td>
<td>no</td>
</tr>
<tr>
<td>Var1,...,VarN</td>
<td>Data variables to be written</td>
<td>dataset variable</td>
<td></td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>

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

**Examples**

```plaintext
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
```

```plaintext
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**

5-42 write_var()
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.
Data Access Functions


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
“carr_to_im()” on page 6-2  “pspec()” on page 6-20
“cdrange()” on page 6-3  “pt()” on page 6-21
“dc_to_rf()” on page 6-4  “remove_noise()” on page 6-22
“ifc()” on page 6-5  “sfdr()” on page 6-23
“ip3_in()” on page 6-6  “snr()” on page 6-25
“ip3_out()” on page 6-8  “spur_track()” on page 6-27
“ipn()” on page 6-10  “spur_track_with_if()” on page 6-29
“it()” on page 6-12  “thd_func()” on page 6-31
“mix()” on page 6-13  “ts()” on page 6-32
“ns_circle()” on page 9-34  “vfc()” on page 6-35
“pae()” on page 6-15  “vspec()” on page 6-37
“pfc()” on page 6-17  “vt()” on page 6-38
“phase_gain()” on page 6-19

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.
Harmonic Balance Functions

**carr_to_im()**

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

**Syntax**

\[ y = \text{carr\_to\_im}(vOut, \text{fundFreq}, \text{imFreq}, \text{Mix}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vOut</td>
<td>signal voltage at the output port</td>
<td>([0, \infty)]</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>fundFreq</td>
<td>harmonic frequency indices for the fundamental frequency</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td>yes</td>
</tr>
<tr>
<td>imFreq</td>
<td>harmonic frequency indices for the IMD product of interest</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td>yes</td>
</tr>
<tr>
<td>Mix</td>
<td>consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td>no</td>
</tr>
</tbody>
</table>

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

**Examples**

\[ a = \text{carr\_to\_im}(\text{out}, (1, 0), (2, -1)) \]

**Defined in**

$HPE\_SO\_DIR/expressions/ael/rf\_system\_fun.ael$

**See Also**

ip3_out()
cdrange()

Returns compression dynamic range

Syntax

\[
y = \text{cdrange}(nf, \text{inpwr\_lin}, \text{outpwr\_lin}, \text{outpwr})
\]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>nf</td>
<td>noise figure at the output port</td>
<td>([0, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>inpwr_lin</td>
<td>input power in the linear region</td>
<td>([0, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>outpwr_lin</td>
<td>output power in the linear region</td>
<td>([0, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>outpwr</td>
<td>output power at 1 dB compression</td>
<td>([0, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[
a = \text{cdrange}(nf2, \text{inpwr\_lin}, \text{outpwr\_lin}, \text{outpwr})
\]

Defined in

$\text{HPEE Sof\_DIR/expressions/acl/rf\_system\_fun.ael}$

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.
Harmonic Balance Functions

**dc_to_rf()**

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

**Syntax**

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

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlusRF</td>
<td>voltage at the positive terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vMinusRF</td>
<td>voltage at the negative terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vPlusDC</td>
<td>DC voltage at the positive terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vMinusDC</td>
<td>DC voltage at the negative terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>currentRF</td>
<td>RF current for power calculation</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>currentDC</td>
<td>DC current for power calculation</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>harm_freq_index</td>
<td>harmonic index of the RF frequency at the output port</td>
<td>(-∞, ∞)</td>
<td>integer array</td>
<td>yes</td>
</tr>
<tr>
<td>Mix</td>
<td>consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †</td>
<td>(-∞, ∞)</td>
<td>matrix</td>
<td>no</td>
</tr>
</tbody>
</table>

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums

**Examples**

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

**Defined in**

```
$HPEESOF_DIR/expressions/ael/circuit_fun.ael
```

6-4  dc_to_rf()
ifc()

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

**Syntax**

\[ y = \text{ifc}(iOut, \text{harm}_\text{freq}_{\text{index}}, \text{Mix}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOut</td>
<td>current through a branch</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>harm_freq_index</td>
<td>harmonic index of the desired frequency †</td>
<td>(-\infty, \infty)</td>
<td>integer array</td>
<td>yes</td>
</tr>
<tr>
<td>Mix</td>
<td>consists of all possible vectors of harmonic frequency (mixing terms) in the analysis † †</td>
<td>(-\infty, \infty)</td>
<td>matrix</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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:

\[ y = \text{ifc}(I_{\text{Probe1}}.i, \{1, 0\}) \]

**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).
Harmonic Balance Functions

**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**

\[ y = \text{ip3\_in}(vOut, \text{ssGain}, \text{fundFreq}, \text{imFreq}, zRef, \text{Mix}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vOut</td>
<td>signal voltage at the output port</td>
<td>([0, \infty))</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>ssGain</td>
<td>small signal gain in dB</td>
<td>([0, \infty))</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>fundFreq</td>
<td>harmonic frequency indices for the fundamental frequency</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>imFreq</td>
<td>harmonic frequency indices for the intermodulation frequency</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>Mix</td>
<td>consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

**Examples**

\[ y = \text{ip3\_in}(vOut, 22, \{1, 0\}, \{2, -1\}, 50) \]

**Defined in**

$HPEE/SOF_DIR/expressions/ael/rf_system_fun.ael$

**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.
Harmonic Balance Functions

**ip3_out()**

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

**Syntax**

\[ y = \text{ip3\_out}(\text{vOut}, \text{fundFreq}, \text{imFreq}, \text{zRef}, \text{Mix}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vOut</td>
<td>signal voltage at the output port</td>
<td>([0, \infty))</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>fundFreq</td>
<td>harmonic frequency indices for the fundamental frequency</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>imFreq</td>
<td>harmonic frequency indices for the intermodulation frequency</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>Mix</td>
<td>consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

**Examples**

\[ y = \text{ip3\_out}(\text{vOut}, \{1, 0\}, \{2, -1\}, 50) \]

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/rf\_system\_fun.ael}$

**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.
Harmonic Balance Functions

**ipn()**

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

**Syntax**

\[ y = \text{ipn}(v_{\text{Plus}}, v_{\text{Minus}}, i_{\text{Out}}, \text{fundFreq}, \text{imFreq}, n, \text{Mix}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlus</td>
<td>voltage at the positive output terminal</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vMinus</td>
<td>voltage at the negative output terminal</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>iOut</td>
<td>current through a branch</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>fundFreq</td>
<td>harmonic indices of the fundamental frequency</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td>yes</td>
</tr>
<tr>
<td>imFreq</td>
<td>harmonic indices of the intermodulation frequency</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td>yes</td>
</tr>
<tr>
<td>n</td>
<td>order of the intercept</td>
<td>[1, (\infty)]</td>
<td>integer</td>
<td>yes</td>
</tr>
<tr>
<td>Mix</td>
<td>consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †</td>
<td>((-\infty, \infty))</td>
<td>matrix</td>
<td>no</td>
</tr>
</tbody>
</table>

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

**Examples**

\[ y = \text{ipn}(v_{\text{Out}}, 0, I_{\text{Probel}.i}, \{1, 0\}, \{2, -1\}, 3) \]

**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.
Harmonic Balance Functions

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

Syntax
it(iOut, tmin, tmax, numOfPnts)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOut</td>
<td>current through a branch</td>
<td>$(-\infty, \infty)$</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>tmin</td>
<td>start time</td>
<td>$[0, \infty)$</td>
<td>real</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>tmax</td>
<td>stop time</td>
<td>$[0, \infty)$</td>
<td>real</td>
<td>2*cycle time</td>
<td>no</td>
</tr>
<tr>
<td>numOfPnts</td>
<td>number of points</td>
<td>$[0, \infty)$</td>
<td>integer</td>
<td>101</td>
<td>no</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>xOut</td>
<td>voltage or a current spectrum</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>harmIndex</td>
<td>desired vector of harmonic frequency</td>
<td>(-∞, ∞)</td>
<td>integer array</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indices (mixing terms)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mix</td>
<td>variable consisting of all possible</td>
<td>(-∞, ∞)</td>
<td>matrix</td>
<td>†</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>vectors of harmonic frequency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>indices (mixing terms) in the analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† 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.

\[
y = \text{mix}(\text{vOut}, \{2, -1\})
\]

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

\[
v\text{Exp}=\text{vOut}^2 \div 50
\]

\[
z = \text{mix}(v\text{Exp}, \{2, -1\}, \text{Mix})
\]

**Defined in**

Built in

**See Also**

find_index()
Harmonic Balance Functions

**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**

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

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlusOut</td>
<td>output voltage at the positive terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vMinusOut</td>
<td>output voltage at the negative terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vPlusIn</td>
<td>input voltage at the positive terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vMinusIn</td>
<td>input voltage at the negative terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vPlusDC</td>
<td>DC voltage at the positive terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vMinusDC</td>
<td>DC voltage at the negative terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>iOut</td>
<td>output current</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>iIn</td>
<td>input current</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>iDC</td>
<td>DC current</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>outFreq</td>
<td>harmonic indices of the fundamental frequency at the output port</td>
<td>(-∞, ∞)</td>
<td>integer array</td>
<td>yes</td>
</tr>
<tr>
<td>inFreq</td>
<td>harmonic indices of the fundamental frequency at the input port</td>
<td>(-∞, ∞)</td>
<td>integer array</td>
<td>yes</td>
</tr>
<tr>
<td>Mix</td>
<td>consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †</td>
<td>(-∞, ∞)</td>
<td>matrix</td>
<td>no</td>
</tr>
</tbody>
</table>

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.
Harmonic Balance Functions

**Examples**

\[ y = \text{pae}(vOut, 0, vIn, 0, v1, 0, I_{\text{Probe1}.i}, I_{\text{Probe2}.i}, I_{\text{Probe3}.i}, 1, 1) \]

**Defined in**

$\text{HPEESOF \_DIR/expressions/acl/circuit\_fun.acl}$

**See Also**

\text{db()}, \text{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**

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

† 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_Probel.i, {1, 0})

**Defined in**
$HPEESOF_DIR/expressions/ael/circuit_fun.ael

**See Also**
ifc(), vfc()
Harmonic Balance Functions

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**

\[ y = \text{phase} \_\text{gain}(\text{Gain}, \text{DesiredPhase}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gain</td>
<td>Two dimensional data representing gain.</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>DesiredPhase</td>
<td>A single value representing the desired phase.</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

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

\[ \text{gainAtZeroPhase} = \text{phase} \_\text{gain}(\text{Vout}/\text{Vin}, 0) \]

returns the gain at zero phase.

**Defined in**

$\text{HPEESOF}\_\text{DIR}/expressions/ael/\text{rf}\_\text{system}\_\text{fun}.ael$
Harmonic Balance Functions

**pspec()**

This measurement gives a power frequency spectrum in harmonic balance analyses.

**Syntax**

```plaintext
y = pspec(vPlus, vMinus, iOut)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlus</td>
<td>voltage at the positive node</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>vMinus</td>
<td>voltage at the negative node</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>iOut</td>
<td>current through a branch</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
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**

\[ y = \text{pt}(vPlus, vMinus, iOut) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlus</td>
<td>voltage at the positive node</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>vMinus</td>
<td>voltage at the negative node</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>iOut</td>
<td>current through a branch</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

\[ y = \text{pt}(vOut, 0, I\_Probe\_1.i) \]

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/circuit\_fun.ael}$

**See Also**

pspec()
Harmonic Balance Functions

**remove_noise()**

Removes noise floor data from noise data and returns an array

**Syntax**

\[ \text{nd} = \text{remove\_noise} (\text{NoiseData}, \text{NoiseFloor}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoiseData</td>
<td>Two dimensional array representing noise data</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>NoiseFloor</td>
<td>Single dimensional array representing noise floor</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ \text{nd} = \text{remove\_noise} (\text{vnoise}, \text{noiseFloor}) \]

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 - NoiseFloor is less than zero, then -200 dBm is used.
sfdr()

Returns the spurious-free dynamic range

**Syntax**

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

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vOut</td>
<td>signal voltage at the output port</td>
<td>([0, \infty))</td>
<td>real, complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>ssGain</td>
<td>small signal gain in dB</td>
<td>([0, \infty))</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>nf</td>
<td>noise figure at the output port</td>
<td>([0, \infty))</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>fundFreq</td>
<td>harmonic frequency indices for the fundamental frequency</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>imFreq</td>
<td>harmonic frequency indices for the intermodulation frequency</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>Mix</td>
<td>consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †</td>
<td>((-\infty, \infty))</td>
<td>integer array</td>
<td></td>
<td>no</td>
</tr>
</tbody>
</table>

† Mix is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums

**Examples**

\[ a = \text{sfdr}(vIn, 12, \text{nf}2, , \{1, 0\}, \{2, -1\}, 50) \]

**Defined in**

$\text{HPEESOF_DIR/expressions/ael/rf_system_fun.ael}$

**See Also**

ip3_out()
Harmonic Balance Functions

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vOut</td>
<td>signal voltage at the output port</td>
<td>[0, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vOut.noise</td>
<td>noise voltage at the output port</td>
<td>[0, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>fundFreq</td>
<td>harmonic frequency indices for the fundamental frequency †</td>
<td>(-∞, ∞)</td>
<td>integer array</td>
<td>yes</td>
</tr>
<tr>
<td>Mix</td>
<td>consists of all possible vectors of harmonic frequency (mixing terms) in the analysis † †</td>
<td>(-∞, ∞)</td>
<td>integer array</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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()
Harmonic Balance Functions

**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

\[
\text{IFspur} = \text{spur\_track}(\text{vs(vout, freq), if\_low, if\_high, rout})
\]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vout</td>
<td>IF output node name</td>
<td>string</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>if_low</td>
<td>lowest frequency in the IF band</td>
<td>[0, (\infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>if_high</td>
<td>highest frequency in the IF band</td>
<td>[0, (\infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>rout</td>
<td>load resistance connected to the IF port, necessary for computing power delivered to the load</td>
<td>[0, (\infty))</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[
\text{IFspur} = \text{spur\_track}(\text{vs(HB.VIF1, freq), Fiflow[0, 0], Fifhigh[0, 0], 50})
\]

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.
Harmonic Balance Functions

Defined in
$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also
spur_track_with_if()

Notes/Equations
Used in Receiver spurious response simulations.

IF spur 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vout</td>
<td>IF output node name</td>
<td>string</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>if_low</td>
<td>lowest frequency in the IF band</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>if_high</td>
<td>highest frequency in the IF band</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>rout</td>
<td>load resistance connected to the IF port, necessary for computing power delivered to the load</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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.
Harmonic Balance Functions

Defined in

\$HPEE\_SOF\_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.

6-30 spur_track_with_if()
thd_func()

This measurement returns the Total Harmonic Distortion percentage

Syntax
y = thd_func(v)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>voltage</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

y = thd_func(Vload)

Defined In

$HPEE SOF_DIR/expressions/ael/rf_system_fun.ael
Harmonic Balance Functions

\textbf{ts()}

Performs a frequency-to-time transform

\textbf{Syntax}

\[ y = \text{ts}(x, \text{tstart}, \text{tstop}, \text{numtpts}, \text{dim}, \text{windowType}, \text{windowConst}, \text{nptsspec}) \]

\textbf{Arguments}

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>frequency-domain data to be transformed</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>tstart</td>
<td>starting time</td>
<td>[0, \infty)</td>
<td>integer, real</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>tstop</td>
<td>stoping time</td>
<td>[0, \infty)</td>
<td>integer, real</td>
<td>tstop = tstart + 2.0/fabs(freq[0])</td>
<td>no</td>
</tr>
<tr>
<td>numtpts</td>
<td>number of time points</td>
<td>[1, \infty)</td>
<td>integer</td>
<td>101</td>
<td>no</td>
</tr>
<tr>
<td>dim</td>
<td>dimension to be transformed (not used currently)</td>
<td>[1, \infty)</td>
<td>integer</td>
<td>highest dimension</td>
<td>no</td>
</tr>
<tr>
<td>windowType</td>
<td>type of window to be applied to the data</td>
<td>[0, 9] \dag</td>
<td>integer, string</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>windowConst</td>
<td>window constant \ddag</td>
<td>[0, \infty)</td>
<td>integer, real</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>nptsspec</td>
<td>number of first harmonics to be transformed</td>
<td>[1, NumFreqs]</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
</tbody>
</table>

\dag 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
\ddag windowConst is not used if windowType is 8510

6-32 ts()
Examples
The following examples of ts assume that a Harmonic Balance simulation was performed with a fundamental frequency of 1 GHz and order = 8:

\[ Y = \text{ts}(vOut) \]
returns the time series (0, 20ps, ..., 2ns)

\[ Y = \text{ts}(vOut, 0, 1ns) \]
returns the time series (0, 10ps, ..., 1ns)

\[ Y = \text{ts}(vOut, 0, 10ns, 201) \]
returns the time series (0, 50ps, ..., 10ns)

\[ Y = \text{ts}(vOut, , , , , , , 3) \]
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
Harmonic Balance Functions

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 innermost 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:

$$\text{tsERR} = \text{ts(Idd.i - Iout.i)}$$

would return the incorrect time axis values. This can be solved by the expression:

$$\text{tsWORKS} = \text{ts(Idd.i)} - \text{ts(Iout.i)}$$

Similarly the expression:

$$\text{tsERR1} = \text{ts(Idd.i[0,:])}$$

would return the incorrect time axis values. This can be solved by the expressions:

$$\text{ts_Idd} = \text{ts(Idd.i)}$$
$$\text{ts_Idd_0} = \text{ts_Idd[0,:]}$$
vfc()

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

Syntax

\[ y = \text{vfc}(vPlus, vMinus, \text{harm_freq_index}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlus</td>
<td>voltage at the positive output terminal</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vMinus</td>
<td>voltage at the negative output terminal</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>harm_freq_index</td>
<td>harmonic index of the desired frequency †</td>
<td>(-\infty, \infty)</td>
<td>integer array</td>
<td>yes</td>
</tr>
<tr>
<td>Mix</td>
<td>consists of all possible vectors of harmonic frequency (mixing terms) in the analysis ††</td>
<td>(-\infty, \infty)</td>
<td>matrix</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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:

\[ a = \text{vfc}(vOut, 0, \{1, 0\}) \]

Defined in

$\text{HPEESOF_DIR/expressions/ael/circuit_fun.ael}$

See Also

ifc(), pfc()
Harmonic Balance Functions

**Notes/Equations**

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

6-36 $\text{vfc}()$
vspec()

Returns the voltage frequency spectrum

Syntax

\[ y = \text{vspec}(\text{vPlus}, \text{vMinus}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlus</td>
<td>voltage at the positive node</td>
<td>((\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vMinus</td>
<td>voltage at the negative node</td>
<td>((\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ a = \text{vspec}(v1, v2) \]

Defined in

$\text{HPEESOF\_DIR/expressions/ael/circuit\_fun.ael}$

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.
Harmonic Balance Functions

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlus</td>
<td>voltage at the positive node</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>vMinus</td>
<td>voltage at the negative node</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>tmin</td>
<td>start time</td>
<td>[0, ∞)</td>
<td>real</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>tmax</td>
<td>stop time</td>
<td>[0, ∞)</td>
<td>real</td>
<td>2*cycle time</td>
<td>no</td>
</tr>
<tr>
<td>numOfPnts</td>
<td>number of points</td>
<td>[0, ∞)</td>
<td>integer</td>
<td>101</td>
<td>no</td>
</tr>
</tbody>
</table>

**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.

A
"abs()" on page 7-4  "asin()" on page 7-9
"acos()" on page 7-5  "asinh()" on page 7-10
"acosh()" on page 7-6  "atan()" on page 7-11
"acot()" on page 7-7  "atan2()" on page 7-12
"acoth()" on page 7-8  "atanh()" on page 7-13

C
"ceil()" on page 7-14  "convOct()" on page 7-22
"cint()" on page 7-15  "cos()" on page 7-23
"cmplx()" on page 7-16  "cosh()" on page 7-24
"complex()" on page 7-17  "cot()" on page 7-25
"conj()" on page 7-18  "coth()" on page 7-26
"convBin()" on page 7-19  "cum_prod()" on page 7-27
"convHex()" on page 7-20  "cum_sum()" on page 7-28
"convInt()" on page 7-21

D,E,F,H
"db()" on page 7-29  "erfinv()" on page 7-39
"dbm()" on page 7-30  "exp()" on page 7-40
"dbmtow()" on page 7-32  "fft()" on page 7-41
"deg()" on page 7-33  "fix()" on page 7-42
"diagonal()" on page 7-34  "float()" on page 7-43
"diff()" on page 7-35  "floor()" on page 7-44
"erf()" on page 7-36  "fmod()" on page 7-45
"erfc()" on page 7-37  "hypot()" on page 7-46
"erfcinv()" on page 7-38
Math Functions

I,J,L

“identity()” on page 7-47  “interpolate()” on page 7-54
“im()” on page 7-48  “inverse()” on page 7-55
“imag()” on page 7-49  “jn()” on page 7-56
“int()” on page 7-50  “ln()” on page 7-57
“integrate()” on page 7-51  “log()” on page 7-58
“interp()” on page 7-53  “log10()” on page 7-59

M,N,O

“mag()” on page 7-60  “min_outer()” on page 7-65
“max()” on page 7-61  “min2()” on page 7-66
“max_outer()” on page 7-62  “num()” on page 7-67
“max2()” on page 7-63  “ones()” on page 7-68
“min()” on page 7-64

P,R

“phase()” on page 7-69  “rad()” on page 7-75
“phasedeg()” on page 7-70  “re()” on page 7-76
“phaserad()” on page 7-71  “real()” on page 7-77
“polar()” on page 7-72  “rms()” on page 7-78
“pow()” on page 7-73  “round()” on page 7-79
“prod()” on page 7-74

S,T,X,Z

“sgn()” on page 7-80  “sum()” on page 7-87
“sin()” on page 7-81  “tan()” on page 7-88
“sinc()” on page 7-82  “tanh()” on page 7-89
“sinh()” on page 7-83  “transpose()” on page 7-90
“sqr()” on page 7-84  “wtodbm()” on page 7-91
“sqrt()” on page 7-85  “xor()” on page 7-92
“step()” on page 7-86  “zeros()” on page 7-93
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.
Math Functions

**abs()**
Returns the absolute value of an integer, real or complex number

**Syntax**
y = abs(x)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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
\[ y = \text{acos}(x) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples
\[ a = \text{acos}(-1) \]
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

a = acot(1.5)  
returns 0.588

Defined in  
Built in

See Also

asin(), atan(), atan2()
Math Functions

**acoth()**

Returns the inverse hyperbolic cotangent

**Syntax**

\[ y = \text{acoth}(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(−∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

a = asin(-1)
returns -1.571

Defined in
Built in

See Also
acos(), atan(), atan2()
Math Functions

**asinh()**
Returns the inverse hyperbolic sine

**Syntax**
y = asinh(x)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

a = atan(-1)
returns −0.785

Defined in
Built in

See Also
acos(), asin(), atan2()

Notes/Equations
Returned value range is -π/2 to π/2.
Math Functions

atan2()
Returns the inverse tangent, or arc tangent, of the rectangular coordinates y and x

Syntax
y = atan2(y, x)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>y</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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 π.
Math Functions

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

$y = \text{ceil}(x)$

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>number</td>
<td>$(-\infty, \infty)$</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

$a = \text{ceil}(5.27)$
returns 6

Defined in

Built in
cint()

Given a non-integer real number, returns a rounded integer

Syntax
y = cint(x)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

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**

\[ y = \text{cmplx}(x, y) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>real part of complex number</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>y</td>
<td>imaginary part of complex number</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{cmplx}(2, -1) \]
returns 2 - 1j

**Defined in**

Built in

**See Also**

complex(), imag(), real()
**complex()**

Returns complex number given real and imaginary

**Syntax**

\[ y = \text{complex}(x, y) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>real part of complex number</td>
<td>((-\infty, \infty))</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>y</td>
<td>imaginary part of complex number</td>
<td>((-\infty, \infty))</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{complex}(2, -1) \]
returns \(2 - 1j\)

**Defined in**

$\text{HPEESOF}_\text{DIR}/\text{expressions/ael/elementary_fun.ael}$

**See Also**

`cmplx()`, `imag()`, `real()`
Math Functions

**conj()**
Returns the conjugate of a complex number

**Syntax**
y = conj(x)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>integer to be converted to a binary string</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>num</td>
<td>number of digits in the binary string</td>
<td>[0, ∞)</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

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**

\[ y = \text{convHex}(\text{val}, \text{num}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>integer to be converted to a hexadecimal string</td>
<td>((-\infty, \infty))</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>num</td>
<td>number of digits in the hexadecimal string</td>
<td>([0, \infty))</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[
a = \text{convHex}(1064, 8)\]

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>string representation of the binary, octal or hexadecimal number to be converted</td>
<td>string</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>base</td>
<td>base of the conversion</td>
<td>2</td>
<td>8</td>
<td>16 †</td>
</tr>
</tbody>
</table>

† 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()`
Math Functions

convOct()

Returns an octal string of an integer with n-digits

Syntax

\[ y = \text{convOct}(\text{val}, \text{num}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>integer to be converted to an octal string</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>num</td>
<td>number of digits in the octal string</td>
<td>[0, ∞)</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number in radians</td>
<td>(-\infty, \infty)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ y = \cos(\pi/3) \]
returns 0.500

Defined in

Built in

See Also

\[ \sin(), \tan() \]
Math Functions

**cosh()**

Returns the hyperbolic cosine

**Syntax**

\[ y = \cosh(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number in radians</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

a = \( \cot(1.5) \)
returns 0.071

**Defined in**
Built in

**See Also**

tan(), tanh()
Math Functions

**coth()**

Returns the hyperbolic cotangent

**Syntax**

```
y = coth(x)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```
a = coth(1.5)
returns 1.105
```

**Defined in**

Built in

**See Also**

`cot()`, `tan()`, `tanh()`
**cum_prod()**

Returns the cumulative product

**Syntax**

\[ y = \text{cum} \_ \text{prod}(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to find cumulative product</td>
<td>((\infty, \infty))</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ y = \text{cum} \_ \text{prod}(1) \]
returns \(1.000\)

\[ y = \text{cum} \_ \text{prod}([1, 2, 3]) \]
returns \([1.000, 2.000, 6.000]\)

\[ y = \text{cum} \_ \text{prod}([i, i]) \]
returns \([i, i^2]\)

**Defined in**

Built in

**See Also**

`cum_sum()`, `max()`, `mean()`, `min()`, `prod()`, `sum()`
Math Functions

**cum_sum()**

Returns the cumulative sum

**Syntax**

```markdown
y = cum_sum(x)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to find cumulative sum</td>
<td>(-∞, ∞)</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```markdown
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()
\texttt{db()}

Returns the decibel measure of a voltage ratio

**Syntax**

\[ y = \text{db}(r, z1, z2) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r)</td>
<td>voltage ratio (vOut/vIn)</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>(z1)</td>
<td>source impedance</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>(z2)</td>
<td>load impedance</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

\[ y = \text{db}(100) \]
returns 40

\[ y = \text{db}(8-6\times j) \]
returns 20

**Defined in**

Built in

**See Also**

\texttt{dbm()}, \texttt{pae()}

**Notes/Equations**

\[ \text{dbValue} = 20 \log(\text{mag}(r)) - 10 \log(\text{zOutfactor}/\text{zInfactor}) \]

\[ \text{zOutfactor} = \text{mag}(z2)^2 / \text{real}(z2) \]

\[ \text{zInfactor} = \text{mag}(z1)^2 / \text{real}(z1). \]
Math Functions

**dbm()**

Returns the decibel measure of a voltage referenced to a 1 milliwatt signal

**Syntax**

\[ y = \text{dbm}(v, z) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>voltage (the peak voltage)</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>z</td>
<td>impedance</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

\[ y = \text{dbm}(100) \]

returns 50

\[ y = \text{dbm}(8-6*i) \]

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:

\[
\text{noise}\_\text{power} = \text{dbm}(\text{vout}\_\text{noise} \times \sqrt{2});
\]
Understanding the dbm() Function

Given a power $P_o$ in Watts, the power in dB is:

$$P_o_{dBW} = 10 \times \log(mag(P_o/(1\ W)))$$

while the power in dBm is:

$$P_o_{dBm} = 10 \times \log(mag(P_o/(1\ mW)))$$

$$= 10 \times \log(mag(P_o/(1\ W))) + 30$$

$$= P_o_{dB} + 30$$

Given a voltage $V_o$ in Volts, the voltage in dB is:

$$V_{dBV} = 20 \times \log(mag((V_o/(1\ V)))$$

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 $Z_o$, the power-voltage relation is:

$$P_o = (V_o)^2/(2*Z_o)$$

Using the above, $P_o$ in dBm is then:

$$P_o_{dBm} = 10 \times \log(mag(P_o/(1\ W))) + 30$$

$$= 10 \times \log(mag((V_o/(1\ V))^2/(2*Z_o/(1\ Ohm)))) + 30$$

$$= 10 \times \log(mag((V_o/(1\ V))^2)) - 10 \times \log(mag(2*Z_o/(1\ Ohm))) + 30$$

$$= 20 \times \log(mag(V_o/(1\ V))) - 10 \times \log(mag(Z_o/(50\ Ohm))) + 10$$

This is the dbm() function - voltage in dBm in a $Z_o$ environment.
dbmtow()

Converts dBm to watts

**Syntax**
wValue = dbmtow(P)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>power expressed in dBm</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[
y = dbmtow(0) \\
\text{returns .001 W}
\]

\[
y = dbmtow(-10) \\
\text{returns 1.000E-4 W}
\]

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/elementary\_fun.ael}$

**See Also**

dbm(), wtodbm()
**deg()**

Converts radians to degrees

**Syntax**

\[ y = \text{deg}(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number in radians</td>
<td>(-(\infty), (\infty))</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ y = \text{deg}(1.5708) \]
returns 90

\[ y = \text{deg}(\pi) \]
returns 180

**Defined in**

Built in

**See Also**

rad()
Math Functions

**diagonal()**

Returns the diagonal of a square matrix as a matrix

**Syntax**

\[ y = \text{diagonal}(\text{Matrix}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>square matrix to find the diagonal</td>
<td>((-\infty, \infty))</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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**

\[ y = \text{diff}(\text{data}, \text{pad}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>data to find numerical derivative</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>pad</td>
<td>pad the differentiated data with an extra value</td>
<td>([0, 1])†</td>
<td>integer</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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**

\[ \text{group\_delay} = -\text{diff}(\text{unwrap}(\text{phaserrad}(\text{S21}), \pi)) / (2\pi) \]

**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".
Math Functions

erf()

Calculates the error function, the area under the Gaussian curve exp(-x**2)

Syntax

y = erf(x)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>[0, 2] †</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

† 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>[-1, 1]</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

† 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()
Math Functions

**exp()**

The exponential function is used to calculate powers of e. Given a complex number, \( x \), the \( \text{exp}(x) \) function calculates \( e \) to the power of \( x \) (i.e. \( e^x \)).

**Syntax**

\[ y = \text{exp}(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>number</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

1. \( a = \text{exp}(1) \)
   
   returns 2.71828

2. \( b = \text{exp}(1+j1) \)
   
   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

\[ e^x = e^a + e^{j*b} = (e^a)(\cos(b) + j*sin(b)) \]
fft()
Performs the discrete Fourier transform

**Syntax**
y = fft(x, length)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to be transformed</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>length</td>
<td>length of the transform</td>
<td>[1, ∞)</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
fft([1, 1, 1, 1])
returns [4+0i, 0+0i]
```

```plaintext
fft([1, 0, 0, 0])
returns [1+0i, 1+0i]
```

```plaintext
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.
Math Functions

fix()

Takes a real number argument, truncates it, and returns an integer value

Syntax
y = fix(x)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples
a = fix(5.9)
returns 5

Defined in
Built in

7-42   fix()
float()

Converts an integer to a real (floating-point) number

Syntax
y = float(x)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number to convert</td>
<td>(-∞, ∞)</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

```py
a = float(10)
returns 10.000
```

Defined in

Built in

See Also

abs(), cint(), int(), log10(), pow(), sgn(), sqrt()
Math Functions

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

\[ y = \text{floor}(x) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number to convert</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ a = \text{floor}(4.3) \]
returns 4

\[ a = \text{floor}(5.6) \]
returns 5

Defined in

Built in
fmod()

Returns the remainder of the division of two real numbers

Syntax
y = fmod(fNum, fDenom)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>fNum</td>
<td>Value of numerator</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>fDenom</td>
<td>Value of denominator</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

fmodV = fmod(4.2, 2.0)
returns 0.2

Defined In

$\$HPEESOF_DIR\expressions\ael\elementary_fun.ael
Math Functions

**hypot()**
Returns the hypotenuse

**Syntax**
y = hypot(xVal, yVal)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>xVal</td>
<td>Value of X</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>yVal</td>
<td>Value of Y</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**
y = hypot(3, 4)
returns 5

**Defined In**

$HPEESOF_DIR/expressions/ael/elementary_fun.ael$

7-46   hypot()
identity()

Returns the identity matrix

Syntax
\[ y = \text{identity}(\text{rows, columns}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>rows</td>
<td>number of rows †</td>
<td>([i, \infty))</td>
<td>integer</td>
<td>yes</td>
</tr>
<tr>
<td>columns</td>
<td>number of columns †</td>
<td>([i, \infty))</td>
<td>integer</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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

\[ y = \text{identity}(2) \]

<table>
<thead>
<tr>
<th>y(1,1)</th>
<th>y(1,2)</th>
<th>y(2,1)</th>
<th>y(2,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Defined in

Built in

See Also

ones(), zeros()
Math Functions

im()
Returns the imaginary component of a complex number

Syntax
y = im(x)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number to convert</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>data to be integrated</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>start</td>
<td>starting value of the integration</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>first point in the data</td>
<td>no</td>
</tr>
<tr>
<td>stop</td>
<td>stop value of the integration</td>
<td>(-∞, ∞)</td>
<td>integer, real †</td>
<td>last point in the data</td>
<td>no</td>
</tr>
<tr>
<td>incr</td>
<td>increment</td>
<td>[0, ∞)</td>
<td>integer, real</td>
<td>(stop - start)/(# data points - 1)</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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()
Math Functions

**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.

If no incr parameter is given, then no increment (or no interpolation) will be performed.
interp()

Returns linearly interpolated data between start and stop with increment

Syntax

\[ y = \text{interp}(\text{Data}, \text{Start}, \text{Stop}, \text{Increment}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Data to be interpolated</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td>Independent value specifying start</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td>First data point</td>
<td>no</td>
</tr>
<tr>
<td>Stop</td>
<td>Independent value specifying stop</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td>Last data point</td>
<td>no</td>
</tr>
<tr>
<td>Increment</td>
<td>Increment between interpolated data points</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td>(Stop-Start)/NumDataPoints (\dagger)</td>
<td>no</td>
</tr>
</tbody>
</table>

\(\dagger\) first value of independent \(\geq\) Start \(\leq\) last value’s independent

\(\dagger\) Where NumDataPoints is number of original data points

Examples

\[ y = \text{interp}(\text{data}{}, \text{start}, \text{stop}{}, \text{incr}{}) \]

Defined in

Built in

See also

interp()
Math Functions

**interpolate()**
Interpolates data

**Syntax**
y = interpolate(InterpType, Data, iVar1, iVal1, iVar2, iVal2, ..., iVarN, iValN)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>InterpType</td>
<td>Type of interpolation</td>
<td>&quot;linear&quot;, &quot;cubic&quot;, &quot;spline&quot;</td>
<td>string</td>
<td>yes</td>
</tr>
<tr>
<td>Data</td>
<td>Data to interpolate</td>
<td>integer, real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>iVar1, iVar2..</td>
<td>Dimension of independent variable to interpolate</td>
<td>integer</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>iVal1, iVal2..</td>
<td>values to interpolate</td>
<td>integer, real</td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**
linI = interpolate("linear", colY, 1, [1::0.5::4])
cubicI = interpolate("cubic", colY, 1, [1::0.2::2])
splineI = interpolate("spline", colY, 1, [-2::0.2::2])

For a 2-D data with 2 independents, the following interpolates dimension 1 (inner-most) from 1 to 4 in steps of 0.5 and second dimension at 0.5:
linI = interpolate("linear", colY, 1, [1::0.5::4], 2, 0.5)

**Defined in**
Built in

**See also**
interp()

**Notes/Equations**
This function can interpolate data over multiple dimensions.
inverse()

Returns the inverse of a matrix

Syntax
y = inverse(Matrix)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>square matrix to find the inverse</td>
<td>(-∞, ∞)</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples
inverse({{1, 2}, {3, 4}})
returns {{–2, 1}, {1.5, –0.5}}

Defined in
Built in

See Also
diagonal(), transpose()
Math Functions

jn()

Computes the Bessel function of the first kind and returns a real number

Syntax
y =jn(n, x)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>order</td>
<td>(-∞, ∞)</td>
<td>integer</td>
<td>yes</td>
</tr>
<tr>
<td>x</td>
<td>value for which the</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>Bessel value is to be found</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
In()  
Returns the natural logarithm (ln)

Syntax  
y = ln(x)

Arguments  

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples  
a = ln(e)  
returns 1

Defined in  
Built in

See Also  
abs(), cint(), exp(), float(), int(), pow(), sgn(), sqrt()
Math Functions

**log()**

Returns the base 10 logarithm of an integer or real number

**Syntax**

`y = \log(x)`

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
a = \log(10)
returns 1
```

```plaintext
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

\[ y = \log_{10}(x) \]

Arguments

- **x**: number

Examples

- \( a = \log_{10}(10) \)
  returns 1

- \( a = \log_{10}(0+0i) \)
  returns NULL and an error message "log of zero"

Defined in

- Built in

See Also

- abs(), cint(), exp(), float(), int(), log(), pow(), sgn(), sqrt()
Math Functions

**mag()**

Returns the magnitude

**Syntax**

\[ y = \text{mag}(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{mag}(3-4j) \]

returns 5.000

**Defined in**

Built in

**See Also**

`conj()`
**max()**

Returns the maximum value

**Syntax**

\[ y = \text{max}(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to find max</td>
<td>((-\infty, \infty))</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{max}([1, 2, 3]) \]

returns 3

**Defined in**

Built in

**See Also**

`cum_prod()`, `cum_sum()`, `max2()`, `mean()`, `min()`, `prod()`, `sum()`
Math Functions

**max_outer()**

Computes the maximum across the outer dimension of two-dimensional data

**Syntax**

```plaintext
y = max_outer(data)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>2-dimensional data to find max</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
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
\[ y = \text{max2}(x, y) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-\infty, \infty)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>y</td>
<td>number</td>
<td>(-\infty, \infty)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ a = \text{max2}(1.5, -1.5) \]
returns 1.500

Defined in
Built in

See Also
\textit{cumprod()}, \textit{cumsum()}, \textit{max()}, \textit{mean()}, \textit{min()}, \textit{prod()}, \textit{sum()}
Math Functions

**min()**

Returns the minimum value

**Syntax**

\[ y = \text{min}(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to find min</td>
<td>(( -\infty, \infty ))</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{min}([1, 2, 3]) \]

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**

\[ y = \text{min}\_\text{outer}(\text{data}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>2-dimensional data to find min</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{min}\_\text{outer}(\text{data}) \]

**Defined in**

$\text{HPEESOF}_{\text{DIR}}/\text{expressions/ael/statistical\_fun.ael}$

**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 \([151 \text{ Monte Carlo iteration } \times 26 \text{ frequency}]\) 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.
Math Functions

**min2()**

Returns the lesser value of two numeric values, or NULL if parameters are invalid

**Syntax**
y = min2(x, y)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>y</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>string to convert to integer</td>
<td>string</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

a = num("/users/myhome/fullpath")
returns 47

a = num("alpha")
returns 97

Defined in

Built in
ones()

Returns ones matrix

Syntax

\[ y = \text{ones}(\text{rows, columns}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>rows</td>
<td>number of rows</td>
<td>([i, \infty))</td>
<td>integer</td>
<td>yes</td>
</tr>
<tr>
<td>columns</td>
<td>number of columns</td>
<td>([i, \infty))</td>
<td>integer</td>
<td>no</td>
</tr>
</tbody>
</table>

† If only one argument is supplied, then a square matrix is returned. If two are supplied, then a matrix of ones with size \(\text{rows} \times \text{cols}\) is returned

Examples

```plaintext
a = \text{ones}(2)
returns \{\{1, 1\}, \{1, 1\}\}
```

Defined in

Built in

See Also

identity(), zeros()
phase()
Phase in degrees

Syntax
y = phase(x)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples
a = phase(1*i)
returns 90

a = phase(1+1i)
returns 45

Defined in
Built-in

See Also
phaserad()
Math Functions

phasedeg()
Phase in degrees

Syntax
y = phasedeg(x)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
a = phaserad(1*i)
returns 1.5708
```

```plaintext
a = phaserad(1+1i)
returns 0.785398
```

**Defined in**

Built in

**See Also**

dev_lin_phase(), diff(), phase(), phasedeg(), ripple(), unwrap()
Math Functions

**polar()**

Builds a complex number from magnitude and angle (in degrees)

**Syntax**

\[ y = \text{polar}(\text{mag}, \text{angle}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>mag</td>
<td>magnitude part of complex number</td>
<td>((-\infty, \infty))</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>angle</td>
<td>angle part of complex number</td>
<td>((-\infty, \infty))</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[
\begin{align*}
a &= \text{polar}(1, 90) \\
&= 0 + 1i \\
a &= \text{polar}(1, 45) \\
&= 0.707107 + 0.707107i
\end{align*}
\]

**Defined in**

Built in
**pow()**

Raises a number to a given power

**Syntax**
yPow = pow(x, y)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>y</td>
<td>exponent</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
a = pow(4, 2)
returns 16
```

```plaintext
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()
Math Functions

**prod()**

Returns the product

**Syntax**

\[ y = \text{prod}(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to find product</td>
<td>((-\infty, \infty))</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{prod}([1, 2, 3]) \]

returns 6

\[ a = \text{prod}([4, 4, 4]) \]

returns 64

**Defined in**

Built-in

**See Also**

*sum()*
rad()

Converts degrees to radians

Syntax
y = rad(x)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

a = rad(90)
returns 1.5708

a = rad(45)
returns 0.785398

Defined in

Built in

See Also

deg()
Math Functions

\textbf{re()}

Returns the real component of a complex number

\textbf{Syntax}
\[ y = \text{re}(x) \]

\textbf{Arguments}

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

\textbf{Examples}
\[ a = \text{re}(1-1*\text{j}) \]
returns 1.000

\textbf{Defined in}
Built in

\textbf{See Also}
\texttt{cmplx()}, \texttt{imag()}, \texttt{real()}

7-76 \texttt{re()}
**real()**

Returns the real component of a complex number

**Syntax**

```plaintext
y = real(x)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
a = real(1-j)
returns 1.000
```

**Defined in**

Built in

**See Also**

`cmplx()`, `imag()`, `re()`
Math Functions

**rms()**

Returns the root mean square value

**Syntax**

```
y = rms(Value)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Value to find RMS</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```
rmsR = rms(2)  
returns 1.414

r = 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**

$\text{HPEESOF_DIR/expressions/ael/elementary_fun.ael}$

**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
\[ y = \text{round}(x) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

- \[ a = \text{round}(0.1) \]
  returns 0
- \[ a = \text{round}(0.5) \]
  returns 1
- \[ a = \text{round}(0.9) \]
  returns 1
- \[ a = \text{round}(-0.1) \]
  returns 0
- \[ a = \text{round}(-0.5) \]
  returns -1
- \[ a = \text{round}(-0.9) \]
  returns -1

Defined in
Built in

See Also
int()
Math Functions

sgn()
Returns the integer sign of an integer or real number, as either 1 or -1

Syntax
\[ y = sgn(x) \]

Arguments

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number in radians</td>
<td>((-\infty, \infty))</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \sin(\pi/2) \]

returns 1

**Defined in**

Built in

**See Also**

\[ \cos(), \tan() \]
Math Functions

**sinc()**

Returns the sinc of a number

**Syntax**

```math
y = sinc(x)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```math
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(πx) / (πx) and sinc(0) = 1.
**sinh()**

Returns the hyperbolic sine

**Syntax**

\[ y = \sinh(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \sinh(0) \]
returns 0

\[ a = \sinh(1) \]
returns 1.1752

**Defined in**

Built in

**See Also**

cosh(), tanh()
Math Functions

**sqr()**
Returns the square of a number

**Syntax**
y = sqr(x)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number to square</td>
<td>$(-\infty, \infty)$</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

\[ y = \text{sum}(x) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to find sum</td>
<td>((-\infty, \infty))</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ a = \text{sum}([1, 2, 3]) \]
returns 6

Defined in

Built in

See Also

\[ \text{max}(), \text{mean}(), \text{min}() \]
Math Functions

**tan()**
Returns the tangent

**Syntax**
y = tan(x)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number in radians</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number in radians</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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()
**Math Functions**

**transpose()**
Returns the transpose of a matrix

**Syntax**
y = transpose(Matrix)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>square matrix to find the transpose</td>
<td>(-∞, ∞)</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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

\[ \text{dbmVal} = \text{wtodbm(Value)} \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Value in Watts</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[
\begin{align*}
\text{wtodbm01}_M &= \text{wtodbm}(0.01) \\
&\text{returns } 10 \\
\text{wtodbm1}_M &= \text{wtodbm}(1) \\
&\text{returns } 30 \\
\text{wtodbmC}_M &= \text{wtodbm}(\text{complex}(10,2)) \\
&\text{returns } 40.094/1.225
\end{align*}
\]

Defined in

\$\text{HPESOF_DIR/expressions/ael/elementary_fun.ael}\$

See Also

dbmtow()
Math Functions

xor()
Returns an integer that represents the exclusive OR between arguments

Syntax
yXor = xor(x, y)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer</td>
<td>yes</td>
</tr>
<tr>
<td>y</td>
<td>number</td>
<td>(-∞, ∞)</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ a = \text{xor}(16, 32) \]
\[ \text{returns } 48 \]

Defined in

Built in
zeros()
Returns zeros matrix

Syntax
\[ y = \text{zeros}(\text{rows}, \text{columns}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>rows</td>
<td>number of rows †</td>
<td>([i, \infty))</td>
<td>integer</td>
<td>yes</td>
</tr>
<tr>
<td>columns</td>
<td>number of columns †</td>
<td>([i, \infty))</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

† 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

\[ a = \text{zeros}(2) \]
returns \([[0, 0], [0, 0]]\)

\[ b = \text{zeros}(2, 3) \]
returns \([[0, 0, 0], [0, 0, 0]]\)

Defined in
Built in
See Also
identity(), ones()
Math Functions

7.94 zeros()
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()” on page 8-2
“ber_pi4dqpsk()” on page 8-3
“ber_qpsk()” on page 8-5
“delay_path()” on page 4-17
“evm_wlan_dsss_cck_pbcc()” on page 4-18
“evm_wlan_ofdm()” on page 4-28
“eye()” on page 8-7
“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
“peak_pwr()” on page 4-42
“peak_to_avg_pwr()” on page 4-44
“power_ccdf()” on page 4-46
“power_ccdf_ref()” on page 4-48
“pwr_vs_t()” on page 4-50
“sample_delay_pi4dqpsk()” on page 4-53
“sample_delay_qpsk()” on page 4-54
“spec_power()” on page 8-18
“total_pwr()” on page 4-61
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

\[ y = \text{add_rf}(T1, T2) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Timed Complex Envelope signals at carrier frequencies Fc1</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>T2</td>
<td>Timed Complex Envelope signals at carrier frequencies Fc2</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ y = \text{add_rf}(T1, T2) \]

Defined in

$\text{HPEESOF_DIR/expressions/ael/signal_proc_fun.ael}$

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vIn</td>
<td>complex envelope voltage signals at the input node</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>vOut</td>
<td>complex envelope voltage signals at the output node</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>symRate</td>
<td>symbol rate (real) of the modulation signal</td>
<td>(0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>noise</td>
<td>RMS noise vector</td>
<td>(0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>samplingDelay</td>
<td>clock phase in seconds</td>
<td>(0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>rotation</td>
<td>carrier phase in radians</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>tranDelay</td>
<td>time in seconds that causes this time duration of symbols to be eliminated from the bit error rate calculation †</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>pathDelay</td>
<td>delay from input to output in seconds</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

† 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(vIn[1], vOut[1], 0.5e6, 0.1:0.01:0.02)
Signal Processing Functions

Defined in

$\text{HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael}$

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, $E_b/N_0$ 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:

\[
\begin{align*}
\text{data}[0] &= \text{symbol probability of error versus } E_b/N_0 \\
\text{data}[1] &= \text{path delay in seconds} \\
\text{data}[2] &= \text{carrier phase in radians} \\
\text{data}[3] &= \text{clock phase in seconds} \\
\text{data}[4] &= \text{complex(Isample, Qsample)}
\end{align*}
\]
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vIn</td>
<td>complex envelope voltage signals at the input node</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>vOut</td>
<td>complex envelope voltage signals at the output node</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>symRate</td>
<td>symbol rate (real) of the modulation signal</td>
<td>(0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>noise</td>
<td>RMS noise vector</td>
<td>(0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>samplingDelay</td>
<td>clock phase in seconds</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>rotation</td>
<td>carrier phase in radians</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>tranDelay</td>
<td>time in seconds that causes this time duration of symbols to be eliminated from the bit error rate calculation †</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>pathDelay</td>
<td>delay from input to output in seconds</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

† 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))
Signal Processing Functions

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>either numeric data or a time domain waveform typically from the I or Q data channel</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>symbolRate</td>
<td>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</td>
<td>(0, ∞)</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>Cycles</td>
<td>number of cycles to repeat</td>
<td>[1, ∞)</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>Delay</td>
<td>sampling delay</td>
<td>[0, ∞)</td>
<td>integer, real</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

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()
**Signal Processing Functions**

**eye_amplitude()**

Returns eye amplitude

**Syntax**

\[ y = \text{eye_amplitude}(Vout\_time, \text{Delay}, \text{BitRate}, \text{SamplePoint}, \text{WindowPct}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vout_time</td>
<td>time domain voltage waveform</td>
<td>(-(\infty), (\infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>Delay</td>
<td>used to remove initial transient in the eye diagram and is expressed in time units</td>
<td>[0, (\infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>BitRate</td>
<td>bit rate of the channel and is expressed in frequency units</td>
<td>[0, (\infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>SamplePoint</td>
<td>marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.</td>
<td></td>
<td>string</td>
<td>yes</td>
</tr>
<tr>
<td>WindowPct</td>
<td>used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.</td>
<td>[0, 1]</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ \text{Eye} \_ \text{amp} = \text{eye} \_ \text{amplitude}(\text{vout}, 12 \ \text{ps}, 10 \ \text{GHz}, \text{m1}, 0.2) \]

**Defined in**

$HPEESOF\_DIR/expressions/ael/DesignGuide\_fun.ael$

**See Also**

\[ \text{cross_hist()}, \text{eye()}, \text{eye_closure()}, \text{eye_fall_time()}, \text{eye_height()}, \text{eye_rise_time()} \]
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.
Signal Processing Functions

**eye_closure()**

Returns eye closure

**Syntax**
y = eye_closure(Vout_time, Delay, BitRate, SamplePoint, WindowPct)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vout_time</td>
<td>time domain voltage waveform</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>Delay</td>
<td>used to remove initial transient in the eye diagram and is expressed in time units</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>BitRate</td>
<td>bit rate of the channel and is expressed in frequency units</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>SamplePoint</td>
<td>marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.</td>
<td>string</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>WindowPct</td>
<td>used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.</td>
<td>[0, 1]</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

Eye_Close = eye_closure(vout,12 ps,10 GHz, m1, 0.2)

**Defined in**

$HPEE5OF_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
\[ y = \text{eye\_fall\_time}(Vout\_time, \text{Delay}, \text{BitRate}, \text{SamplePoint}, \text{WindowPct}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vout_time</td>
<td>time domain voltage waveform</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>Delay</td>
<td>used to remove initial transient in the eye diagram and is expressed in time units</td>
<td>[0, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>BitRate</td>
<td>bit rate of the channel and is expressed in frequency units</td>
<td>[0, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>SamplePoint</td>
<td>marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.</td>
<td></td>
<td>string</td>
<td>yes</td>
</tr>
<tr>
<td>WindowPct</td>
<td>used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.</td>
<td>[0, 1]</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[
\text{Eye\_Fall} = \text{eye\_fall\_time}(\text{vout}, 12 \text{ ps}, 10 \text{ GHz}, \text{m1}, 0.2)
\]

Defined in

$\text{HPEESOF\_DIR/expressions/ael/DesignGuide\_fun.ael}$

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.
Signal Processing Functions

**eye_height()**

Returns eye height

**Syntax**

\[ y = \text{eye_height}(Vout\_time, \text{Delay}, \text{BitRate}, \text{SamplePoint}, \text{WindowPct}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vout_time</td>
<td>time domain voltage waveform</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>Delay</td>
<td>used to remove initial transient in the eye diagram and is expressed in time units</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>BitRate</td>
<td>bit rate of the channel and is expressed in frequency units</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>SamplePoint</td>
<td>marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.</td>
<td>string</td>
<td>string</td>
<td>yes</td>
</tr>
<tr>
<td>WindowPct</td>
<td>used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.</td>
<td>[0, 1]</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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.
Signal Processing Functions

**eye_rise_time()**

Returns eye rise time

**Syntax**

\[
y = \text{eye_rise_time}(\text{Vout\_time}, \text{Delay}, \text{BitRate}, \text{SamplePoint}, \text{WindowPct})
\]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vout_time</td>
<td>time domain voltage waveform</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>Delay</td>
<td>used to remove initial transient in the eye diagram and is expressed in time units</td>
<td>[0, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>BitRate</td>
<td>bit rate of the channel and is expressed in frequency units</td>
<td>[0, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>SamplePoint</td>
<td>marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.</td>
<td>string</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>WindowPct</td>
<td>used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.</td>
<td>[0, 1]</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

Eye_Rise = eye_rise_time(vout, 12 ps, 10 GHz, m1, 0.2)

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/DesignGuide\_fun.ael}$

**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**

```plaintext
y = spec_power(spectralData{, lowerFrequencyLimit, upperFrequencyLimit})
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>spectralData</td>
<td>spectral data in dBm</td>
<td>real</td>
<td></td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>lowerFrequencyLimit</td>
<td>lower frequency limit to be used in calculating the integrated power</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>min(indep(spectralData))</td>
<td>no</td>
</tr>
<tr>
<td>upperFrequencyLimit</td>
<td>upper frequency limit to be used in calculating the integrated power</td>
<td>[lowerFrequencyLimit, ∞)</td>
<td>real</td>
<td>max(indep(spectralData))</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
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.

```plaintext
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**

`$HPEEESOF_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.
Signal Processing Functions

8-20 spec_power()
Chapter 9: S-Parameter Analysis Functions

This chapter describes the S-parameter analysis functions in detail. The functions are listed in alphabetical order.

A, B, C, D, G
“abcdtoh()” on page 9-3  “dev_lin_phase()” on page 9-11
“abcdtos()” on page 9-4  “ga_circle()” on page 9-12
“abcdtoy()” on page 9-5  “gain_comp()” on page 9-14
“abcdtoz()” on page 9-6  “gl_circle()” on page 9-15
“bandwidth_func()” on page 9-7  “gp_circle()” on page 9-17
“center_freq()” on page 9-9  “gs_circle()” on page 9-19
“dev_lin_gain()” on page 9-10

H, I, L, M
“htoabcd()” on page 9-21  “l_stab_region()” on page 9-28
“htos()” on page 9-22  “map1_circle()” on page 9-29
“htoy()” on page 9-23  “map2_circle()” on page 9-30
“htoz()” on page 9-24  “max_gain()” on page 9-31
“ispec()” on page 9-25  “mu()” on page 9-32
“l_stab_circle()” on page 9-26  “mu_prime()” on page 9-33
“l_stab_circle_center_radius()” on page 9-27

N, P, R
“ns_circle()” on page 9-34  “phase_comp()” on page 9-38
“ns_pwr_int()” on page 9-36  “pwr_gain()” on page 9-39
“ns_pwr_ref_bw()” on page 9-37  “ripple()” on page 9-40

S
“s_stab_circle()” on page 9-41  “stab_fact()” on page 9-50
“s_stab_circle_center_radius()” on page 9-42  “stab_meas()” on page 9-51
S-Parameter Analysis Functions

“s_stab_region()” on page 9-43
“sm_gamma1()” on page 9-44
“sm_gamma2()” on page 9-45
“sm_y1()” on page 9-46
“sm_y2()” on page 9-47
“sm_z1()” on page 9-48
“sm_z2()” on page 9-49

T,U,V,W,Y
“tdr_sp_gamma()” on page 9-59
“tdr_sp_imped()” on page 9-61
“tdr_step_imped()” on page 9-63
“ttos()” on page 9-64
“unilateral_figure()” on page 9-65
“unwrap()” on page 9-67
“v_dc()” on page 9-68
“volt_gain()” on page 9-69
“volt_gain_max()” on page 9-71

Z
“zin()” on page 9-81
“zopt()” on page 9-82
“ztoabcd()” on page 9-83

“stoabcd()” on page 9-52
“stoh()” on page 9-53
“stos()” on page 9-54
“stot()” on page 9-56
“stoy()” on page 9-57
“stoz()” on page 9-58

“vswr()” on page 9-72
“write_snp()” on page 9-73
“yin()” on page 9-75
“yopt()” on page 9-76
“ytaabcd()” on page 9-77
“ytoh()” on page 9-78
“ytos()” on page 9-79
“ytoz()” on page 9-80

“ztoh()” on page 9-84
“ztos()” on page 9-85
“ztoabcd()” on page 9-83
“ztoy()” on page 9-86
abcdtoh()

This measurement transforms the chain (ABCD) matrix of a 2-port network to a hybrid matrix

Syntax

\[ h = \text{abcdtoh}(a) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>chain (ABCD) matrix of a 2-port network</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ h = \text{abcdtoh}(a) \]

Defined in

$\text{HPEE SOF_DIR/expressions/ael/network_fun.ael}$

See Also

abcdtos(), abcdtoy(), abcdtoz(), htoabcd(), stoabcd(), ytoabcd(), ztoabcd()
S-Parameter Analysis Functions

abcdtos()

His measurement transforms the chain (ABCD) matrix of a 2-port network to a scattering matrix

Syntax
sp = abcdtos(A, zRef)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>chain (ABCD) matrix of a 2-port network</td>
<td>$(-\infty, \infty)$</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>$(-\infty, \infty)$</td>
<td>integer, real or complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>chain (ABCD) matrix of a 2-port network</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples
y = abcdtoy(a)

Defined in
$HPEE SOF_DIR/expressions/ael/network_fun.ael

See Also
abcdtoh(), abcdtos(), abcdtoz(), htoabcd(), stoabcd(), ytoabcd(), ztoabcd()
S-Parameter Analysis Functions

`abcdtoz()`

This measurement transforms the chain (ABCD) matrix of a 2-port network to impedance matrix

**Syntax**

```
z = abcdtoz(a)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>chain (ABCD) matrix of a 2-port network</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>data (usually gain) to find the bandwidth</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>DesiredValue</td>
<td>A single value representing the desired bandwidth level.</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>Type</td>
<td>Type of response</td>
<td>[0, 3] †</td>
<td>integer</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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

$HPEE SOF_DIR/expressions/ael/ rf_system_fun.ael

See Also

center_freq()
S-Parameter Analysis Functions

Data can be from 1 to 4 dimensions.
center_freq()

Returns the center frequency at the specified level as a real number.

Syntax

\[ fc = \text{center_freq}(\text{Data}, \text{ReferenceBW}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>data (usually gain) to find the center frequency</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>ReferenceBW</td>
<td>a single value representing the reference bandwidth</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

This example assumes that an S-parameter analysis has been performed.

\[ fc3 = \text{center_freq}(\text{db}(S21), 3) \]

returns the center frequency using the 3dB point as reference.

Defined in

$\text{HPEESOF_DIR/expressions/ael/rf_system_fun.ael}$

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.
S-Parameter Analysis Functions

**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**

```plaintext
y = dev_lin_gain(voltGain)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltGain</td>
<td>gain as a function of frequency</td>
<td>[0, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
a = dev_lin_gain(volt_gain(S, PortZ(1), PortZ(2)))
```

**Defined in**

$HPEESOF_DIR/expressions/acl/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**

```matlab
y = dev_lin_phase(voltGain)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltGain</td>
<td>function of frequency</td>
<td>[0, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```matlab
a = dev_lin_phase(S21)
```

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/rf\_system\_fun.ael}$

**See Also**

`dev_lin_gain(), diff(), phasdeg(), 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.
S-Parameter Analysis Functions

**ga_circle()**
Generates an available gain circle

**Syntax**
y = ga_circle(S, gain, numOfPts, numCircles, gainStep)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network.</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>gain</td>
<td>specified gain in dB</td>
<td>[0, ∞)</td>
<td>integer or real array</td>
<td>†</td>
<td>no</td>
</tr>
<tr>
<td>numOfPts</td>
<td>desired number of points per circle</td>
<td>[1, ∞)</td>
<td>integer</td>
<td>51</td>
<td>no</td>
</tr>
<tr>
<td>numCircles</td>
<td>number of desired circles. This is used if gain is not specified.</td>
<td>[0, ∞)</td>
<td>integer</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>gainStep</td>
<td>gain step size. This is used if gain is not specified.</td>
<td>[0, ∞)</td>
<td>integer or real</td>
<td>1.0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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, [51, 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).

\[
Ca(\text{Center}) = \frac{gaC_1^*}{(1 + ga(|S_{11}|^2 - |\Delta|^2))}
\]

\[
Ra(\text{radius}) = \frac{\sqrt{1 - 2K |S_{12}S_{21}| ga + |S_{12}S_{21}|^2 ga^2}}{|1 + ga(|S_{11}|^2 - |\Delta|^2)|}
\]

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sji</td>
<td>Sji is a power-dependent complex transmission coefficient obtained from large-signal S-parameter simulation.</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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**

\[ y = \text{gl\_circle}(S, \text{gain}, \text{numOfPts}, \text{numCircles}, \text{gainStep}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2-port network.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gain</td>
<td>specified gain in dB</td>
<td>([0, \infty)]</td>
<td>integer or real</td>
<td>maxGain - ([0, 1, 2, 3])</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>numOfPts</td>
<td>desired number of points per circle</td>
<td>([1, \infty)]</td>
<td>integer</td>
<td>51</td>
<td>no</td>
</tr>
<tr>
<td>numCircles</td>
<td>number of desired circles. This is used if gain is not specified.</td>
<td>([0, \infty)]</td>
<td>integer</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>gainStep</td>
<td>gain step size. This is used if gain is not specified.</td>
<td>([0, \infty)]</td>
<td>integer or real</td>
<td>1.0</td>
<td>no</td>
</tr>
</tbody>
</table>

† Where maxGain = 10*\log(1 / (1 - \text{mag}(S22)^2))

**Examples**

\[ \text{circleData} = \text{gl\_circle}(S, 2, 51) \]

\[ \text{circleData} = \text{gl\_circle}(S, \{2, 3, 4\}, 51) \]

return the points on the circle(s).

\[ \text{circleData} = \text{gl\_circle}(S, , 51, 5, 0.5) \]

return the points on the circle(s) for 5 circles at maxGain - \([0, 0.5, 1.0, 1.5, 2.0]\)

\[ \text{circleData} = \text{gl\_circle}(S, , 2, 1.0) \]

return the points on the circle(s) for 2 circles at maxGain - \([0, 1.0]\)

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/circle\_fun.ael}$
S-Parameter Analysis Functions

See Also

gl_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.

\[
\begin{align*}
\text{Cl}(\text{Center}) &= \frac{(1 - |S_{22}|^2 \text{Gabs})|S_{22}|^*}{1 - |S_{22}|^2 (1 - |S_{22}|^2 \text{Gabs})} \\
\text{RI}(\text{radius}) &= \sqrt[4]{1 - (1 - |S_{22}|^2 \text{Gabs})(1 - |S_{22}|^2)} \\
\text{Where Gabs is the absolute gain, and is given by:} \\
\text{Gabs} &= (10^{\text{dB Gain}})/10.0
\end{align*}
\]
gp_circle()

Generates a power gain circle

Syntax

\[ y = \text{gp\_circle}(S, \text{gain}, \text{numOfPts}, \text{numCircles}, \text{gainStep}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network.</td>
<td>((-∞, ∞))</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>gain</td>
<td>specified gain in dB</td>
<td>([0, ∞))</td>
<td>integer or real array</td>
<td>† no</td>
<td></td>
</tr>
<tr>
<td>numOfPts</td>
<td>desired number of points per circle</td>
<td>([1, ∞))</td>
<td>integer</td>
<td>51</td>
<td>no</td>
</tr>
<tr>
<td>numCircles</td>
<td>number of desired circles. This is used if</td>
<td>([0, ∞))</td>
<td>integer</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gain is not specified.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gainStep</td>
<td>gain step size. This is used if gain is</td>
<td>([0, ∞))</td>
<td>integer or real</td>
<td>1.0</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>not specified.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† Default value for gain is \(\min(\max(\text{gain}(S))) - \{1, 2, 3\}\)

Examples

\[
\text{circleData} = \text{gp\_circle}(S, 2, 51) \\
\text{circleData} = \text{gp\_circle}(S, \{2, 3, 4\}, 51) \\
\text{return the points on the circle(s)}
\]

\[
\text{circleData} = \text{gp\_circle}(S, 51, 5, 0.5) \\
\text{return the points on the circle(s) for 5 circles at maxGain - \{0,0.5,1.0,1.5,2.0\}}
\]

\[
\text{circleData} = \text{gp\_circle}(S, 2, 1.0) \\
\text{return the points on the circle(s) for 2 circles at maxGain - \{0,1.0\}}
\]

Defined in

$\text{HPEESOF_DIR/expressions/ael/circle\_fun.ael}$

See Also

gp\_circle(), \text{ga\_circle()}, \text{gl\_circle()}, \text{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 \( \max\_gain(S) \).

\[
C_p(\text{Center}) = \frac{\text{gp} C_2^*}{(1 + \text{gp}(|S_{22}|^2 - |\Delta|^2))}
\]

\[
R_p(\text{radius}) = \sqrt{\frac{1 - 2K|S_{12}S_{21}| \text{gp} + |S_{12}S_{21}|^2 \text{gp}^2}{1 + \text{gp}(|S_{22}|^2 - |\Delta|^2)}}
\]

Where:

\( K \) = Stability Factor

\( \text{gp}(G) = \frac{\text{Gp}}{|S_{21}|^2} \) (Gp is Desired Gain in absolute terms)

\( C_2 = S_{22} - |\Delta|^2 S_{11}^* \)
gs_circle()

Returns a source-mismatch gain circle

Syntax

\[ y = \text{gs\_circle}(S, \text{gain}, \text{numOfPts}, \text{numCircles}, \text{gainStep}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network.</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>gain</td>
<td>specified gain in dB</td>
<td>[0, \infty)</td>
<td>integer or real array</td>
<td>maxGain - {0, 1, 2, 3} †</td>
<td>no</td>
</tr>
<tr>
<td>numOfPts</td>
<td>desired number of points per circle</td>
<td>[1, \infty)</td>
<td>integer</td>
<td>51</td>
<td>no</td>
</tr>
<tr>
<td>numCircles</td>
<td>number of desired circles. This is used if gain is not specified.</td>
<td>[0, \infty)</td>
<td>integer</td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>gainStep</td>
<td>gain step size. This is used if gain is not specified.</td>
<td>[0, \infty)</td>
<td>integer or real</td>
<td>1.0</td>
<td>no</td>
</tr>
</tbody>
</table>

† Where \( \text{maxGain} = 10 \log(1 / (1 - \text{mag}(S11)^2)) \)

Examples

\[ \text{circleData} = \text{gs\_circle}(S, 2, 51) \]
\[ \text{circleData} = \text{gs\_circle}(S, \{2, 3, 4\}, 51) \]
\[ \text{return the points on the circle(s)} \]

\[ \text{circleData} = \text{gs\_circle}(S, , 51, 5, 0.5) \]
\[ \text{return the points on the circle(s) for 5 circles at maxGain - \{0,0.5,1.0,1.5,2.0\}} \]

\[ \text{circleData} = \text{gs\_circle}(S, , 2, 1.0) \]
\[ \text{return the points on the circle(s) for 2 circles at maxGain - \{0,1.0\}} \]

Defined in

\$\text{HPEESOF\_DIR/expressions/ael/circle\_fun.ael}$
S-Parameter Analysis Functions

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`.

\[
Cs(Center) = \frac{(1 - |S_{11}|^2 Gabs)|S_{11}|^*}{1 - |S_{11}|^2 (1 - |S_{11}|^2 Gabs)}
\]

\[
Rs(radius) = \sqrt{\frac{1 - (1 - |S_{11}|^2 Gabs)(1 - |S_{11}|^2)}{1 - |S_{11}|^2 (1 - |S_{11}|^2 Gabs)}}
\]

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

\[ a = \text{htoabcd}(H) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>hybrid matrix of a 2-port network</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ a = \text{htoabcd}(H) \]

Defined in

$\text{HPEESOF_DIR/expressions/ael/network_fun.ael}$

See Also

\[ \text{abcdtoh()}, \text{htoz()}, \text{ytoh()} \]
S-Parameter Analysis Functions

htos()

This measurement transforms the hybrid matrix of a 2-port network to a scattering matrix.

Syntax

\[ sp = \text{htos}(h, \text{zRef}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>hybrid matrix of a 2-port network</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>((-\infty, \infty))</td>
<td>integer, real or complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

Examples

\[ s = \text{htos}(h, 50) \]

Defined in

$\text{HPEESOF\_DIR/expressions/ael/network\_fun.ael}$

See Also

htoy(), htoz(), stoh()
htoy()

This measurement transforms the hybrid matrix of a 2-port network to an admittance matrix.

Syntax

\[ y = \text{htoy}(H) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>hybrid matrix of a 2-port network</td>
<td>(-(\infty), (\infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ y = \text{htoy}(H) \]

Defined in

$\text{HPEESOF_DIR/expressions/ael/network_fun.ael}$

See Also

htos(), htoz(), ytoh()
S-Parameter Analysis Functions

htoz()

This measurement transforms the hybrid matrix of a 2-port network to an impedance matrix.

Syntax

\[ z = \text{htoz}(H) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>hybrid matrix of a 2-port network</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ z = \text{htoz}(H) \]

Defined in

$\text{HPSOF_DIR/expressions/ael/network_fun.ael}$

See Also

htos(), htoy(), ytoh()
ispec()

Returns the current frequency spectrum

Syntax
y = ispec(current)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>current</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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.
S-Parameter Analysis Functions

\textbf{l stab circle()}

Returns load (output) stability circles

\textbf{Syntax}

\[ y = \text{l stab circle}(S, \text{numOfPts}) \]

\textbf{Arguments}

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network.</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>numOfPts</td>
<td>desired number of points per circle</td>
<td>[1, \infty)</td>
<td>integer</td>
<td>51</td>
<td>no</td>
</tr>
</tbody>
</table>

\textbf{Examples}

\[ \text{circleData} = \text{l stab circle}(S, 51) \]

returns the points on the circle(s)

\textbf{Defined in}

$\text{HPEESOF_DIR/expressions/ael/circle_fun.ael}$

\textbf{See Also}

\text{l stab circle center radius()}, \text{l stab region()}, \text{s stab circle()}, \text{s stab region()}

\textbf{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 \text{l stab circle center radius()} to find the center and radius of the stability circle.

Use the function \text{l stab region}(S) to determine the region of stability.
\texttt{l\_stab\_circle\_center\_radius()}

Returns the center and radius of the load (output) stability circle

\textbf{Syntax}

\texttt{l\_cr = l\_stab\_circle\_center\_radius(S, Type)}

\textbf{Arguments}

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>2-port S-Parameters</td>
<td>complex</td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Type of parameter to return</td>
<td></td>
<td>string</td>
<td>&quot;both&quot;</td>
<td>no</td>
</tr>
</tbody>
</table>

\textbf{Examples}

\texttt{l\_cr = l\_stab\_circle\_center\_radius(S)}
returns a 1X2 matrix containing center and radius

\texttt{lCirc = expand(circle(l\_cr(1), real(l\_cr(2)), 51))}
returns data for the load stability circle

\texttt{l\_radius = l\_stab\_circle\_center\_radius(S, "radius")}
returns the radius

\textbf{Defined in}

$\text{HPEESOF\_DIR/expressions/ael/circle\_fun.ael}$

\textbf{See Also}

\texttt{l\_stab\_circle()}, \texttt{s\_stab\_circle()}, \texttt{s\_stab\_circle\_center\_radius()}

\textbf{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.
S-Parameter Analysis Functions

**l_stab_region()**

This expression returns a string identifying the region of stability of the corresponding load stability circle.

**Syntax**

```matlab
y = l_stab_region(S)
```

**Arguments**

- **S**: Scattering matrix of a 2-port network.

**Range**: \((-\infty, \infty)\)

**Type**: complex

**Required**: yes

**Examples**

```matlab
region = l_stab_region(S)
returns “Outside” or “Inside”
```

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/circle\_fun.ael}$

**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 2 to port 1

Syntax

circleData = map1_circle(S, numOfPts)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network.</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>numOfPts</td>
<td>desired number of points per circle</td>
<td>[1, ∞)</td>
<td>integer</td>
<td>51</td>
<td>no</td>
</tr>
</tbody>
</table>

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 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.
map2_circle()

Returns load-mapping circles from port 1 to port 2

Syntax

circleData = map2_circle(S, numOfPts)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network.</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>numOfPts</td>
<td>desired number of points per circle</td>
<td>([1, \infty))</td>
<td>integer</td>
<td>51</td>
<td>no</td>
</tr>
</tbody>
</table>

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 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.
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

\[ y = \text{max\_gain}(S) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of 2-port network</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ y = \text{max\_gain}(S) \]

Defined in

$\text{HPEESOF\_DIR/expressions/ael/rf\_system\_fun.ael}$

See Also

sm\_gamma1(), sm\_gamma2(), stab\_fact(), stab\_meas()

Notes/Equations

This function should be used in a stable system, i.e. when the stability factor (k) is greater than 1. The function actually works for both \(k > 1\) and \(k \leq 1\). If \(k > 1\), then \(k\) is used in the max gain calculations. However, if \(k \leq 1\), then \(k = 1\) is used in the max gain calculations.
S-Parameter Analysis Functions

\textbf{\texttt{mu()}}

Returns the geometrically derived stability factor for the load

\textbf{Syntax}
\[ y = \texttt{mu}(S) \]

\textbf{Arguments}

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{S}</td>
<td>scattering matrix of a 2-port network</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

\textbf{Examples}
\[ x = \texttt{mu}(S) \]

\textbf{Defined in}
$\text{HPEESOF_DIR/expressions/ael/circuit_fun.ael}$

\textbf{See Also}
\texttt{mu_prime()}

\textbf{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:

\[ \text{mu} = \frac{1 - |S_{11}|^2}{|S_{22} - \text{conj}(S_{11})*\text{Delta}| + |S_{12}*S_{21}|} \]

where \( \text{Delta} \) is the determinant of the \( S \)-parameter matrix. Having \( \text{mu} > 1 \) is the single necessary and sufficient condition for unconditional stability of the 2-port network.

\textbf{Reference}
mu_prime()

Returns the geometrically derived stability factor for the source

Syntax
y = mu_prime(S)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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:

\[
u_{\text{mu_prime}} = \frac{1 - |S_{22}|^2}{|S_{11} - \text{conj}(S_{22})*\Delta| + |S_{21}*S_{12}|}
\]

where \( \Delta \) is the determinant of the S-parameter matrix. Having \( u_{\text{mu_prime}} > 1 \) is the single necessary and sufficient condition for unconditional stability of the 2-port network.

Reference

S-Parameter Analysis Functions

**ns_circle()**

Returns noise-figure circles

**Syntax**

\[ y = \text{ns\_circle}(nf2, NFmin, Sopt, rn, numOfPts, numCircles, NFStep) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>nf2</td>
<td>specified noise figure</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>†</td>
<td>no</td>
</tr>
<tr>
<td>NFmin</td>
<td>minimum noise figure</td>
<td>([0, \infty))</td>
<td>integer or real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Sopt</td>
<td>optimum mismatch</td>
<td>([0, \infty))</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>rn</td>
<td>equivalent normalized noise resistance of a 2-port network ‡</td>
<td>([0, \infty))</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>numOfPts</td>
<td>desired number of points per circle</td>
<td>([1, \infty))</td>
<td>integer</td>
<td>51</td>
<td>no</td>
</tr>
<tr>
<td>numCircles</td>
<td>number of desired circles. This is used if nf2 is not specified.</td>
<td>([0, \infty))</td>
<td>integer</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>NFStep</td>
<td>nf step size. This is used if nf2 is not specified.</td>
<td>([0, \infty))</td>
<td>integer or real</td>
<td>1.0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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**

```plaintext
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\}
```

9-34  ns_circle()
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.
S-Parameter Analysis Functions

**ns\_pwr\_int()**

Returns the integrated noise power

Syntax

```plaintext
y = ns_pwr_int(Sji, nf, resBW, stop)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sji</td>
<td>complex transmission coefficient</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>nf</td>
<td>noise figure at the output port (in dB)</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>resBW</td>
<td>user-defined resolution bandwidth</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>stop</td>
<td>stop value (works for nonuniform delta frequency)</td>
<td>[0, ∞)</td>
<td>real</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
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

\[ y = \text{ns_pwr_ref_bw}(S_{ji}, nf, \text{resBW}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sji</td>
<td>complex transmission coefficient</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>nf</td>
<td>noise figure at the output port (in dB)</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>resBW</td>
<td>user-defined resolution bandwidth</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ Y = \text{ns_pwr_ref_bw}(S21, nf2, 1\text{MHz}) \]
returns the noise power with respect to the reference bandwidth

Defined in

$\text{HPEESOF_DIR/expressions/ael/rf_system_fun.ael}$

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sji</td>
<td>power-dependent parameter obtained from large-signal S-parameters simulation</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
a = phase_comp(S21[:, 0])
```

**Defined in**

$\text{HP/ESOF_DIR/expressions/ael/rf_systems_fun.ael}$

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>2 X 2 scattering matrix</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>Zs</td>
<td>input impedance</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>Zl</td>
<td>Output impedance</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>Zref</td>
<td>reference impedance</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ pGain = pwr\_gain(S, 50, 75) \]
Zref defaults to 50 ohms

\[ pGain1 = pwr\_gain(S, 50, 75, 75) \]
Zref = 75 ohms

Defined in

$HPEESOF\_DIR/expressions/acl/RF\_system\_fun.acl$

See Also

stos(), volt_gain(), volt_gain_max()
ripple()

This function measures the deviation of x from the average of x

Syntax
\[ y = \text{ripple}(x, \text{fstart}, \text{fstop}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>can be a gain or group delay data over a given frequency range ([-\infty, \infty])</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>fstart</td>
<td>start frequency ((0, \infty))</td>
<td>real</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>fstop</td>
<td>stop frequency ((0, \infty))</td>
<td>real</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

Examples

\[ a = \text{ripple}(\text{pwr_gain}(S21)) \]
returns the ripple of the transducer power gain of S21

\[ a1 = \text{ripple}(\text{pwr_gain}(S21), 1\text{GHz}, 3\text{GHz}) \]
returns the ripple frequency between the start frequency (1GHz) and stop frequency (3GHz)

Defined in

$\text{HPEESOF_DIR/expressions/ael/elementary_fun.ael}$

See Also

\text{dev_lin_gain()}, \text{dev_lin_phase()}, \text{diff()}, \text{mean()}, \text{phasedeg()}, \text{phaserad()}, \text{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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network.</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>numOfPts</td>
<td>desired number of points per circle</td>
<td>[1, ∞)</td>
<td>integer</td>
<td>51</td>
<td>no</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>2-port S-Parameters</td>
<td>complex</td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Type of parameter to return</td>
<td>[&quot;both&quot;,&quot;center&quot;,&quot;radius&quot;]</td>
<td>string</td>
<td>&quot;both&quot;</td>
<td>no</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network.</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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
S-Parameter Analysis Functions

**sm_gamma1()**

Returns the simultaneous-match input-reflection coefficient

**Syntax**

```plaintext
y = sm_gamma1(S)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of 2-port network</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
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
\[ y = \text{sm}_\gamma 2(S) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of 2-port network</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples
\[ a = \text{sm}_\gamma 2(S) \]

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 \( \text{stab}_\text{fact}(S) \) is less than unity for the analyzed circuit, then \( \text{sm}_\gamma 2(S) \) returns 1+j*0.
S-Parameter Analysis Functions

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

\[ y = \text{sm}_y1(S, Z) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>Z</td>
<td>port impedance</td>
<td>(-∞, ∞)</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ a = \text{sm}_y1(S, 50) \]

Defined in

$\text{HPEESOF_DIR/expressions/ael/circuit_fun.ael}$

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

\[ y = \text{sm}_y2(S, Z) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>Z</td>
<td>port impedance</td>
<td>(-\infty, \infty)</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ a = \text{sm}_y2(S, 50) \]

Defined in

$\text{HPEESOF_DIR/expressions/ael/circuit_fun.ael}$

See Also

sm_y1()
S-Parameter Analysis Functions

**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**

\[ y = \text{sm}_z1(S, Z) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>Z</td>
<td>port impedance</td>
<td>((-\infty, \infty))</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{sm}_z1(S, 50) \]

**Defined in**

$\text{HPEESOF_DIR/expressions/ael/circuit_fun.ael}$

**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

\[ y = \text{sm}_z2(S, Z) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>Z</td>
<td>port impedance</td>
<td>(-\infty, \infty)</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ a = \text{sm}_z2(S, 50) \]

Defined in

$\text{HPEESOF_DIR}/expressions/ael/circuit_fun.ael$

See Also

sm_z1()
S-Parameter Analysis Functions

**stab_fact()**

Returns the Rollett stability factor

**Syntax**

```matlab
k = stab_fact(S)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of 2-port network</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```matlab
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:

\[
k = \frac{1 - |S_{11}|^2 - |S_{22}|^2 + |S_{11}S_{22} - S_{12}S_{21}|^2}{2|S_{12}S_{21}|}
\]

The necessary and sufficient conditions for unconditional stability are that the stability factor is greater than unity and the stability measure is positive.

**Reference**

**stab_meas()**

Returns the stability measure

**Syntax**

\[ k = \text{stab_meas}(S) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of 2-port network</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ b = \text{stab_meas}(S) \]

**Defined in**

$\text{HPEESOF_DIR/expressions/ael/rf_system_fun.ael}$

**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**

S-Parameter Analysis Functions

stoabcd()

This measurement transforms the scattering matrix of a 2-port network to a chain (ABCD) matrix

Syntax

\[
y = \text{stoabcd}(S, \text{zRef})
\]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>(-∞, ∞)</td>
<td>integer, real or complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

Examples

\[
a = \text{stoabcd}(S, 50)
\]

Defined in

$\text{HPEESOF_DIR/expressions/ael/network_fun.ael}$

See Also

abcdtoh(), stoh(), stoy()
stoh()

This measurement transforms the scattering matrix of a 2-port network to a hybrid matrix.

**Syntax**

\[ y = \text{stoh}(S, z\text{Ref}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>((-\infty, \infty))</td>
<td>integer, real or complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

\[ h = \text{stoh}(S, 50) \]

**Defined in**

$\text{HPEESOF_DIR/expressions/ael/network_fun.ael}$

**See Also**

htos(), stoabcd(), stoy()
S-Parameter Analysis Functions

**stos()**

Changes the normalizing impedance of a scattering matrix

**Syntax**

\[ y = \text{stos}(S, z\text{Ref}, z\text{New}, zy) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix</td>
<td>(-\infty, \infty)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>normalizing impedance</td>
<td>(-\infty, \infty)</td>
<td>integer, real or complex</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>zNew</td>
<td>new normalizing impedance</td>
<td>(-\infty, \infty)</td>
<td>integer, real or complex</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>zy</td>
<td>directs the conversion</td>
<td>[0, 1]</td>
<td>integer</td>
<td>1 (Y-matrix)</td>
<td>no</td>
</tr>
</tbody>
</table>

† If \( zy=0 \), 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**

\[ a = \text{stos}(S, 50, 75, 1) \]

converts the 50 ohm terminated S-parameters to 75 ohm terminated S-parameters through the Y-matrix

\[ a = \text{stos}(S, 75) \]

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 1 terminated in 50 ohms, and port 2 in 75 ohms. The expression below converts the S-parameters at a 50 ohm impedance termination at both ports:

\[ S_{50} = \text{stos}(S, \text{PortZ}, 50, 1) \]

The converted S-parameters can then be written to a S2P file using the function `write_snp()`

**Defined in**

$HPEESOF_DIR/expressions/ael/network_fun.ael$

9-54  stos()
See Also

stoy(), stoz()
S-Parameter Analysis Functions

**stot()**

This function transforms the scattering matrix of a 2-port network to a chain scattering matrix

**Syntax**

```
t = stot(S)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network</td>
<td>$(-\infty, \infty)$</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>(-∞, ∞)</td>
<td>integer, real or complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**
y = stoy(S, 50.0)

**Defined in**
$HPEESOF_DIR/expressions/acl/network_fun.acl$

**See Also**
stoh(), stoz(), ytos()
S-Parameter Analysis Functions

stoz()
This measurement transforms a scattering matrix to an impedance matrix

Syntax
z = stoz(S, zRef)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>scattering matrix of a 2-port network</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>(-∞, ∞)</td>
<td>integer, real or complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

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

\[ y = \text{tdr}_\text{sp}_\text{gamma}(Sii, \text{delay}, T\text{start}, T\text{stop}, \text{NumPts}, \text{window}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sii</td>
<td>Complex reflection</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>delay</td>
<td>delay value</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Tstart</td>
<td>Start Time</td>
<td>[0, ∞)</td>
<td>real</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>Tstop</td>
<td>Stop Time</td>
<td>[0, ∞)</td>
<td>real</td>
<td>2 cycles</td>
<td>no</td>
</tr>
<tr>
<td>NumPts</td>
<td>Number of points</td>
<td>[0, ∞)</td>
<td>real or string</td>
<td>101</td>
<td>no</td>
</tr>
<tr>
<td>Window</td>
<td>Windowing to be applied</td>
<td>[0, 9]†</td>
<td>real</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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 = \text{tdr}_\text{sp}_\text{gamma}(S(1,1), 0.05\text{ns}, -0.2 \text{ ns}, 3.8 \text{ ns}, 401, "\text{Hamming}"\) \]

Defined in

$\text{HPEESOF DIR/expressions/ael/DesignGuide_fun.ael}$
S-Parameter Analysis Functions

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sii</td>
<td>Complex reflection</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>delay</td>
<td>delay value</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>zRef</td>
<td>Reference impedance</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Tstart</td>
<td>Start Time</td>
<td>[0, ∞)</td>
<td>real</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>Tstop</td>
<td>Stop Time</td>
<td>[0, ∞)</td>
<td>real</td>
<td>2 cycles</td>
<td>no</td>
</tr>
<tr>
<td>NumPts</td>
<td>Number of points</td>
<td>[0, ∞)</td>
<td>real</td>
<td>101</td>
<td>no</td>
</tr>
<tr>
<td>Window</td>
<td>Windowing to be applied</td>
<td>[0, 7] †</td>
<td>real</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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
S-Parameter Analysis Functions

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

\[ y = \text{tdr\_step\_imped}(\text{time\_waveform}, \, z\text{Ref}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_waveform</td>
<td>time domain pulse TDR waveform</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>Reference impedance</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ x = \text{tdr\_step\_imped}(\text{vout}, \, 50) \]

Defined in

$\text{HPEESOF\_DIR/expressions/ael/DesignGuide\_fun.ael}$

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.
S-Parameter Analysis Functions

**ttos()**

This function transforms the chain scattering matrix of a 2-port network to a scattering matrix

**Syntax**

```plaintext
sp = ttos(T)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>scattering matrix of a 2-port network</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
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 = unilateral_figure(SParam)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SParam</td>
<td>2-Port S-Parameters</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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 = unilateral_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 S12 (unilateral behavior of device). It is calculated as:

\[
U = \frac{|S_{11}| |S_{12}| |S_{21}| |S_{22}|}{(1 - |S_{11}|^2)(1 - |S_{22}|^2)}
\]
S-Parameter Analysis Functions

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
\[
y = \text{unwrap}(\text{phase}, \text{jump})
\]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>phase</td>
<td>swept real variable</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>jump</td>
<td>absolute jump</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>180.0</td>
<td>no</td>
</tr>
</tbody>
</table>

Examples

\[
a = \text{unwrap} \left( \text{phase}(S21) \right)
a = \text{unwrap} \left( \text{phaserad}(S21), \pi \right)
\]

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 successive 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlus</td>
<td>voltage at the positive output terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vMinus</td>
<td>voltage at the negative output terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

y = v_dc(vp, vm)

Defined In

$HPEESOF_DIR/expressions/acl/circuit_fun.acl
volt_gain()

Returns the voltage gain

Syntax

\[ y = \text{volt\_gain}(S, Z_s, Z_l, Z_{\text{ref}}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>2 2 scattering matrix measured with equal terminations of Z_{\text{ref}}</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Z_s</td>
<td>input impedance</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>Z_l</td>
<td>Output impedance</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>Z_{\text{ref}}</td>
<td>reference impedance</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ a = \text{volt\_gain}(S, 50, 75) \]

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.

\[
\text{volt\_gain} = \frac{S_{21}}{2} \cdot \frac{(1 - \Gamma_S) \cdot (1 + \Gamma_L)}{(S_{11} \cdot \Gamma_S - 1) \cdot (S_{22} \cdot \Gamma_L - 1) - S_{12} \cdot S_{21} \cdot \Gamma_S \cdot \Gamma_L}
\]

(9-1)

Figure 9-1 illustrates the volt_gain measurement.
S-Parameter Analysis Functions

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:

   \[
   \text{volt\_gain}(S, 100, 50, 50)
   \]

   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**

\[ y = \text{volt_gain_max}(S, Zs, Zl, Zref) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>2 X 2 scattering matrix</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>Zs</td>
<td>input impedance</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>Zl</td>
<td>Output impedance</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>Zref</td>
<td>reference impedance</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ vGain = \text{volt_gain_max}(S, 50, 75) \]

\[ Zref \text{ defaults to 50 ohms} \]

\[ vGain1 = \text{volt_gain}(S, 50, 75, 75) \]

\[ Zref = 75 \text{ ohms} \]

**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.
S-Parameter Analysis Functions

vswr()

Given a complex reflection coefficient, this measurement returns the voltage standing wave ratio

Syntax
y = stab_meas(Sii)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sii</td>
<td>complex reflection coefficient</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples
a = vswr(Sii)

Defined in
$HPEE$OF_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

\[ y = \text{write}_\text{snp}(\text{FileName}, \text{S}, \text{Comment}, \text{FreqUnit}, \text{DataFormat}, \text{Zref}, \text{Znorm}, \text{ZorY}, \text{Precision}, \text{Delimiter}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>Name or full-path of the S-Parameter file.</td>
<td></td>
<td>string</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>S-parameter matrix variable</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td>Text that is to be written at the top of file.</td>
<td>string</td>
<td>**</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>FreqUnit</td>
<td>Frequency Unit</td>
<td>&quot;Hz&quot;, &quot;KHz&quot;, &quot;MHz&quot;, &quot;GHz&quot;, &quot;THz&quot;</td>
<td>string</td>
<td>GHz</td>
<td>no</td>
</tr>
<tr>
<td>DataFormat</td>
<td>Format of S-Parameter that is to be output</td>
<td>&quot;MA&quot;, &quot;DB&quot;, &quot;RI&quot;†</td>
<td>string</td>
<td>&quot;MA&quot;</td>
<td>no</td>
</tr>
<tr>
<td>Zref</td>
<td>Reference impedance (scalar or vector)</td>
<td>(0, \infty)</td>
<td>real</td>
<td>50</td>
<td>no</td>
</tr>
<tr>
<td>Znorm</td>
<td>Normalizing impedance (a scalar value)</td>
<td>(0, \infty)</td>
<td>real</td>
<td>50</td>
<td>no</td>
</tr>
<tr>
<td>ZorY</td>
<td>Directs the conversion through Z or Y transform ‡</td>
<td>[0, 1]</td>
<td>integer</td>
<td>1 (Y-matrix)</td>
<td>no</td>
</tr>
<tr>
<td>Precision</td>
<td>precision of the data</td>
<td>[1, 64]</td>
<td>integer</td>
<td>6</td>
<td>no</td>
</tr>
<tr>
<td>Delimiter</td>
<td>Delimiter that separates the data</td>
<td>string</td>
<td>**</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

† "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.
writes the S-parameters to the file spar_ts.s2p in mag-phase format

write_sn(np("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_sn(np("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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sii</td>
<td>complex reflection coefficient</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>(-∞, ∞)</td>
<td>integer, real or complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

Examples
yIN = yin(S11, 50)

Defined in
$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also
vswr(), zin()
S-Parameter Analysis Functions

**yopt()**

Returns optimum admittance for noise match

**Syntax**

\[ y = \text{yopt}(\gammaOpt, \text{zRef}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>gammaOpt</td>
<td>optimum reflection coefficient</td>
<td>((\infty, \infty))</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>((\infty, \infty))</td>
<td>real, complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{yopt}(\text{Sopt}, 50) \]

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/circuit\_fun.ael}$

**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 (\(\text{NF}_{\text{min}}\)).
ytoabcd()

This measurement transforms an admittance matrix of a 2-port network into a hybrid matrix.

Syntax

a = ytoabcd(Y)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>2-port admittance matrix</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

a = ytoabcd(Y)

Defined in

$HPEE$OF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh(), htoabcd()
S-Parameter Analysis Functions

ytoh()

This measurement transforms an admittance matrix of a 2-port network into a hybrid matrix

**Syntax**

\( h = ytoh(Y) \)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>2-port admittance matrix</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\( h = ytoh(Y) \)

**Defined in**

$HPEE\_SOF\_DIR/expressions/ael/network_fun.ael$

**See Also**

htoy(), ytoabcd()
**ytos()**

This measurement transforms an admittance matrix into a scattering matrix.

**Syntax**

\[ z = ytos(Y, z\text{Ref}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>admittance matrix</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>((-\infty, \infty))</td>
<td>integer, real or complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

\[ s = ytos(Y, 50.0) \]

**Defined in**

$\text{HPEESOF_DIR/expressions/ael/network\_fun.ael}$

**See Also**

`stoy()`, `ytoz()`
S-Parameter Analysis Functions

ytoz()
This measurement transforms an admittance matrix to an impedance matrix

Syntax
Z = ytoz(Y)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>admittance matrix</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

a = ytoz(Y)

Defined in
$HPEE$OF_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

\( z = \text{zin}(S_{ii}, Z) \)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sii</td>
<td>complex reflection coefficient.</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>((-\infty, \infty))</td>
<td>integer, real or complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

Examples

\( zIN = \text{zin}(S_{11}, 50.0) \)

Defined in

$\text{HPEESOF_DIR/expressions/ael/network_fun.ael}$

See Also

\( \text{vswr()}, \text{yin()} \)
zopt()  
Returns optimum impedance for noise match

Syntax
\[ y = \text{zopt}(\gamma_{\text{Opt}}, z_{\text{Ref}}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>gammaOpt</td>
<td>optimum reflection coefficient</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>50.0</td>
<td>no</td>
</tr>
</tbody>
</table>

Examples
\[ a = \text{zopt}(\text{Sopt}, 50) \]

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. \(\gamma_{\text{Opt}}\) is the optimum reflection coefficient that must be presented at the input of the network to realize the minimum noise figure \((\text{NF}_{\text{min}})\).
ztoabcd()  
This measurement transforms an impedance matrix of a 2-port network into a chain (ABCD) matrix

Syntax
a = ztoabcd(Z)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>2-port impedance matrix</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples
a = ztoabcd(Z)

Defined in
$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also
abcdtoz(), ytoabcd(), ztoh()
S-Parameter Analysis Functions

ztoh()
This measurement transforms an impedance matrix of a 2-port network into a hybrid matrix

Syntax
h = ztoh(Z)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>2-port impedance</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples
h = ztoh(Z)

Defined in
$HPEE Sof_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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>impedance matrix</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>zRef</td>
<td>reference impedance</td>
<td>(-∞, ∞)</td>
<td>integer, real or</td>
<td>50.0</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>complex</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples
s = ztos(Z, 50.0)

Defined in
$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also
stoz(), ytoz(), ztoy()
S-Parameter Analysis Functions

ztoy()
This measurement transforms an impedance matrix to an admittance matrix

**Syntax**
y = ztoy(Z)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>impedance matrix</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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.

“cdf()” on page 10-2 “moving_average()” on page 10-18
“cross_corr()” on page 10-3 “norm_dist_inv1D()” on page 10-19
“fun_2d_outer()” on page 10-4 “norm_dist1D()” on page 10-20
“histogram()” on page 10-6 “norms_dist_inv1D()” on page 10-22
“histogram_multiDim()” on page 10-8 “norms_dist1D()” on page 10-23
“histogram_sens()” on page 10-9 “pdf()” on page 10-24
“histogram_stat()” on page 10-11 “stddev()” on page 10-25
“lognorm_dist_inv1D()” on page 10-13 “stddev_outer()” on page 10-26
“lognorm_dist1D()” on page 10-14 “uniform_dist_inv1D()” on page 10-27
“mean()” on page 10-15 “uniform_dist1D()” on page 10-28
“mean_outer()” on page 10-16 “yield_sens()” on page 10-29
“median()” on page 10-17
cdf()
Returns the cumulative distribution function (CDF)

Syntax
\[ y = \text{cdf}(\text{data}, \text{numBins}, \text{minBin}, \text{maxBin}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>the signal</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>numBins</td>
<td>number of subintervals or bins used to measure CDF</td>
<td>[1, \infty)</td>
<td>real</td>
<td>\log(\text{numOf Pts})/\log(2.0)</td>
<td>no</td>
</tr>
<tr>
<td>minBin</td>
<td>beginning of the evaluation of the CDF</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>minimum value of the data</td>
<td>no</td>
</tr>
<tr>
<td>maxBin</td>
<td>end of the evaluation of the CDF</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>maximum value of the data</td>
<td>no</td>
</tr>
</tbody>
</table>

Examples
\[ y = \text{cdf}(\text{data}) \]
\[ y = \text{cdf}(\text{data}, 20) \]

Defined in
$\text{HPE ESOF } \text{DIR/expressions/ael/statistical_fun.ael}$

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>one-dimensional data</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>v2</td>
<td>one-dimensional data</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

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
Statistical Analysis Functions

**fun_2d_outer()**
Applies a function to the outer dimension of two-dimensional data

**Syntax**
y = fun_2d_outer(data, fun)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>two-dimensional data</td>
<td>(-∞, ∞) integer, real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>fun</td>
<td>name of function (usually mean, max, or min) that will be applied to the outer dimension of the data</td>
<td>string</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>

**Examples**
y = fun_2d_outer(data, min)

**Defined in**
$HPEESOF_DIR/expressions/ael/statistical_fun.ael$

**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

---

10-4 fun_2d_outer()
fun_2d_outer allows the mean to be computed over each element in the outer dimension.
Statistical Analysis Functions

**histogram()**

Generates a histogram representation. This function creates a histogram that represents data.

**Syntax**

\[
y = \text{histogram}(\text{data}, \text{numBins}, \text{minBin}, \text{maxBin})
\]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>signal (must be one-dimensional)</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>numBins</td>
<td>number of subintervals or bins used to measure the histogram.</td>
<td>([1, \infty))</td>
<td>integer</td>
<td>(\log(\text{numOf Pts})/\log(2.0))</td>
<td>no</td>
</tr>
<tr>
<td>minBin</td>
<td>beginning of the evaluation of the histogram</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>minimum value of the data</td>
<td>no</td>
</tr>
<tr>
<td>maxBin</td>
<td>end of the evaluation of the histogram</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>maximum value of the data</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

\[
y = \text{histogram}(\text{data})
\]

\[
y = \text{histogram}(\text{data}, 20)
\]

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:

\[
y = \text{histogram}(\text{dB}(S_{21}[::,99]))
\]

**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.
**Statistical Analysis Functions**

**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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>the signal</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>normalized</td>
<td>sets normalization of data †</td>
<td>&quot;no&quot;, &quot;yes&quot;</td>
<td>string</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>numBins</td>
<td>number of subintervals or bins used to measure CDF</td>
<td>[1, ∞)</td>
<td>real</td>
<td>log(numOfPts)/log(2.0)</td>
<td>no</td>
</tr>
<tr>
<td>minBin</td>
<td>beginning of the evaluation of the CDF</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>minimum value of the data</td>
<td>no</td>
</tr>
<tr>
<td>maxBin</td>
<td>end of the evaluation of the CDF</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>maximum value of the data</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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 S12. 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**

\[
y = \text{histogram}\_\text{sens}(\text{data, sensitivityVar, goalMin, goalMax, innermostIndepLow, innermostIndepHigh, numBins})
\]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>statistical response</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>sensitivityVar</td>
<td>selected statistical variable</td>
<td></td>
<td>string</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>goalMin</td>
<td>specifies the performance range†</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>goalMax</td>
<td>specifies the performance range†</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>innermostIndepLow</td>
<td>specifies the low value of the subrange of data with the inner most sweep variable</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>innermostIndepHigh</td>
<td>specifies the high value of the subrange of data with the inner most sweep variable</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>numBins</td>
<td>number of sub-intervals or bins used to measure the histogram.</td>
<td>(1, \infty)</td>
<td>real</td>
<td>(\log(\text{numOf Pts})/\log(2.0))</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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.
Statistical Analysis Functions

The design requires the maximum of \( \text{db}(S11) \), -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:

\[
\text{Histogram\_sens\_S11} = \text{histogram\_sens}(\text{db}(S11), C1v, , -18.0, 200\text{MHz}, 400\text{MHz})
\]

Defined in

\$HPEE\_SOF\_DIR/expressions/ael/statistical\_fun.ael

See Also

\text{histogram()}, \text{histogram\_MultiDim()}, \text{histogram\_stat()}, \text{build\_subrange()}, \text{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 multi-dimension data, then calls histogram_multiDim() to produce the histogram.

**Syntax**

```
y = histogram_stat(data, normalized, innermostIndepLow, innermostIndepHigh, numBins, minBin, maxBin)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>statistical data to be analyzed</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>normalized</td>
<td>sets normalization of data †</td>
<td>&quot;no&quot;, &quot;yes&quot;</td>
<td>string</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>innermostIndepLow</td>
<td>specifies the low value of the subrange of data with the inner most sweep variable</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>innermostIndepHigh</td>
<td>specifies the high value of the subrange of data with the inner most sweep variable</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>numBins</td>
<td>number of subintervals or bins used to measure histogram</td>
<td>[1, ∞)</td>
<td>real</td>
<td>log(numOf Pts)/log(2.0)</td>
<td>no</td>
</tr>
<tr>
<td>minBin</td>
<td>beginning of the evaluation of the histogram</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>minimum value of the data</td>
<td>no</td>
</tr>
<tr>
<td>maxBin</td>
<td>end of the evaluation of the histogram</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>maximum value of the data</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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.
Statistical Analysis Functions

```
Histogram_stat_S12 = histogram_stat(db(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

\[ y = \text{lognorm_dist_inv1D}(\text{data}, m, s, \text{absS}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>real number representing the cumulative probability</td>
<td>[0, 1]</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>m</td>
<td>mean for the lognormal distribution</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>s</td>
<td>standard deviation for the lognormal distribution</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>absS</td>
<td>specifies absolute or relative standard deviation</td>
<td>[0, \infty) †</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
</tbody>
</table>

† absS is not equal to “0”, s is the absolute standard deviation; otherwise, s is the relative standard deviation

Examples

\[ X_{\text{cpf}} = 0.5 \]
\[ X = \text{lognorm_dist_inv1D}(X_{\text{cpf}}, 2.0, 0.2, 1) \]
\[ XX_{\text{cpf}} = [0.0::0.01::1.0] \]
\[ XX = \text{lognorm_dist_inv1D}(XX_{\text{cpf}}, 2.0, 0.2, 1) \]

Defined in

$\text{HPEESOF_DIR/expressions/ael/statistical_fun.ael}$
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>number or a one-dimensional data</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>m</td>
<td>mean for the lognormal distribution</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>s</td>
<td>standard deviation for the lognormal distribution</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>absS</td>
<td>specifies absolute or relative standard deviation</td>
<td>[0, ∞) †</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>is_cpf</td>
<td>specifies cdf or pdf of the lognormal distribution</td>
<td>[0, ∞) † †</td>
<td>integer</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† When absS is not equal to "0", s is the absolute standard deviation; otherwise, s is the relative standard deviation
† † When is_cpf 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data to find mean</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

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  

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>2-dimensional data to find mean</td>
<td>(-∞, ∞)</td>
<td>integer, real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples  
a = mean_outer(data)

Defined in  
$HPEESOF_DIR/expressions/ael/statistical_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**

\[ y = \text{median}(x) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>data to find the median</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[
a = \text{median}([1, 2, 3, 4])
\]

returns 2.5

**Defined in**

$\text{HPEESOF _DIR/expressions/ael/statistical\_fun.ael}$

**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**

```plaintext
y = moving_average(Data, NumPoints)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>one or multi-dimensional sequence of numbers</td>
<td>(-∞, ∞)</td>
<td>integer, real or complex</td>
<td>yes</td>
</tr>
<tr>
<td>NumPoints</td>
<td>Number of points to be averaged together</td>
<td>[1, ∞) †</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

† 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**

```plaintext
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

```plaintext
y = norm_dist_inv1D(data, m, s, absS)
```

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>represents the cumulative probability</td>
<td>[0, 1]</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>m</td>
<td>mean for the normal distribution</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>s</td>
<td>standard deviation for the normal distribution</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>absS</td>
<td>specifies absolute or relative standard deviation</td>
<td>[0, ∞) †</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
</tbody>
</table>

† When absS is not equal to "0", s is the absolute standard deviation; otherwise, s is the relative standard deviation.

Examples

```plaintext
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
\[ y = \text{norm\_dist1D}(\text{data}, \text{m}, \text{s}, \text{absS}, \text{is\_cdf}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>number or a one-dimensional data</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>mean for the normal distribution</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>standard deviation for the normal distribution</td>
<td>(-\infty, \infty)</td>
<td>integer, real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>absS</td>
<td>specifies absolute or relative standard deviation</td>
<td>[0, \infty) †</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>is_cdf</td>
<td>specifies cdf or pdf of normal distribution</td>
<td>[0, \infty) † †</td>
<td>integer</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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

\[ y = \text{norms\_dist\_inv1D}(\text{data}) \]

Arguments

- **data**: number represents the cumulative probability
  - Range: \([0, 1]\)
  - Type: integer, real
  - Required: yes

Examples

- \( X_{\text{cpf}} = 0.5 \)
- \( X = \text{norms\_dist\_inv1D}(X_{\text{cpf}}) \)
  - will be equal to 0.0
- \( XX_{\text{cpf}} = [0.0::0.01::1.0] \)
- \( XX = \text{norms\_dist\_inv1D}(XX_{\text{cpf}}) \)

Defined in

$\text{HPEESOF}\_\text{DIR}/\text{expressions/ael/statistical\_fun.ael}$

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

\[ y = \text{norms}\_\text{dist}1D(\text{data}, \text{is}\_\text{cdf}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>number or a one-dimensional data</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>is_cdf</td>
<td>specifies cdf or pdf of standard normal distribution</td>
<td>[0, \infty] †</td>
<td>integer</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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

\[ X = 0.5 \]
\[ X\_\text{pdf} = \text{norms}\_\text{dist}1D(X, 0) \]
\[ X\_\text{cdf} = \text{norms}\_\text{dist}1D(X, 1) \]
\[ XX = [-3.9::0.1::3.9] \]
\[ XX\_\text{pdf} = \text{norms}\_\text{dist}1D(XX, 0) \]
\[ XX\_\text{cdf} = \text{norms}\_\text{dist}1D(XX, 1) \]

Defined in

$\text{HPEESOF}\_\text{DIR/expressions/ael/statistical\_fun.ael}$

See Also

\text{norm}\_\text{dist}1D(), \text{norm}\_\text{dist}\_\text{inv}1D(), \text{norms}\_\text{dist}\_\text{inv}1D(), \text{lognorm}\_\text{dist}\_\text{inv}1D(), \text{lognorm}\_\text{dist}1D(), \text{uniform}\_\text{dist}\_\text{inv}1D(), \text{uniform}\_\text{dist}1D()
 Statistical Analysis Functions

**pdf()**

Returns a probability density function (PDF)

**Syntax**
y = pdf(data, numBins, minBin, maxBin)

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>the signal</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>numBins</td>
<td>number of subintervals or bins used to</td>
<td>[1, ∞)</td>
<td>real</td>
<td>log(numOf Pts)/log(2.0)</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>measure PDF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>minBin</td>
<td>beginning of the evaluation of the PDF</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>minimum value of the data</td>
<td>no</td>
</tr>
<tr>
<td>maxBin</td>
<td>end of the evaluation of the PDF</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>maximum value of the data</td>
<td>no</td>
</tr>
</tbody>
</table>

**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**

\[ y = \text{stddev}(\text{data}, \text{flag}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>data to find the stddev</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>flag</td>
<td>indicates how stddev normalizes</td>
<td>[0, 1] †</td>
<td>integer</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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**

$\text{HPEESOF_DIR/expressions/ael/statistical_fun.ael}$

**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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>data to find the stddev</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>flag</td>
<td>indicates how stddev normalizes</td>
<td>[0, 1] †</td>
<td>integer</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>represents the cumulative probability</td>
<td>[0, 1]</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>A</td>
<td>uniform distributed range</td>
<td>[0, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
<tr>
<td>B</td>
<td>uniform distributed range</td>
<td>[0, ∞)</td>
<td>integer, real</td>
<td>yes</td>
</tr>
</tbody>
</table>

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()
Statistical Analysis Functions

**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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>number or a one-dimensional data</td>
<td>(-∞, ∞)</td>
<td>integer, real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>uniform distributed range</td>
<td>[0, ∞)</td>
<td>integer, real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>uniform distributed range</td>
<td>[0, ∞)</td>
<td>integer, real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>is_cdf</td>
<td>specifies cdf or pdf of standard uniform distribution</td>
<td>[0, ∞)†</td>
<td>integer</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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**

\[ y = \text{yield}_sens(pf\_data, \text{numBins}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>pf_data</td>
<td>binary-valued scalar data set indicating the pass/fail status of each value of a companion independent variable</td>
<td>[0-1]</td>
<td>integer</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>numBins</td>
<td>number of subintervals or bins used to measure yield_sens</td>
<td>[1, ∞)</td>
<td>real</td>
<td>( \log(\text{numOf Pts})/\log(2.0) )</td>
<td>no</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{yield}_sens(pf\_data) \]

\[ a = \text{yield}_sens(pf\_data, 20) \]

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/statistical\_fun.ael}$

**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.
10-30 yield_sens()
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

“constellation()” on page 11-2          “pfc_tran()” on page 11-12
“cross()” on page 11-4                 “pt tran()” on page 11-13
“fs()” on page 4-37                   “pspec tran()” on page 11-14
“fspot()” on page 11-5                 “vfc tran()” on page 11-16
“ifc_tran()” on page 11-9              “vspec tran()” on page 11-17
“ispec_tran()” on page 11-10           “vt tran()” on page 11-19

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 “[ ]”.
Transient Analysis Functions

**constellation()**
Generates the constellation diagram from Circuit Envelope, Transient, or Ptolemy simulation I and Q data

**Syntax**
```matlab
Const = constellation(i_data, q_data, symbol_rate, delay)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_data</td>
<td>in-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †</td>
<td>(-∞, ∞)</td>
<td>complex</td>
<td>yes</td>
</tr>
<tr>
<td>q_data</td>
<td>quadrature-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>symbol_rate</td>
<td>symbol rate of the modulation signal</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>delay</td>
<td>delay value † †</td>
<td>[0, ∞)</td>
<td>real</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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.
Transient Analysis Functions

**cross()**

Computes the zero crossings of a signal, interval between successive zero crossings or slope at the crossing

**Syntax**

\[ y_{\text{Cross}} = \text{cross}(\text{signal}, \text{direction}, \text{slope}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>signal</td>
<td>the signal for which the zero crossing is to be found</td>
<td>((\infty, \infty))</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>direction</td>
<td>type of zero crossing †</td>
<td>([-1, 1])</td>
<td>integer</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>slope</td>
<td>specifies if slope is to be calculated, rather than interval between zero crossing</td>
<td>([0, 1])</td>
<td>integer</td>
<td>0 (no slope)</td>
<td>no</td>
</tr>
</tbody>
</table>

† If direction = +1, compute positive going zero crossings. If direction = -1, compute negative going zero crossings. If direction = 0, compute all zero crossings

**Examples**

\[ \text{period} = \text{cross}(\text{vosc}-2.0, 1) \]

this computes the period of each cycle of the vosc signal. The period is measured from each positive-going transition through 2.0V

\[ \text{period} = \text{cross}(\text{vosc}-2.0, 1, 1) \]

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()
Perform a single-frequency time-to-frequency transform

Syntax
\[
y = \text{fspot}(x, \text{fund}, \text{harm}, \text{windowType}, \text{windowConst}, \text{interpOrder}, \text{tstart})
\]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>time domain signal</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>fund</td>
<td>period (1/\text{fund}) for the Fourier transform</td>
<td>([1, \infty))</td>
<td>real</td>
<td>period that matches the length of the independent axis of (x)</td>
<td>no</td>
</tr>
<tr>
<td>harm</td>
<td>harmonic number (†)</td>
<td>([1, \infty))</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>windowType</td>
<td>type of window to apply to the data</td>
<td>([0, 9]‡)</td>
<td>integer, string</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>windowConst</td>
<td>window constant (‡ †)</td>
<td>([0, \infty))</td>
<td>real</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>interpOrder</td>
<td>interpolation scheme</td>
<td>([1, 3]‡ ‡)</td>
<td>integer</td>
<td></td>
<td>no</td>
</tr>
</tbody>
</table>
Transient Analysis Functions

### Transient Analysis Functions

#### 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

---

#### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>tstart</td>
<td>start time</td>
<td>([0, \infty))</td>
<td>real</td>
<td>1</td>
<td>no</td>
</tr>
</tbody>
</table>

- † 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

- † † 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
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.
Transient Analysis Functions

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOut</td>
<td>current through a branch</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>fundFreq</td>
<td>fundamental frequency</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>harmNum</td>
<td>harmonic number of the fundamental</td>
<td>[0, ∞)</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

**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

\[ y = \text{ispec_tran}(\text{iOut}, \text{fundFreq}, \text{numHarm}, \text{windowType}, \text{windowConst}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOut</td>
<td>current through a branch</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>fundFreq</td>
<td>fundamental frequency</td>
<td>[0, \infty)</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>numHarm</td>
<td>number of harmonics of fundamental</td>
<td>[0, \infty)</td>
<td>integer</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>windowType</td>
<td>type of window to be applied to the data</td>
<td>[0, 9]†</td>
<td>integer, string</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>windowConst</td>
<td>window constant ‡ †</td>
<td>[0, \infty)</td>
<td>integer, real</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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

\[ y = \text{ispec_tran}(\text{I_Probe1.i, 1GHz, 8}) \]

Defined in

$\text{HPEESOF_DIR/expressions/ael/circuit_fun.ael}$

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.
Transient Analysis Functions

\textbf{pfc\_tran()}

Returns frequency-selective power

\textbf{Syntax}

\[
y = \text{pfc\_tran}(v\text{Plus}, v\text{Minus}, i\text{Out}, \text{fundFreq}, \text{harmNum})
\]

\textbf{Arguments}

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>v\text{Plus}</td>
<td>voltage at the positive terminal</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>v\text{Minus}</td>
<td>voltage at the negative terminal</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>i\text{Out}</td>
<td>current through a branch measured for</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>power calculation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fundFreq</td>
<td>fundamental frequency</td>
<td>[0, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>harmNum</td>
<td>harmonic number of the fundamental frequency</td>
<td>[0, \infty)</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

\textbf{Examples}

\[
a = \text{pfc\_tran}(v1, v2, I\_Probel\_i, 1GHz, 1)
\]

\textbf{Defined in}

\$\text{HPEESOF\_DIR/expressions/ael/circuit\_fun.ael}$

\textbf{See Also}

\textit{ifc\_tran()}, \textit{vfc\_tran()}

\textbf{Notes/Equations}

This measurement gives RMS power, delivered to any part of the circuit at a particular frequency of interest. \text{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. \text{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.

pt_tran()

11-13


Transient Analysis Functions

pspec_tran()

Returns transient power spectrum

Syntax

\( y = \text{pspec\_tran}(v\text{Plus}, v\text{Minus}, i\text{Out}, \text{fundFreq}, \text{numHarm}, \text{windowType}, \text{windowConst}) \)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlus</td>
<td>voltage at the positive terminal</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>vMinus</td>
<td>voltage at the negative terminal</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>iOut</td>
<td>current through a branch measured for power calculation</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>fundFreq</td>
<td>fundamental frequency</td>
<td>[0, \infty)</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>numHarm</td>
<td>number of harmonics of fundamental frequency</td>
<td>[0, \infty)</td>
<td>integer</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>windowType</td>
<td>type of window to be applied to the data</td>
<td>[0, 9] †</td>
<td>integer, string</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>windowConst</td>
<td>window constant ‡</td>
<td>[0, \infty)</td>
<td>integer, real</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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

\( a = \text{pspec\_tran}(v1, v2, I\_Probel.i, 1\text{GHz}, 8) \)

11-14  pspec_tran()
Defined in
$HPEE SO_F_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.
Transient Analysis Functions

**vfc_tran()**

Returns the transient frequency-selective voltage

**Syntax**

\[ y = \text{vfc}_\text{tran}(vPlus, vMinus, \text{fundFreq}, \text{harmNum}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlus</td>
<td>voltage at the positive terminal</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vMinus</td>
<td>voltage at the negative terminal</td>
<td>(-\infty, \infty)</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>fundFreq</td>
<td>fundamental frequency</td>
<td>[0, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>harmNum</td>
<td>harmonic number of the fundamental frequency</td>
<td>[0, \infty)</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ a = \text{vfc}_\text{tran}(vOut, 0, 1\text{GHz}, 1) \]

**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).
vspecl tran()

Returns the transient voltage spectrum

Syntax
y = vspec tran(vPlus, vMinus, fundFreq, numHarm, windowType, windowConst)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlus</td>
<td>voltage at the positive terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>vMinus</td>
<td>voltage at the negative terminal</td>
<td>(-∞, ∞)</td>
<td>real, complex</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>fundFreq</td>
<td>fundamental frequency</td>
<td>[0, ∞)</td>
<td>real</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>numHarm</td>
<td>number of harmonics of fundamental frequency</td>
<td>[0, ∞)</td>
<td>integer</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>windowType</td>
<td>type of window to be applied to the data</td>
<td>[0, 9] †</td>
<td>integer, string</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>windowConst</td>
<td>window constant</td>
<td>[0, ∞)</td>
<td>integer, real</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

† 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
Transient Analysis Functions

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
\[ y = \text{vt}_\text{tran}(\text{vPlus}, \text{vMinus}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPlus</td>
<td>voltage at the positive terminal</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
</tr>
<tr>
<td>vMinus</td>
<td>voltage at the negative terminal</td>
<td>((-\infty, \infty))</td>
<td>real, complex</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples
\[ a = \text{vt}_\text{tran}(v1, v2) \]

Defined in
$\text{HPEESOF\_DIR}/\text{expressions/ael/circuit\_fun.ael}$

See Also
vt()
Transient Analysis Functions

11-20  vt_tran()
Chapter 12: FrontPanel Eye Diagram Functions

This chapter describes the eye diagram FrontPanel and associated functions in detail. These functions are not generalized functions, they are provided specifically to support the eye diagram utility available from the Data Display.

E,F
“eye_binning()” on page 12-3
“eye_density()” on page 12-4
“FrontPanel_eye()” on page 12-5
“FrontPanel_eye_2d_indepvar_maximum_inner()” on page 12-7
“FrontPanel_eye_amplitude_histogram()” on page 12-8
“FrontPanel_eye_crossings()” on page 12-9
“FrontPanel_eye_delay()” on page 12-11
“FrontPanel_eye_fall_trace()” on page 12-13
“FrontPanel_eye_horizontal_histogram()” on page 12-15
“FrontPanel_eye_regular()” on page 12-17
“FrontPanel_eye_rise_trace()” on page 12-18
“FrontPanel_eye_risefall_marker()” on page 12-20
“FrontPanel_eye_topbase()” on page 12-22
“FrontPanel_get_histogram_mean_stdev()” on page 12-24
“FrontPanel_pp_rms_jitter()” on page 12-26
“FrontPanel_wave_1st_falling_edge_period()” on page 12-27
“FrontPanel_wave_1st_rising_edge_period()” on page 12-29
“FrontPanel_wave_1st_transition_fall_time()” on page 12-31
“FrontPanel_wave_1st_transition_rise_time()” on page 12-33
“FrontPanel_wave_datarate()” on page 12-35
“FrontPanel_wave_negative_pulse_width()” on page 12-36
“FrontPanel_wave_positive_pulse_width()” on page 12-38
“FrontPanel_wave_topbase()” on page 12-40
FrontPanel Eye Diagram Functions

**Working with the Eye Diagram FrontPanel**

The eye diagram FrontPanel is available from the Data Display Tools > Eye Diagram menu. The FrontPanel features two modes of operation: oscilloscope and eye/mask. For more information on working with the eye diagram FrontPanel, refer to the Data Display documentation.
eye_binning()

Returns density binning data

Syntax

```matlab
y = eye_binning(eye_data, Indep_bins, Dep_bins)
```

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>eye_data</td>
<td>eye diagram data</td>
<td>$(-\infty, \infty)$</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>Indep_bins</td>
<td>number of independent bins</td>
<td>$[1, \infty)$</td>
<td>integer</td>
<td>yes</td>
</tr>
<tr>
<td>Dep_bins</td>
<td>number of dependent bins</td>
<td>$[1, \infty)$</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

```matlab
Eye_bins = eye_binning(vout, 100, 70);
```

Defined in

$\$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael$

See Also

eye(), eye_density(), FrontPanel_eye(), FrontPanel_eye_2d_indepvar_maximum_inner(), FrontPanel_eye_amplitude_histogram(), FrontPanel_eye_crossings(), FrontPanel_eye_delay(), FrontPanel_eye_fall_trace(), FrontPanel_eye_horizontal_histogram(), FrontPanel_eye_regular(), FrontPanel_eye_rise_trace(), FrontPanel_eye_risefall_marker(), FrontPanel_eye_topbase(), FrontPanel_get_histogram_mean_stddev(), FrontPanel_pp_rms_jitter(), FrontPanel_wave_1st_falling_edge_period(), FrontPanel_wave_1st_rising_edge_period(), FrontPanel_wave_1st_transition_fall_time(), FrontPanel_wave_1st_transition_rise_time(), FrontPanel_wave_datacrate(), FrontPanel_wave_negative_pulse_width(), FrontPanel_wave_positive_pulse_width(), FrontPanel_wave_topbase()
FrontPanel Eye Diagram Functions

**eye_density()**

Returns density trace

**Syntax**

```matlab
y = eye_density(eye_data, Indep_bins, Dep_bins)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>eye_data</td>
<td>eye diagram data</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>Indep_bins</td>
<td>number of independent bins</td>
<td>[1, ∞)</td>
<td>integer</td>
<td>yes</td>
</tr>
<tr>
<td>Dep_bins</td>
<td>number of dependent bins</td>
<td>[1, ∞)</td>
<td>integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```matlab
Eye_bins = eye_density(vout, 100, 70);
```

**Defined in**

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael$

**See Also**

`eye()`, `eye_binning()`, `FrontPanel_eye()`, `FrontPanel_eye_2d_indepvar_maximum_inner()`, `FrontPanel_eye_amplitude_histogram()`, `FrontPanel_eye_crossings()`, `FrontPanel_eye_delay()`, `FrontPanel_eye_fall_trace()`, `FrontPanel_eye_horizontal_histogram()`, `FrontPanel_eye_regular()`, `FrontPanel_eye_rise_trace()`, `FrontPanel_eye_risefall_marker()`, `FrontPanel_eye_topbase()`, `FrontPanel_get_histogram_mean_stddev()`, `FrontPanel_pp_rms_jitter()`, `FrontPanel_wave_1st_falling_edge_period()`, `FrontPanel_wave_1st_rising_edge_period()`, `FrontPanel_wave_1st_transition_fall_time()`, `FrontPanel_wave_1st_transition_rise_time()`, `FrontPanel_wave_datarate()`, `FrontPanel_wave_negative_pulse_width()`, `FrontPanel_wave_positive_pulse_width()`, `FrontPanel_wave_topbase()`
FrontPanel_eye()

Creates data for an eye diagram plot with one bit shift per trace, adding an additional cycle to the number of cycles defined.

Syntax

\[ y = \text{FrontPanel\_eye}(\text{NRZ\_data}, \text{symbol\_Rate}, \text{Cycles}, \text{Delay}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRZ_data</td>
<td>either numeric data or a time domain waveform, typically NRZ data</td>
<td>((-\infty, \infty))</td>
<td>complex</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>symbolRate</td>
<td>bit rate of the channel</td>
<td>(0, \infty)</td>
<td>real</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Cycles</td>
<td>number of cycles to repeat</td>
<td>[1, \infty)</td>
<td>integer</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>Delay</td>
<td>sampling delay</td>
<td>(0, \infty)</td>
<td>integer, real</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

Examples

\[ y = \text{FrontPanel\_eye}(\text{NRZ\_data}, \text{symbol\_rate}) \]

Defined in

Built in

See Also

eye_binning(), eye_density(), FrontPanel_eye_2d_indepvar_maximum_inner(), FrontPanel_eye_amplitude_histogram(), FrontPanel_eye_crossings(), FrontPanel_eye_delay(), FrontPanel_eye_fall_trace(), FrontPanel_eye_horizontal_histogram(), FrontPanel_eye_regular(), FrontPanel_eye_rise_trace(), FrontPanel_eye_risefall_marker(), FrontPanel_eye_topbase(), FrontPanel_get_histogram_mean_stddev(), FrontPanel_pp_rms_jitter(), FrontPanel_wave_1st_falling_edge_period(), FrontPanel_wave_1st_rising_edge_period(), FrontPanel_wave_1st_transition_fall_time(), FrontPanel_wave_1st_transition_rise_time(), FrontPanel_wave_datarate(), FrontPanel_wave_negative_pulse_width(), FrontPanel_wave_positive_pulse_width(), FrontPanel_wave_topbase()
FrontPanel Eye Diagram Functions

**Notes/Equations**
This function will shift the eye trace by one bit when multiple eyes are plotted as opposed to eye() function.
FrontPanel_eye_2d_indepvar_maximum_inner()

Given two dimensional data, returns maximum value of the inner independent variable

Syntax
\[
y = \text{FrontPanel}_\text{eye}_2d_{\text{indeppvar}}_{\text{maximum}}_{\text{inner}}(\text{eye\_bin\_data})
\]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>eye_bin_data</td>
<td>eye binning data</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

maximum_indep=FrontPanel_eye_2d_indepvar_maximum_inner(eye_bin_data)

Defined in

$HPEESOF\_DIR/expressions/ael/FrontPanel\_fun.ael$

See Also

eye_binning(), eye_density(), FrontPanel_eye(),
FrontPanel_eye_amplitude_histogram(), FrontPanel_eye_crossings(),
FrontPanel_eye_delay(), FrontPanel_eye_fall_trace(),
FrontPanel_eye_horizontal_histogram(), FrontPanel_eye_regular(),
FrontPanel_eye_rise_trace(), FrontPanel_eye_risefall_marker(),
FrontPanel_eye_topbase(), FrontPanel_get_histogram_mean_stddev(),
FrontPanel_pp_rms_jitter(), FrontPanel_wave_1st_falling_edge_period(),
FrontPanel_wave_1st_rising_edge_period(),
FrontPanel_wave_1st_transition_fall_time(),
FrontPanel_wave_1st_transition_rise_time(), FrontPanel_wave_datarate(),
FrontPanel_wave_negative_pulse_width(), FrontPanel_wave_positive_pulse_width(),
FrontPanel_wave_topbase()

Notes/Equations

The FrontPanel_eye_2d_indepvar_maximum_inner() function essentially takes two dimensional binning data and returns maximum value of inner independent variable.
FrontPanel Eye Diagram Functions

**FrontPanel_eye_amplitude_histogram**

Returns amplitude histogram from binning data

**Syntax**

\[ y = \text{FrontPanel_eye_amplitude_histogram}(\text{data}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Eye binning data, usually the output of eye_binning function</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ \text{Find\_delay} = \text{FrontPanel_eye_amplitude_histogram}(\text{eye\_bin\_data}) \]

**Defined in**

$\text{HPESOF\_DIR/expressions/ael/FrontPanel\_fun.ael}$

**See Also**

- eye_binning()
- eye_density()
- FrontPanel_eye()
- FrontPanel_eye_2d_indepvar_maximum_inner()
- FrontPanel_eye_crossings()
- FrontPanel_eye_delay()
- FrontPanel_eye_fall_trace()
- FrontPanel_eye_horizontal_histogram()
- FrontPanel_eye_regular()
- FrontPanel_eye_rise_trace()
- FrontPanel_eye_risefall_marker()
- FrontPanel_eye_topbase()
- FrontPanel_get_histogram_mean_stddev()
- FrontPanel_pp_rms_jitter()
- FrontPanel_wave_1st_falling_edge_period()
- FrontPanel_wave_1st_rising_edge_period()
- FrontPanel_wave_1st_transition_fall_time()
- FrontPanel_wave_1st_transition_rise_time()
- FrontPanel_wave_datarate()
- FrontPanel_wave_negative_pulse_width()
- FrontPanel_wave_positive_pulse_width()
- FrontPanel_wave_topbase()

**Notes/Equations**

The FrontPanel_eye_amplitude_histogram() function calculates the amplitude histogram from eye binning data.
**FrontPanel_eye_crossings()**

Given lower boundary, top boundary and the data type returns the eye crossing points

**Syntax**

\[ y = \text{FrontPanel_eye_crossings}(\text{data}, \text{base_upper}, \text{top_lower}, \text{data_type}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>eye binning data</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base_upper</td>
<td>upper three sigma point of the base boundary</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>top_lower</td>
<td>lower three sigma point of the top boundary</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>data_type</td>
<td>data type, current version supports only &quot;NRZ&quot; data</td>
<td>string</td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
Eye_crossing_array = FrontPanel_eye_crossings(eye_bin_data, 0.2, 0.8, "NRZ")
Eye_crossing_1_level=Eye_crossing_array[0]
Eye_crossing_1_time=Eye_crossing_array[1]
Eye_crossing_2_level=Eye_crossing_array[2]
Eye_crossing_2_time=Eye_crossing_array[3]
```

**Defined in**

$\$HPEESOF\_DIR/expressions/ael/FrontPanel_fun.ael$

**See Also**

- `eye_binning()`, `eye_density()`, `FrontPanel_eye()`,
- `FrontPanel_eye_2d_indepvar_maximum_inner()`,
- `FrontPanel_eye_amplitude_histogram()`, `FrontPanel_eye_delay()`,
- `FrontPanel_eye_fall_trace()`, `FrontPanel_eye_horizontal_histogram()`,
- `FrontPanel_eye_regular()`, `FrontPanel_eye_rise_trace()`,
- `FrontPanel_eye_risefall_marker()`, `FrontPanel_eye_topbase()`,
- `FrontPanel_get_histogram_mean_stddev()`, `FrontPanel_pp_rms_jitter()`,
FrontPanel Eye Diagram Functions

FrontPanel_wave_1st_falling_edge_period(), FrontPanel_wave_1st_rising_edge_period(),
FrontPanel_wave_1st_transition_fall_time(), FrontPanel_wave_1st_transition_rise_time(), FrontPanel_wave_datarate(),
FrontPanel_wave_negative_pulse_width(), FrontPanel_wave_positive_pulse_width(),
FrontPanel_wave_topbase()

Notes/Equations
The FrontPanel_eye_crossings() function essentially takes the eye binning data, the top and base boundary limits, and the data type, and returns crossing points as an array.
FrontPanel_eye_delay()

Returns delay required for eye pattern positioning

Syntax

\[ y = \text{FrontPanel\_eye\_delay}(\text{Vout\_time}, \text{BitRate}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vout_time</td>
<td>time domain voltage waveform</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>BitRate</td>
<td>bit rate of the channel and is expressed in frequency units</td>
<td>[0, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ \text{Find\_delay} = \text{FrontPanel\_eye\_delay}(\text{vout}, 1e9) \]

Defined in

$\text{HPEESOF\_DIR/expressions/ael/FrontPanel\_fun.ael}$

See Also

\text{eye\_binning()}, \text{eye\_density()}, \text{FrontPanel\_eye()}, \text{FrontPanel\_eye\_2d\_indepvar\_maximum\_inner()}, \text{FrontPanel\_eye\_amplitude\_histogram()}, \text{FrontPanel\_eye\_crossings()}, \text{FrontPanel\_eye\_fall\_trace()}, \text{FrontPanel\_eye\_horizontal\_histogram()}, \text{FrontPanel\_eye\_regular()}, \text{FrontPanel\_eye\_rise\_trace()}, \text{FrontPanel\_eye\_risefall\_marker()}, \text{FrontPanel\_eye\_topbase()}, \text{FrontPanel\_get\_histogram\_mean\_stddev()}, \text{FrontPanel\_pp\_rms\_jitter()}, \text{FrontPanel\_wave\_1st\_falling\_edge\_period()}, \text{FrontPanel\_wave\_1st\_rising\_edge\_period()}, \text{FrontPanel\_wave\_1st\_transition\_fall\_time()}, \text{FrontPanel\_wave\_1st\_transition\_rise\_time()}, \text{FrontPanel\_wave\_datarate()}, \text{FrontPanel\_wave\_negative\_pulse\_width()}, \text{FrontPanel\_wave\_positive\_pulse\_width()}, \text{FrontPanel\_wave\_topbase()}$

Notes/Equations
FrontPanel Eye Diagram Functions

The FrontPanel_eye_delay() function determines eye mid crossing point of NRZ data and calculates the delay required to position 50% eye crossing at the center of unit interval.
FrontPanel_eye_fall_trace()

Returns eye falling edges

Syntax

\[ y = \text{FrontPanel\_eye\_fall\_trace}(\text{eye\_bin\_data}, \text{level\_zero}, \text{level\_one}, \text{low\_threshold}, \text{high\_threshold}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>eye_bin_data</td>
<td>eye binning data</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>level_zero</td>
<td>logic level zero amplitude</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>level_one</td>
<td>logic level one amplitude</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>low_threshold</td>
<td>low threshold percentage</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>high_threshold</td>
<td>high threshold percentage</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

```plaintext
Eye_falling_traces = FrontPanel_eye_fall_trace(eye_bin_data, 0, 1.8, 10, 90)
```

Defined in

$HPEESOF\_DIR/expressions/ael/FrontPanel\_fun.ael$

See Also

- `eye_binning()`, `eye_density()`, `FrontPanel\_eye()`, `FrontPanel\_eye_2d\_indepvart\_maximum\_inner()`, `FrontPanel\_eye\_amplitude\_histogram()`, `FrontPanel\_eye\_crossings()`, `FrontPanel\_eye\_delay()`, `FrontPanel\_eye\_horizontal\_histogram()`, `FrontPanel\_eye\_regular()`, `FrontPanel\_eye\_rise\_trace()`, `FrontPanel\_eye\_risefall\_marker()`, `FrontPanel\_eye\_topbase()`, `FrontPanel\_get\_histogram\_mean\_stddev()`, `FrontPanel\_pp\_rms\_jitter()`, `FrontPanel\_wave\_1st\_falling\_edge\_period()`, `FrontPanel\_wave\_1st\_rising\_edge\_period()`, `FrontPanel\_wave\_1st\_transition\_fall\_time()`, `FrontPanel\_wave\_1st\_transition\_rise\_time()`, `FrontPanel\_wave\_datarate()`, `FrontPanel\_eye\_fall\_trace()`
FrontPanel Eye Diagram Functions

*FrontPanel_wave_negative_pulse_width*, *FrontPanel_wave_positive_pulse_width*,
*FrontPanel_wave_topbase*

**Notes/Equations**

The `FrontPanel_eye_fall_trace` function essentially takes the eye binning data and returns the falling edges which crosses the lower and upper threshold points.
**FrontPanel_eye_horizontal_histogram()**

Given start, stop time and amplitude, returns horizontal histogram from the eye binning data

**Syntax**

\[ y = \text{FrontPanel_eye_horizontal_histogram}(\text{data}, \text{start\_time}, \text{stop\_time}, \text{start\_amp}, \text{stop\_amp}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>eye binning data</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>start_time</td>
<td>start time of horizontal histogram</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>stop_time</td>
<td>stop time of horizontal histogram</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>start_amp</td>
<td>start amplitude of horizontal histogram</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>stop_amp</td>
<td>start amplitude of horizontal histogram</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ \text{Eye\_histogram} = \text{FrontPanel\_eye\_horizontal\_histogram}(\text{eye\_bin\_data}, 0, 10e-9, -0.1, 0.1) \]

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/FrontPanel\_fun.ael}$

**See Also**

`eye\_binning()`, `eye\_density()`, `FrontPanel\_eye()`, `FrontPanel\_eye_2d\_indepvar\_maximum\_inner()`, `FrontPanel\_eye\_amplitude\_histogram()`, `FrontPanel\_eye\_crossings()`, `FrontPanel\_eye\_delay()`, `FrontPanel\_eye\_fall\_trace()`, `FrontPanel\_eye\_regular()`, `FrontPanel\_eye\_rise\_trace()`, `FrontPanel\_eye\_risefall\_marker()`, `FrontPanel\_eye\_topbase()`, `FrontPanel\_get\_histogram\_mean\_stddev()`, `FrontPanel\_pp\_rms\_jitter()`, `FrontPanel\_wave\_1st\_falling\_edge\_period()`, `FrontPanel\_wave\_1st\_rising\_edge\_period()`, `FrontPanel\_wave\_1st\_transition\_fall\_time()`
FrontPanel Eye Diagram Functions

FrontPanel_wave_1st_transition_rise_time(), FrontPanel_wave_datarate(),
FrontPanel_wave_negative_pulse_width(), FrontPanel_wave_positive_pulse_width(),
FrontPanel_wave_topbase()

Notes/Equations
The FrontPanel_eye_horizontal_histogram() function essentially takes the eye
binning data and returns horizontal histogram between start and stop points.
FrontPanel_eye_regular()

Given eye data and the resolution, returns uniformly spaced eye data

Syntax
\[ y = \text{FrontPanel}_\text{eye}_\text{regular}(\text{eye}_\text{data}, \text{resolution}_x) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>eye_data</td>
<td>eye diagram data</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>resolution_x</td>
<td>inner independent variable resolution</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ \text{eye}_\text{data}=\text{eye}(\text{vout},1\text{GHz}) \]
\[ \text{uniform}_\text{eye}_\text{data}=\text{FrontPanel}_\text{eye}_\text{regular}(\text{eye}_\text{data},450) \]

Defined in

$\text{HPEESOF}\_\text{DIR}/\text{expressions/ael/FrontPanel}\_\text{fun.ael}$

See Also

\text{eye}\_\text{binning}(), \text{eye}\_\text{density}(), \text{FrontPanel}_\text{eye}(),
\text{FrontPanel}_\text{eye}\_\text{2d}\_\text{indepvar}\_\text{maximum}\_\text{inner}(),
\text{FrontPanel}_\text{eye}\_\text{amplitude}\_\text{histogram}(), \text{FrontPanel}_\text{eye}\_\text{crossings}(),
\text{FrontPanel}_\text{eye}\_\text{delay}(), \text{FrontPanel}_\text{eye}\_\text{fall}\_\text{trace}(),
\text{FrontPanel}_\text{eye}\_\text{horizontal}\_\text{histogram}(), \text{FrontPanel}_\text{eye}\_\text{rise}\_\text{trace}(),
\text{FrontPanel}_\text{eye}\_\text{rise}\_\text{fall}\_\text{marker}(), \text{FrontPanel}_\text{eye}\_\text{topbase}(),
\text{FrontPanel}\_\text{get}\_\text{histogram}\_\text{mean}\_\text{stddev}(), \text{FrontPanel}_\text{pp}\_\text{rms}\_\text{jitter}(),
\text{FrontPanel}_\text{wave}\_\text{1st}\_\text{falling}\_\text{edge}\_\text{period}(),
\text{FrontPanel}_\text{wave}\_\text{1st}\_\text{rising}\_\text{edge}\_\text{period}(),
\text{FrontPanel}_\text{wave}\_\text{1st}\_\text{transition}\_\text{fall}\_\text{time}(),
\text{FrontPanel}_\text{wave}\_\text{1st}\_\text{transition}\_\text{rise}\_\text{time}(), \text{FrontPanel}_\text{wave}\_\text{datarate}(),
\text{FrontPanel}_\text{wave}\_\text{negative}\_\text{pulse}\_\text{width}(), \text{FrontPanel}_\text{wave}\_\text{positive}\_\text{pulse}\_\text{width}(),
\text{FrontPanel}_\text{wave}\_\text{topbase}()\

Notes/Equations

The \text{FrontPanel}_\text{eye}_\text{regular}() function essentially takes eye data, and returns uniformly spaced data. The first and last trace of the eye diagram will be removed when using this function.
FrontPanel Eye Diagram Functions

**FrontPanel_eye_rise_trace()**

Returns eye rising edges

**Syntax**

\[ y = \text{FrontPanel\_eye\_rise\_trace}(\text{data}, \text{level\_zero}, \text{level\_one}, \text{low\_threshold}, \text{high\_threshold}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>eye binning data</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>level_zero</td>
<td>logic level zero amplitude</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>level_one</td>
<td>logic level one amplitude</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>low_threshold</td>
<td>low threshold percentage</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>high_threshold</td>
<td>high threshold percentage</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ \text{Eye\_rising\_traces} = \text{FrontPanel\_eye\_rise\_trace}(\text{eye\_bin\_data}, 0, 1.8, 10, 90) \]

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/FrontPanel\_fun.ael}$

**See Also**

\text{eye\_binning()}, \text{eye\_density()}, \text{FrontPanel\_eye()},
\text{FrontPanel\_eye\_2d\_indepvar\_maximum\_inner()},
\text{FrontPanel\_eye\_amplitude\_histogram()}, \text{FrontPanel\_eye\_crossings()},
\text{FrontPanel\_eye\_delay()}, \text{FrontPanel\_eye\_fall\_trace()},
\text{FrontPanel\_eye\_horizontal\_histogram()}, \text{FrontPanel\_eye\_regular()},
\text{FrontPanel\_eye\_risefall\_marker()}, \text{FrontPanel\_eye\_topbase()},
\text{FrontPanel\_get\_histogram\_mean\_stddev()}, \text{FrontPanel\_pp\_rms\_jitter()},
\text{FrontPanel\_wave\_1st\_falling\_edge\_period()},
\text{FrontPanel\_wave\_1st\_rising\_edge\_period()},
\text{FrontPanel\_wave\_1st\_transition\_fall\_time()},
\text{FrontPanel\_wave\_1st\_transition\_rise\_time()}, \text{FrontPanel\_wave\_datarate}()
Notes/Equations

The FrontPanel_eye_rise_trace() function essentially takes the eye binning data and returns the rising edges which crosses the lower and upper threshold points.
FrontPanel Eye Diagram Functions

**FrontPanel_eye_risefall_marker()**

Returns the threshold points from rising or falling edge data

**Syntax**

\[ y = \text{FrontPanel\_eye\_risefall\_marker}(\text{data}, \text{level\_zero}, \text{level\_one}, \text{low\_threshold}, \text{high\_threshold}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>eye binning data</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>level_zero</td>
<td>logic level zero amplitude</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>level_one</td>
<td>logic level one amplitude</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>low_threshold</td>
<td>low threshold percentage</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>high_threshold</td>
<td>high threshold percentage</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ \text{Eye\_rise\_marker} = \text{FrontPanel\_eye\_risefall\_marker}(\text{eye\_bin\_data}, 0, 1.8, 10, 90) \]

**Defined in**

$\text{HPEESOF\_DIR/expressions/ael/FrontPanel\_fun.ael}$

**See Also**

- eye_binning(), eye_density(), FrontPanel_eye(),
- FrontPanel_eye_2d_indepvar_maximum_inner(),
- FrontPanel_eye_amplitude_histogram(), FrontPanel_eye_crossings(),
- FrontPanel_eye_delay(), FrontPanel_eye_fall_trace(),
- FrontPanel_eye_horizontal_histogram(), FrontPanel_eye_regular(),
- FrontPanel_eye_rise_trace(), FrontPanel_eye_topbase(),
- FrontPanel_get_histogram_mean_stddev(), FrontPanel_pp_rms_jitter(),
- FrontPanel_wave_1st_falling_edge_period(),
- FrontPanel_wave_1st_rising_edge_period(),
- FrontPanel_wave_1st_transition_fall_time(),
- FrontPanel_wave_1st_transition_rise_time(), FrontPanel_wave_datarate(),

12-20 FrontPanel_eye_risefall_marker()
FrontPanel_wave_negative_pulse_width(), FrontPanel_wave_positive_pulse_width(), FrontPanel_wave_topbase()

**Notes/Equations**

The FrontPanel_eye_risefall_marker() function essentially takes the eye binning data and returns the gaussian mean position at lower threshold, 50% crossing, and upper threshold crossing points as an array. The first element of the array will give the independent and dependent value at the lower threshold, the second element will give the 50% crossing point, and the third element will give the high threshold point.
FrontPanel Eye Diagram Functions

**FrontPanel_eye_topbase()**

Given lower boundary, top boundary and the data type returns the eye crossing points

**Syntax**

\[
y = \text{FrontPanel_eye_topbase}(\text{data}, \text{eye_crossing_points}, \text{lower_threshold}, \text{upper_threshold}, \text{data_type})
\]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>eye binning data</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>eye_crossing_points</td>
<td>array of eye crossing points, typically FrontPanel_eye_crossings function returned value</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>lower_threshold</td>
<td>lower threshold point %</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>upper_threshold</td>
<td>upper threshold point %</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>data_type</td>
<td>data type, current version supports only &quot;NRZ&quot; data</td>
<td></td>
<td>string</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
eye_crossing_points=FrontPanel_eye_crossings(eye_bin_data,-1.0,1,0,"NRZ")
eye_top_base = FrontPanel_eye_topbase(eye_bin_data,eye_crossing_points,20,80,"NRZ")
```

**Defined in**

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

**See Also**

`eye_binning(), eye_density(), FrontPanel_eye(), FrontPanel_eye_2d_indepvar_maximum_inner(), FrontPanel_eye_amplitude_histogram(), FrontPanel_eye_crossings(), FrontPanel_eye_delay(), FrontPanel_eye_fall_trace()`

12-22 FrontPanel_eye_topbase()
FrontPanel_eye_horizontal_histogram(), FrontPanel_eye_regular(),
FrontPanel_eye_rise_trace(), FrontPanel_eye_risefall_marker(),
FrontPanel_get_histogram_mean_stddev(), FrontPanel_pp_rms_jitter(),
FrontPanel_wave_1st_falling_edge_period(),
FrontPanel_wave_1st_rising_edge_period(),
FrontPanel_wave_1st_transition_fall_time(),
FrontPanel_wave_1st_transition_rise_time(), FrontPanel_wave_datarate(),
FrontPanel_wave_negative_pulse_width(), FrontPanel_wave_positive_pulse_width(),
FrontPanel_wave_topbase()

**Notes/Equations**

The `FrontPanel_eye_topbase()` function essentially takes the eye binning data, the crossing points, the lower and upper threshold percentage value, and returned top and base levels.
FrontPanel Eye Diagram Functions

**FrontPanel_get_histogram_mean_stddev()**

Given histogram data computes mean and standard deviation values

**Syntax**

\[ y = \text{Frontpanel_get_histogram_mean_stddev}(\text{data}, \text{start_bin}, \text{stop_bin}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>histogram data</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>start_bin</td>
<td>start bin number</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>stop_bin</td>
<td>stop bin number</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

\[
\text{histogram_statistics} = \text{Frontpanel_get_histogram_mean_stddev}(\text{histogram_data}, 10, 100)
\]

\[
\text{histogram_mean} = \text{histogram_statistics}[0]
\]

\[
\text{histogram_stddev} = \text{histogram_statistics}[1]
\]

**Defined in**

`$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael`

**See Also**

`eye_binning()`, `eye_density()`, `FrontPanel_eye()`,
`FrontPanel_eye_2d_indepvar_maximum_inner()`,
`FrontPanel_eye_amplitude_histogram()`, `FrontPanel_eye_crossings()`,
`FrontPanel_eye_delay()`, `FrontPanel_eye_fall_trace()`,
`FrontPanel_eye_horizontal_histogram()`, `FrontPanel_eye_regular()`,
`FrontPanel_eye_rise_trace()`, `FrontPanel_eye_risefall_marker()`,
`FrontPanel_eye_topbase()`, `FrontPanel_pp_rms_jitter()`,
`FrontPanel_wave_1st_falling_edge_period()`,
`FrontPanel_wave_1st_rising_edge_period()`,
`FrontPanel_wave_1st_transition_fall_time()`,
`FrontPanel_wave_1st_transition_rise_time()`, `FrontPanel_wave_datastrate()`,
`FrontPanel_wave_negative_pulse_width()`, `FrontPanel_wave_positive_pulse_width()`,
`FrontPanel_wave_topbase()`

**Notes/Equations**

---

12-24 FrontPanel_get_histogram_mean_stddev()
The Frontpanel_get_histogram_mean_stddev() function essentially takes histogram data and returns mean and standard deviation.
FrontPanel Eye Diagram Functions

**FrontPanel_pp_rms_jitter()**

Given histogram data computes peak to peak jitter value and RMS jitter value.

**Syntax**

```plaintext
y = FrontPanel_pp_rms_jitter(histogram_data)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram_data</td>
<td>histogram data</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
eye_jitter = FrontPanel_pp_rms_jitter(histogram_data)
peak2peak_jitter = eye_jitter[0]
rms_jitter = eye_jitter[1]
```

**Defined in**

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael$

**See Also**

- `eye_binning()`, `eye_density()`, `FrontPanel_eye()`, `FrontPanel_eye_2d_indepvar_maximum_inner()`, `FrontPanel_eye_amplitude_histogram()`, `FrontPanel_eye_crossings()`, `FrontPanel_eye_delay()`, `FrontPanel_eye_1st_falling_edge_period()`, `FrontPanel_eye_1st_rising_edge_period()`, `FrontPanel_eye_1st_transition_fall_time()`, `FrontPanel_eye_1st_transition_rise_time()`, `FrontPanel_wave_datarate()`, `FrontPanel_wave_negative_pulse_width()`, `FrontPanel_wave_positive_pulse_width()`, `FrontPanel_wave_topbase()`

**Notes/Equations**

The `FrontPanel_pp_rms_jitter()` function essentially takes histogram data, and returns peak to peak jitter by determining non-zero start and stop values and one standard deviation. The first element of the array represents peak-to-peak value and the second element represents one standard deviation.
FrontPanel_wave_1st_falling_edge_period()

Given time domain voltage waveform, the logic levels, and the threshold levels, returns first falling edge period.

Syntax

\[ y = \text{FrontPanel_wave_1st_falling_edge_period}(\text{data}, \text{base}, \text{top}, \text{base_threshold_pct}, \text{top_threshold_pct}) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>time domain voltage waveform</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base</td>
<td>logic level zero voltage level</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>top</td>
<td>logic level one voltage level</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base_threshold_pct</td>
<td>low threshold point</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>high_threshold_pct</td>
<td>high threshold point</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ \text{get_rise_time} = \text{FrontPanel_wave_1st_falling_edge_period}(\text{data}, -1.0, 1.0, 20, 80) \]

Defined in

$\text{HP EESOF_DIR/expressions/ael/FrontPanel_fun.ael}$

See Also

eye_binning(), eye_density(), FrontPanel_eye(),
FrontPanel_eye_2d_indepvar_maximum_inner(),
FrontPanel_eye_amplitude_histogram(), FrontPanel_eye_crossings(),
FrontPanel_eye_delay(), FrontPanel_eye_fall_trace(),
FrontPanel_eye_horizontal_histogram(), FrontPanel_eye_regular(),
FrontPanel_eye_rise_trace(), FrontPanel_eye_risefall_marker(),
FrontPanel_eye_topbase(), FrontPanel_get_histogram_mean_stddev(),
FrontPanel_pp_rms_jitter(), FrontPanel_wave_1st_rising_edge_period(),
FrontPanel_wave_1st_transition_fall_time(),
FrontPanel Eye Diagram Functions

FrontPanel_wave_1st_transition_rise_time(), FrontPanel_wave_datarate(),
FrontPanel_wave_negative_pulse_width(), FrontPanel_wave_positive_pulse_width(),
FrontPanel_wave_topbase()

Notes/Equations

Period is a measure of the time between the first detected edge of a waveform and the
next occurrence of the same type of edge. The time between the edges of the
waveform used for measurement is taken at the middle threshold crossings. The
FrontPanel_wave_1st_falling_edge_period() function essentially takes time domain
data and returns the first falling edge period.
FrontPanel_wave_1st_rising_edge_period()

Given time domain voltage waveform, the logic levels, and the threshold levels, returns first rising edge period

Syntax

\[
y = \text{FrontPanel\_wave\_1st\_rising\_edge\_period}(\text{data}, \text{base}, \text{top}, \text{base\_threshold\_pct}, \text{top\_threshold\_pct})
\]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>time domain voltage waveform</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base</td>
<td>logic level zero voltage level</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>top</td>
<td>logic level one voltage level</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base_threshold_pct</td>
<td>low threshold point</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>high_threshold_pct</td>
<td>high threshold point</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

get\_rising\_period=\text{FrontPanel\_wave\_1st\_rising\_edge\_period}(\text{data}, -1.0, 1.0, 20, 80)

Defined in

$\text{HPEESOF\_DIR/expressions/ael/FrontPanel\_fun.ael}$

See Also

\text{eye\_binning()}, \text{eye\_density()}, \text{FrontPanel\_eye()}, \text{FrontPanel\_eye\_2d\_indepvar\_maximum\_inner()}, \text{FrontPanel\_eye\_amplitude\_histogram()}, \text{FrontPanel\_eye\_crossings()}, \text{FrontPanel\_eye\_delay()}, \text{FrontPanel\_eye\_fall\_trace()}, \text{FrontPanel\_eye\_horizontal\_histogram()}, \text{FrontPanel\_eye\_regular()}, \text{FrontPanel\_eye\_rise\_trace()}, \text{FrontPanel\_eye\_risefall\_marker()}, \text{FrontPanel\_eye\_topbase()}, \text{FrontPanel\_get\_histogram\_mean\_stddev()}, \text{FrontPanel\_pp\_rms\_jitter()}, \text{FrontPanel\_wave\_1st\_falling\_edge\_period}()
FrontPanel Eye Diagram Functions

FrontPanel_wave_1st_transition_fall_time(),
FrontPanel_wave_1st_transition_rise_time(), FrontPanel_wave_datarate(),
FrontPanel_wave_negative_pulse_width(), FrontPanel_wave_positive_pulse_width(),
FrontPanel_wave_topbase()

Notes/Equations

Period is a measure of the time between the first detected edge of a waveform and the
next occurrence of the same type of edge. The time between the edges of the
waveform used for measurement is taken at the middle threshold crossings. The
FrontPanel_wave_1st_rising_edge_period() function essentially takes time domain
data and returns the first rising edge period.
FrontPanel_wave_1st_transition_fall_time()

Given time domain voltage waveform, the logic levels, and the threshold levels, returns wave first negative transition fall time

Syntax
y = FrontPanel_wave_1st_transition_fall_time(data,base,top,base_threshold_pct,top_threshold_pct)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>time domain voltage waveform</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base</td>
<td>logic level zero voltage level</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>top</td>
<td>logic level one voltage level</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base_threshold_pct</td>
<td>low threshold point</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>high_threshold_pct</td>
<td>high threshold point</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples
get_rise_time=FrontPanel_wave_1st_transition_fall_time(data,-1.0,1.0,20,80)

Defined in
$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also
eye_binning(), eye_density(), FrontPanel_eye(),
FrontPanel_eye_2d_indepvar_maximum_inner(),
FrontPanel_eye_amplitude_histogram(), FrontPanel_eye_crossings(),
FrontPanel_eye_delay(), FrontPanel_eye_fall_trace(),
FrontPanel_eye_horizontal_histogram(), FrontPanel_eye_regular(),
FrontPanel_eye_rise_trace(), FrontPanel_eye_risefall_marker(),
FrontPanel_eye_topbase(), FrontPanel_get_histogram_mean_stddev(),
FrontPanel_pp_rms_jitter(), FrontPanel_wave_1st_falling_edge_period(),
FrontPanel_wave_1st_rising_edge_period(),
FrontPanel Eye Diagram Functions

- `FrontPanel_wave_1st_transition_rise_time()`, `FrontPanel_wave_datarate()`, `FrontPanel_wave_negative_pulse_width()`, `FrontPanel_wave_positive_pulse_width()`, `FrontPanel_wave_topbase()`

**Notes/Equations**

The `FrontPanel_wave_1st_transition_rise_time()` function essentially takes time domain data and measures the fall time of the first negative edge of the waveform. The fall time will not be measured until the falling edge completes the transition through the upper and lower threshold.
FrontPanel_wave_1st_transition_rise_time()

Given time domain voltage waveform, the logic levels, and the threshold levels, returns wave first transition rise time

Syntax

\[
y = \text{FrontPanel_wave_1st_transition_rise_time}(\text{data, base, top, base_threshold_pct, top_threshold_pct})
\]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>time domain voltage waveform</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base</td>
<td>logic level zero voltage level</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>top</td>
<td>logic level one voltage level</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base_threshold_pct</td>
<td>low threshold point</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>high_threshold_pct</td>
<td>high threshold point</td>
<td>(-\infty, \infty)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

get_rise_time = FrontPanel_wave_1st_transition_rise_time(data, -1.0, 1.0, 20, 80)

Defined in

$\text{HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael}$

See Also

eye_binning(), eye_density(), FrontPanel_eye(), FrontPanel_eye_2d_indepvar_maximum_inner(), FrontPanel_eye_amplitude_histogram(), FrontPanel_eye_crossings(), FrontPanel_eye_delay(), FrontPanel_eye_fall_trace(), FrontPanel_eye_horizontal_histogram(), FrontPanel_eye_regular(), FrontPanel_eye_rise_trace(), FrontPanel_eye_risefall_marker(), FrontPanel_eye_topbase(), FrontPanel_get_histogram_mean_stddev(), FrontPanel_pp_rms_jitter(), FrontPanel_wave_1st_falling_edge_period(), FrontPanel_wave_1st_rising_edge_period(),
FrontPanel Eye Diagram Functions

*FrontPanel_wave_1st_transition_fall_time(),* *FrontPanel_wave_datarate(),*  
*FrontPanel_wave_negative_pulse_width(),* *FrontPanel_wave_positive_pulse_width(),*  
*FrontPanel_wave_topbase()*

**Notes/Equations**

The *FrontPanel_wave_1st_transition_rise_time()* function essentially takes time domain data and measures the rise time of the first positive edge of the waveform. The rise time will not be measured until the rising edge completes the transition through the lower and upper threshold.
FrontPanel_wave_datarate()

Estimate data rate from NRZ bitstream

Syntax

\[ y = \text{FrontPanel\_wave\_datarate}(Vout\_time, Data\_Type) \]

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vout_time</td>
<td>time domain voltage waveform</td>
<td>((-\infty, \infty))</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

Examples

\[ \text{Find\_datarate} = \text{FrontPanel\_wave\_datarate}(\text{Vout}, \text{"NRZ"}) \]

Defined in

$\text{HPEESOF\_DIR\_expressions\_ael/FrontPanel\_fun.ael}$

See Also

\[ \text{eye\_binning()}, \text{eye\_density()}, \text{FrontPanel\_eye()}, \]
\[ \text{FrontPanel\_eye\_2d\_indepvar\_maximum\_inner()}, \]
\[ \text{FrontPanel\_eye\_amplitude\_histogram()}, \text{FrontPanel\_eye\_crossings()}, \]
\[ \text{FrontPanel\_eye\_delay()}, \text{FrontPanel\_eye\_fall\_trace()}, \]
\[ \text{FrontPanel\_eye\_horizontal\_histogram()}, \text{FrontPanel\_eye\_regular()}, \]
\[ \text{FrontPanel\_eye\_rise\_trace()}, \text{FrontPanel\_eye\_risefall\_marker()}, \]
\[ \text{FrontPanel\_eye\_topbase()}, \text{FrontPanel\_get\_histogram\_mean\_stddev()}, \]
\[ \text{FrontPanel\_pp\_rms\_jitter()}, \text{FrontPanel\_wave\_1st\_falling\_edge\_period()}, \]
\[ \text{FrontPanel\_wave\_1st\_rising\_edge\_period()}, \]
\[ \text{FrontPanel\_wave\_1st\_transition\_fall\_time()}, \]
\[ \text{FrontPanel\_wave\_1st\_transition\_rise\_time()}, \]
\[ \text{FrontPanel\_wave\_negative\_pulse\_width()}, \text{FrontPanel\_wave\_positive\_pulse\_width()}, \]
\[ \text{FrontPanel\_wave\_topbase()} \]

Notes/Equations

The FrontPanel_wave_datarate() estimate the data rate of the NRZ bit stream.
FrontPanel Eye Diagram Functions

**FrontPanel_wave_negative_pulse_width()**

Given time domain voltage waveform, the logic levels, and the threshold levels, returns negative pulse width of the first cycle

**Syntax**

```matlab
y = FrontPanel_wave_negative_pulse_width(data, base, top, base_threshold_pct, top_threshold_pct)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>time domain voltage waveform</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base</td>
<td>logic level zero voltage level</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>top</td>
<td>logic level one voltage level</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base_threshold_pct</td>
<td>low threshold point</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>high_threshold_pct</td>
<td>high threshold point</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```matlab
get_negative_pulse_width=FrontPanel_wave_negative_pulse_width(data, -1.0, 1.0, 20, 80)
```

**Defined in**

`$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael`

**See Also**

- `eye_binning()`, `eye_density()`, `FrontPanel_eye()`,
- `FrontPanel_eye_2d_indepvar_maximum_inner()`,
- `FrontPanel_eye_amplitude_histogram()`, `FrontPanel_eye_crossings()`,
- `FrontPanel_eye_delay()`, `FrontPanel_eye_fall_trace()`,
- `FrontPanel_eye_horizontal_histogram()`, `FrontPanel_eye_regular()`,
- `FrontPanel_eye_rise_trace()`, `FrontPanel_wave_1st_falling_edge_period()`,
- `FrontPanel_wave_1st_falling_edge_period()`,
- `FrontPanel_pp_rms_jitter()`,

12-36  FrontPanel_wave_negative_pulse_width()
FrontPanel_wave_1st_rising_edge_period(),
FrontPanel_wave_1st_transition_fall_time(),
FrontPanel_wave_1st_transition_rise_time(), FrontPanel_wave_datarate(),
FrontPanel_wave_positive_pulse_width(), FrontPanel_wave_topbase()

Notes/Equations
Negative pulse width is defined as the time from the mid-threshold of the first falling edge to the mid-threshold of the next rising edge. The FrontPanel_wave_negative_pulse_width() function essentially takes time domain data and returns the first negative pulse width.
FrontPanel Eye Diagram Functions

**FrontPanel_wave_positive_pulse_width()**

Given time domain voltage waveform, the logic levels, and the threshold levels, returns positive pulse width of the first cycle.

**Syntax**

\[
y = \text{FrontPanel_wave_positive_pulse_width}(\text{data}, \text{base}, \text{top}, \text{base_threshold_pct}, \text{top_threshold_pct})
\]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>time domain voltage waveform</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base</td>
<td>logic level zero voltage level</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>top</td>
<td>logic level one voltage level</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>base_threshold_pct</td>
<td>low threshold point</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>high_threshold_pct</td>
<td>high threshold point</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

get_negative_pulse_width = FrontPanel_wave_positive_pulse_width(data, -1.0, 1.0, 20, 80)

**Defined in**

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

**See Also**

eye_binning(), eye_density(), FrontPanel_eye(), FrontPanel_eye_2d_indepvar_maximum_inner(), FrontPanel_eye_amplitude_histogram(), FrontPanel_eye_crossings(), FrontPanel_eye_delay(), FrontPanel_eye_fall_trace(), FrontPanel_eye_horizontal_histogram(), FrontPanel_eye_regular(), FrontPanel_eye_rise_trace(), FrontPanel_eye_risefall_marker(), FrontPanel_eye_topbase(), FrontPanel_get_histogram_mean_stddev(), FrontPanel_pp_rms_jitter(), FrontPanel_wave_1st_falling_edge_period()
Notes/Equations

Positive pulse width is defined as the time from the mid-threshold of the first rising edge to the mid-threshold of the next falling edge. The FrontPanel_wave_positive_pulse_width() function essentially takes time domain data and returns the first positive pulse width.
FrontPanel Eye Diagram Functions

**FrontPanel_wave_topbase()**

Given amplitude histogram of a time domain waveform, returns logic level one and zero statistics

**Syntax**

```plaintext
y = FrontPanel_wave_topbase(data, data_type)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>amplitude histogram data</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
<tr>
<td>data_type</td>
<td>data type, only &quot;NRZ&quot; data is supported in current release</td>
<td>(-∞, ∞)</td>
<td>real</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
get_level=FrontPanel_wave_topbase(amplitude_histogram,"NRZ")
top_data=get_level[0::3]
level_one_mean=top_data[0]
level_one_stddev=top_data[1]
level_one_start=top_data[2]
level_one_stop=top_data[3]
base_data=get_level[4::7]
level_zero_mean=base_data[0]
level_zero_stddev=base_data[1]
level_zero_start=base_data[2]
level_zero_stop=base_data[3]
```

**Defined in**

$HPE ESOF_DIR/expressions/ael/FrontPanel_fun.ael

**See Also**

`eye_binning()`, `eye_density()`, `FrontPanel_eye()`,
`FrontPanel_eye_2d_indepvar_maximum_inner()`,
`FrontPanel_eye_amplitude_histogram()`, `FrontPanel_eye_crossings()`,
`FrontPanel_eye_delay()`, `FrontPanel_eye_fall_trace()`,
`FrontPanel_eye_horizontal_histogram()`, `FrontPanel_eye_regular()`,
`FrontPanel_eye_rise_trace()`, `FrontPanel_eye_risefall_marker()`,
`FrontPanel_eye_topbase()`, `FrontPanel_get_histogram_mean_stddev()`,
`FrontPanel_pp_rms_jitter()`, `FrontPanel_wave_1st_falling_edge_period()`,
FrontPanel_wave_1st_rising_edge_period(),
FrontPanel_wave_1st_transition_fall_time(),
FrontPanel_wave_1st_transition_rise_time(), FrontPanel_wave_datarate(),
FrontPanel_wave_negative_pulse_width(), FrontPanel_wave_positive_pulse_width()

Notes/Equations
The FrontPanel_wave_topbase() function essentially takes amplitude histogram data and returns logic level one and zero statistics as an array.
FrontPanel Eye Diagram Functions
## Index

### A
- abcdtoh, 9-3
- abcdtos, 9-4
- abcdtoy, 9-5
- abcdtoz, 9-6
- abs, 7-4
- acos, 7-5
- acosh, 7-6
- acot, 7-7
- acoth, 7-8
- acpr_v1, 4-3
- acpr_vr, 4-5
- add_Tf, 8-2
- asin, 7-9
- asinh, 7-10
- atan, 7-11
- atan2, 7-12
- atanh, 7-13

### B
- Backward Traveling Waves, 3-3
- bandwidth_func, 9-7
- ber_pi4dpsk, 8-3
- ber_qpsk, 8-5
- bud_freq, 3-4
- bud_gain, 3-7
- bud_gain_comp, 3-10
- bud_gamma, 3-13
- bud_l3p3_deg, 3-15
- bud_lnf, 3-17
- bud_lnfd_deg, 3-19
- bud_noise_pwr, 3-21
- bud_pwr, 3-23
- bud_pwr_inc, 3-25
- bud_pwr_refl, 3-27
- bud_snr, 3-29
- bud_sn, 3-31
- bud_vswr, 3-33
- Budget Measurement, 3-1
- build_subrange, 5-2
- Built-In Constants, 1-5

### C
- carr_to_im, 6-2
- Case Sensitivity, 1-4
- cdf, 10-2
- cdrange, 6-3
- ceil, 7-14
- center_freq, 9-9
- channel_power_vi, 4-8
- channel_power_vr, 4-10
- chop, 5-3
- chr, 5-4
- cint, 7-15
- circle, 5-5
- cmplx, 7-16
- collapse, 5-6
- complex, 7-17
- Component Options, 2-2
- conditional expressions, 1-6
- conj, 7-18
- const_evm, 4-12
- constellation, 11-2
- contour, 5-8
- contour_polar, 5-10
- convBin, 7-19
- convHex, 7-20
- convInt, 7-21
- convOct, 7-22
- copy, 5-12
- cos, 7-23
- cosh, 7-24
- cot, 7-25
- coth, 7-26
- create, 5-13
- cross, 11-4
- cross_corr, 10-3
- cross_hist, 4-15
- cum_prod, 7-27
- cum_sum, 7-28

### D
- db, 7-29
- dbm, 7-30
- dbmtow, 7-32
- dc_to_rf, 6-4
- deg, 7-33
- delay_path, 4-17
- delete, 5-16
- dev_lin_gain, 9-10
dev_lin_phase, 9-11
diagonal, 7-34
diff, 7-35
Display Parameter on schematic, 2-2

E
Envelope Data, 4-1
erf, 7-36
erfc, 7-37
erfcinv, 7-38
erfinv, 7-39
evm_wlan_dsss_cck_pbcc, 4-18
evm_wlan_ofdm, 4-28
Examples Using the DesignGuides, 2-9
exp, 7-40
expand, 5-17
eye, 8-7
eye_amplitude, 8-8
eye_binning, 12-3
eye_closure, 8-10
eye_density, 12-4
eye_fall_time, 8-12
eye_height, 8-14
eye_rise_time, 8-16

F
fft, 7-41
find, 5-19
find_index, 5-21
fix, 7-42
float, 7-43
floor, 7-44
fmod, 7-45
Frequency Plan, 3-2
FrontPanel_eye, 12-5
FrontPanel_eye_2d_indepvar_maximum_inner, 12-7
FrontPanel_eye_amplitude_histogram, 12-8
FrontPanel_eye_crossings, 12-9
FrontPanel_eye_delay, 12-11
FrontPanel_eye_fall_trace, 12-13
FrontPanel_eye_horizontal_histogram, 12-15
FrontPanel_eye_regular, 12-17
FrontPanel_eye_rise_trace, 12-18
FrontPanel_eye_risefall_marker, 12-20
FrontPanel_eye_topbase, 12-22
FrontPanel_get_histogram_mean_stddev, 12-24
FrontPanel_pp_rms_jitter, 12-26
FrontPanel_wave_1st_falling_edge_period, 12-27
FrontPanel_wave_1st_falling_edge_period, 12-29
FrontPanel_wave_1st_transition_fall_time, 12-31
FrontPanel_wave_1st_transition_rise_time, 12-33
FrontPanel_wave_data_rate, 12-35
FrontPanel_wave_negative_pulse_width, 12-36
FrontPanel_wave_positive_pulse_width, 12-38
FrontPanel_wave_topbase, 12-40
fs, 4-37
fspot, 11-5
fun_2d_outer, 10-4

G
ga_circle, 9-12
gain_comp, 9-14
generate, 5-22
Generating Data, 1-8
get_attr, 5-23
get_indep_values, 5-24
gl_circle, 9-15
gp_circle, 9-17
gs_circle, 9-19

H
Harmonic Balance Data, 6-1
histogram, 10-6
histogram_multiDim, 10-8
histogram_sens, 10-9
histogram_stat, 10-11
htoabcd, 9-21
htos, 9-22
htoy, 9-23
htoz, 9-24
hypot, 7-46
identity, 7-47
ifc, 6-5
ifc_tran, 11-9
Index-4

pspec, 6-20
pspec_tran, 11-14
pt, 6-21
pt_tran, 11-13
pwr_gain, 9-39
pwr_vs_t, 4-50

R
rad, 7-75
re, 7-76
real, 7-77
Reflections, 3-3
relative_noise_bw, 4-51
remove_noise, 6-22
ripple, 9-40
rms, 7-78
round, 7-79

S
s_stab_circle, 9-41
s_stab_circle_center_radius, 9-42
s_stab_region, 9-43
sample_delay_pi4dqpsk, 4-53
sample_delay_qpsk, 4-54
Select Parameter, 2-2
set_attr, 5-33
sfdr, 6-23
sgn, 7-80
Simulation Data, 1-7
sin, 7-81
sinc, 7-82
sinh, 7-83
size, 5-34
sm_gamma1, 9-44
sm_gamma2, 9-45
sm_y1, 9-46
sm_y2, 9-47
sm_z1, 9-48
sm_z2, 9-49
snr, 6-25
sort, 5-35
S-parameters, 1-9
spec_power, 8-18
spectrum_analyzer, 4-55
spur_track, 6-27
spur_track_with_if, 6-29
sqr, 7-84
sqrt, 7-85
stab_fact, 9-50
stab_meas, 9-51
Stability, 9-32, 9-33, 9-50, 9-51
stddev, 10-25
stddev_outer, 10-26
step, 7-86
stoh, 9-53
stos, 9-54
stot, 9-56
stoy, 9-57
stoz, 9-58
sum, 7-87
sweep_dim, 5-36
sweep_size, 5-37

T
tan, 7-88
tanh, 7-89
tdr_sp_gamma, 9-59
tdr_sp_imped, 9-61
tdr_step_imped, 9-63
thd_func, 6-31
total_pwr, 4-61
trajectory, 4-62
Transient Data, 11-1
transpose, 7-90
ts, 6-32
ttos, 9-64
type, 5-39

U
uniform_dist_inv1D, 10-27
uniform_dist1D, 10-28
unilateral_figure, 9-65
unwrap, 9-67
User-Defined Functions, 1-10

V
v_dc, 9-68
Variable Names, 1-4
vfc, 6-35
vfc_tran, 11-16
volt_gain, 9-69
volt_gain_max, 9-71
vs, 5-40
vspec, 6-37
vspec_tran, 11-17
vswr, 9-72
vt, 6-38
vt_tran, 11-19

W
what, 5-41
write_snp, 9-73
write_var, 5-42
wtodbm, 7-91

X
xor, 7-92

Y
yield_sens, 10-29
yin, 9-75
yopt, 9-76
ytoabcd, 9-77
ytoh, 9-78
ytos, 9-79
ytoz, 9-80

Z
zeros, 7-93
zin, 9-81
zopt, 9-82
ztoabcd, 9-83
ztoh, 9-84
ztos, 9-85
zttoy, 9-86