Notice

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

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

Warranty

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

Restricted Rights Legend

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

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

Contents

1 Introduction to Expressions and Functions
   Simulator Expressions .......................................................... 1-1
   Measurement Equations ...................................................... 1-1
   AEL Math Functions .......................................................... 1-2

2 Using the MeasEqn Function Reference
   Manipulating Simulation Data with Equations ........................... 2-2
     Simulation Data ................................................................. 2-2
     Case Sensitivity ............................................................... 2-2
     Measurements and Expressions .......................................... 2-2
     Variable Names ............................................................... 2-3
     Simple Sweeps and Using ‘[]’ ........................................... 2-4
     S-Parameters and Matrices ................................................. 2-5
     Matrices ........................................................................ 2-5
     Multidimensional Sweeps and Indexing ............................... 2-5
     Working with Harmonic Balance (HB) Data ......................... 2-6
     Working with Transient Data ............................................ 2-6
     Working with Envelope Data ............................................ 2-6
     if-then-else Construct .................................................... 2-7
     Generating Data ............................................................ 2-7
     Operator Precedence ....................................................... 2-7
     Built-in Constants .......................................................... 2-9
     Budget Measurement Analysis ........................................... 2-10
     MeasEqn ....................................................................... 2-11
     User-Defined Functions .................................................. 2-12
   Measurement Expressions (RFDE Only) ................................. 2-13
   Expression Builder (RFDE Only) .......................................... 2-14

3 MeasEqn Function Reference
   abcdtoh() ........................................................................ 3-2
   abcdtos() ........................................................................ 3-3
   abcdtoy() ........................................................................ 3-4
   abcdtoz() ........................................................................ 3-5
   abs() ............................................................................... 3-6
   acos() .............................................................................. 3-7
   acpr_vi() .......................................................................... 3-8
   acpr_vr() .......................................................................... 3-9
   add_rf() ........................................................................... 3-10
   asin() ............................................................................... 3-14
   atan() ............................................................................... 3-15
atan2() ...................................................................................................................... 3-16
ber_pi4dpsk() ........................................................................................................... 3-17
ber_qpsk() ................................................................................................................ 3-19
bud_freq() ................................................................................................................. 3-21
bud_gain() ................................................................................................................ 3-24
bud_gain_comp() ...................................................................................................... 3-26
bud_gamma() ........................................................................................................... 3-28
bud_ip3_deg() .......................................................................................................... 3-30
bud_nf() .................................................................................................................... 3-32
bud_nf_deg() ............................................................................................................ 3-34
bud_noise_pwr() ..................................................................................................... 3-36
bud_pwr() ................................................................................................................ 3-38
bud_pwr_inc() .......................................................................................................... 3-40
bud_pwr_refl() ........................................................................................................ 3-42
bud_snr() .................................................................................................................. 3-44
bud_tni() ................................................................................................................... 3-46
bud_vswr() ................................................................................................................ 3-48
carr_to_im() ........................................................................................................... 3-50
cdf() .......................................................................................................................... 3-51
cdrange() .................................................................................................................. 3-52
channel_power_vi() ............................................................................................... 3-53
channel_power_vr() ............................................................................................... 3-55
chop() ....................................................................................................................... 3-57
chr() .......................................................................................................................... 3-58
circle() ..................................................................................................................... 3-59
cint() ........................................................................................................................ 3-60
cmplx() .................................................................................................................... 3-61
conj() ....................................................................................................................... 3-62
const_evm() ............................................................................................................. 3-63
constellation() ....................................................................................................... 3-66
contour() .................................................................................................................. 3-69
contour_polar() ..................................................................................................... 3-70
cos() ........................................................................................................................ 3-71
cosh() ...................................................................................................................... 3-72
cross() ...................................................................................................................... 3-73
cross_corr() ............................................................................................................ 3-74
cum_prod() .............................................................................................................. 3-75
cum_sum() .............................................................................................................. 3-76
dB() .......................................................................................................................... 3-77
dBm() ....................................................................................................................... 3-78
dBmtoW() .................................................................................................................. 3-80
dc_to_rf() .................................................................................................................. 3-81
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln()</td>
<td>3-129</td>
</tr>
<tr>
<td>log()</td>
<td>3-130</td>
</tr>
<tr>
<td>log10()</td>
<td>3-131</td>
</tr>
<tr>
<td>mag()</td>
<td>3-132</td>
</tr>
<tr>
<td>map1_circle()</td>
<td>3-133</td>
</tr>
<tr>
<td>map2_circle()</td>
<td>3-134</td>
</tr>
<tr>
<td>max()</td>
<td>3-135</td>
</tr>
<tr>
<td>max_gain()</td>
<td>3-136</td>
</tr>
<tr>
<td>max_index()</td>
<td>3-137</td>
</tr>
<tr>
<td>max_outer()</td>
<td>3-138</td>
</tr>
<tr>
<td>mean()</td>
<td>3-139</td>
</tr>
<tr>
<td>mean_outer()</td>
<td>3-140</td>
</tr>
<tr>
<td>median()</td>
<td>3-141</td>
</tr>
<tr>
<td>min()</td>
<td>3-142</td>
</tr>
<tr>
<td>min_index()</td>
<td>3-143</td>
</tr>
<tr>
<td>min_outer()</td>
<td>3-144</td>
</tr>
<tr>
<td>mix()</td>
<td>3-145</td>
</tr>
<tr>
<td>moving_average()</td>
<td>3-146</td>
</tr>
<tr>
<td>mu()</td>
<td>3-147</td>
</tr>
<tr>
<td>mu_prime()</td>
<td>3-148</td>
</tr>
<tr>
<td>ns_circle()</td>
<td>3-149</td>
</tr>
<tr>
<td>ns_pwr_int()</td>
<td>3-150</td>
</tr>
<tr>
<td>ns_pwr_ref_bw()</td>
<td>3-151</td>
</tr>
<tr>
<td>ones()</td>
<td>3-152</td>
</tr>
<tr>
<td>pae()</td>
<td>3-153</td>
</tr>
<tr>
<td>pdf()</td>
<td>3-154</td>
</tr>
<tr>
<td>permute()</td>
<td>3-155</td>
</tr>
<tr>
<td>pfc()</td>
<td>3-156</td>
</tr>
<tr>
<td>pfc_tran()</td>
<td>3-157</td>
</tr>
<tr>
<td>phase()</td>
<td>3-158</td>
</tr>
<tr>
<td>phase_comp()</td>
<td>3-159</td>
</tr>
<tr>
<td>phasedeg()</td>
<td>3-160</td>
</tr>
<tr>
<td>phaserad()</td>
<td>3-161</td>
</tr>
<tr>
<td>plot_vs()</td>
<td>3-162</td>
</tr>
<tr>
<td>polar()</td>
<td>3-163</td>
</tr>
<tr>
<td>pow()</td>
<td>3-164</td>
</tr>
<tr>
<td>pspec()</td>
<td>3-165</td>
</tr>
<tr>
<td>pspec_tran()</td>
<td>3-166</td>
</tr>
<tr>
<td>prod()</td>
<td>3-167</td>
</tr>
<tr>
<td>pt()</td>
<td>3-168</td>
</tr>
<tr>
<td>pwr_gain()</td>
<td>3-169</td>
</tr>
<tr>
<td>rad()</td>
<td>3-170</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>real()</td>
<td>3-172</td>
</tr>
<tr>
<td>relative_noise_bw()</td>
<td>3-173</td>
</tr>
<tr>
<td>ripple()</td>
<td>3-175</td>
</tr>
<tr>
<td>round()</td>
<td>3-176</td>
</tr>
<tr>
<td>s_stab_circle()</td>
<td>3-177</td>
</tr>
<tr>
<td>s_stab_region()</td>
<td>3-178</td>
</tr>
<tr>
<td>sample_delay_pi4dqpsk()</td>
<td>3-179</td>
</tr>
<tr>
<td>sample_delay_qpsk()</td>
<td>3-180</td>
</tr>
<tr>
<td>set_attr()</td>
<td>3-181</td>
</tr>
<tr>
<td>sfdr()</td>
<td>3-182</td>
</tr>
<tr>
<td>sgn()</td>
<td>3-183</td>
</tr>
<tr>
<td>sin()</td>
<td>3-184</td>
</tr>
<tr>
<td>sinc()</td>
<td>3-185</td>
</tr>
<tr>
<td>sinh()</td>
<td>3-186</td>
</tr>
<tr>
<td>size()</td>
<td>3-187</td>
</tr>
<tr>
<td>sm_gamma1()</td>
<td>3-188</td>
</tr>
<tr>
<td>sm_gamma2()</td>
<td>3-189</td>
</tr>
<tr>
<td>sm_y1()</td>
<td>3-190</td>
</tr>
<tr>
<td>sm_y2()</td>
<td>3-191</td>
</tr>
<tr>
<td>sm_z1()</td>
<td>3-192</td>
</tr>
<tr>
<td>sm_z2()</td>
<td>3-193</td>
</tr>
<tr>
<td>snr()</td>
<td>3-194</td>
</tr>
<tr>
<td>sort()</td>
<td>3-195</td>
</tr>
<tr>
<td>spec_power()</td>
<td>3-196</td>
</tr>
<tr>
<td>spur_track()</td>
<td>3-198</td>
</tr>
<tr>
<td>spur_track_with_lif()</td>
<td>3-200</td>
</tr>
<tr>
<td>sqrt()</td>
<td>3-202</td>
</tr>
<tr>
<td>stab_fact()</td>
<td>3-203</td>
</tr>
<tr>
<td>stab_meas()</td>
<td>3-204</td>
</tr>
<tr>
<td>stddev()</td>
<td>3-205</td>
</tr>
<tr>
<td>stoabcd()</td>
<td>3-206</td>
</tr>
<tr>
<td>stoh()</td>
<td>3-207</td>
</tr>
<tr>
<td>stos()</td>
<td>3-208</td>
</tr>
<tr>
<td>stoy()</td>
<td>3-209</td>
</tr>
<tr>
<td>stoz()</td>
<td>3-210</td>
</tr>
<tr>
<td>sum()</td>
<td>3-211</td>
</tr>
<tr>
<td>sweep_dim()</td>
<td>3-212</td>
</tr>
<tr>
<td>sweep_size()</td>
<td>3-213</td>
</tr>
<tr>
<td>tan()</td>
<td>3-214</td>
</tr>
<tr>
<td>tanh()</td>
<td>3-215</td>
</tr>
<tr>
<td>trajectory()</td>
<td>3-216</td>
</tr>
<tr>
<td>transpose()</td>
<td>3-218</td>
</tr>
</tbody>
</table>
ts() ................................................................. 3-219
type() .................................................................. 3-221
unwrap() ............................................................ 3-222
vfc() ..................................................................... 3-223
vfc_tran() ........................................................... 3-224
volt_gain() .......................................................... 3-225
volt_gain_max() .................................................... 3-227
vs() ...................................................................... 3-228
vspec_tran() ........................................................ 3-229
vswr() .................................................................. 3-230
vt() ...................................................................... 3-231
vt_tran() .............................................................. 3-232
what() .................................................................. 3-233
yield_sens() .......................................................... 3-234
yin() .................................................................... 3-235
yopt() .................................................................. 3-236
ytoabed() ............................................................... 3-237
ytoh() .................................................................. 3-238
ytos() .................................................................. 3-239
ytoz() .................................................................. 3-240
zeros() ................................................................. 3-241
zin() ..................................................................... 3-242
zopt() .................................................................. 3-243
ztoabed() ............................................................... 3-244
ztoh() .................................................................. 3-245
ztos() .................................................................. 3-246
ztoy() .................................................................. 3-247

4 Simulator Expression Reference
   How to Enter Simulator Expressions in ADS .......... 4-1
   Simulator Expressions ........................................ 4-2
   Simulator Variables and Constants .................... 4-18
   Mathematical Operators and Hierarchy ............... 4-20

Index
Chapter 1: Introduction to Expressions and Functions

This document describes the expressions and functions that are available within Advanced Design System. These functions or equations are divided into two distinct categories based on their roles in ADS. Although there is an overlap among many of the more commonly used functions, they are derived from separate sources, evaluated by ADS at different times, and can have subtle differences in their usages. Thus, they need to be considered separately. Refer to Figure 1-1 in this chapter for an overview of how ADS evaluates and uses Simulator Expressions and Measurement Equations.

Simulator Expressions

The first category of equations or functions are the ones used internally during simulation runtime, known as Simulator Expressions or sometimes as VarEqn functions. These expressions or functions can be entered into the program by means of the VarEqn component or used in place of a parameter for any component: for example in a resistor, R=sin(5). These functions are evaluated at the start of simulation. If a term is undefined at the start of simulation, such as R=S(1,1), where the results of S(1,1) will not be known until the simulation is complete, an error will be returned.

For information on the general use of VarEqn components, place a VarEqn component on a Schematic, double click it, and then click the Help button in the Component dialog box. Or from any ADS Window choose Help > Topics and Index > Components > Circuit Components > Introduction and Simulation Components > Chapter 1, Introduction > VarEqn.

For a list and description of the Simulator Expressions, refer to Chapter 4, Simulator Expression Reference.

Measurement Equations

The second category of equations are used during simulation post processing, and are known as Measurement Equations or MeasEqn for short. These are entered into the program by means of the MeasEqn component, available on the Simulation palettes in an Analog/RF Systems Schematic window (such as Simulation-AC or
Introduction to Expressions and Functions

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. Many common expressions are included as measurements, which makes it easy for beginning users to use 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 AEL, ADS’s Application Extension Language. 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). 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.

For information on how to interpret the function descriptions found in the MeasEqn Function Reference, see Chapter 2, Using the MeasEqn Function Reference.

For a complete list of ADS MeasEqn functions, refer to Chapter 3, MeasEqn Function Reference.

**AEL Math Functions**

As stated earlier, the measurement components and their underlying expressions are based on ADS’s Application Extension Language (AEL). AEL includes many math functions that can be used as part of writing AEL code for various purposes. While many of these functions overlap with the Measurement Equations, they are documented in the AEL manual. Refer to Math Functions in the AEL manual for more information.
Figure 1-1. How ADS Evaluates and Uses Simulator Expressions and Measurement Equations
Chapter 2: Using the MeasEqn Function Reference

This chapter explains how to interpret the function descriptions found in Chapter 3, MeasEqn Function Reference.

The following figure illustrates how the measurement functions and mathematical functions are described. In the case of Application Extension Language (AEL in Advanced Design System) measurements, the entries for Used in and Available as measurement component? is noted as Not applicable.

<function name>

Purpose
States what the function does.

Synopsis
Presents the syntax of the function.

Examples
Presents typical uses of the function.

Used in
Lists applicable simulation types, if any.

Available as measurement component?
Indicates whether the measurement function is available as a component within simulation palettes (where applicable).

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

See also
Lists related functions, if any.
Manipulating Simulation Data with Equations

In Advanced Design System, you can also place a MeasEqn (measurement equation) component in a schematic window. Ready-made measurements, found in the various simulator palettes in ADS, are simply preconfigured equations. This description of ADS equations is accompanied by a set of example designs and data display pages. These designs can be found in the project express_meas_prj, in the examples/Tutorial directory in ADS.

Simulation Data

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

Case Sensitivity

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

Measurements and Expressions

In ADS, refer also to simple_meas_1.dsn in /examples/Tutorial/express_meas_prj/networks.

In Advanced Design System, expressions are available on the schematic page by means of the MeasEqn component. Also available in various simulation palettes are...
preconfigured measurements. These are designed to help the user by presenting an initial equation, which can be modified to suit the particular instance.

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.

It is not possible to reference a VarEqn variable in a MeasEqn equation. However, a MeasEqn equation can reference other MeasEqns, any simulation output, and any swept variable.

Variable Names

In ADS, refer also to names_1.dsn and names_1.dds in /examples/Tutorial/express_meas_prj.

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

DatasetName.AnalysisName.AnalysisType.CircuitPath.VariableName

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

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 ADS, refer also to names_2.dsn, and names_2.dds in /examples/Tutorial/express_meas_prj.
Using the MeasEqn Function Reference

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.

In ADS, refer also to names_3.dds in / examples/ Tutorial/ express_meas_prj.

**Simple Sweeps and Using “[ ]”**

In ADS, refer also to sweep_1.dsn, sweep_1.dsn and sweep_2.dds in / examples/ Tutorial/ express_meas_prj.

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

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

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

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

Because increment defaults to one,

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

is equivalent to

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

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

In ADS, refer also to sparam_1.dsn and sparam_1.dds in /examples/Tutorial/express_meas_prj.

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 Snm. 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 wildcards and sequences to extract a submatrix from an original matrix.

Matrices

In ADS, refer also to matrix_1.dds in /examples/Tutorial/express_meas_prj.

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 nonswept quantities to be combined. For example, the first matrix in a sweep may be subtracted from all matrices in that sweep as

\[ Y = X - X[0] \]

In ADS, refer also to matrix_2.dsn and matrix_2.dds in /examples/Tutorial/express_meas_prj.

Multidimensional Sweeps and Indexing

In ADS, refer also to multi_dim_1.dsn and multi_dim_1.dds in /examples/Tutorial/express_meas_prj.

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 wildcards are inserted automatically to the left. This is best explained by referring to the above example files.

**Working with Harmonic Balance (HB) Data**

In ADS, refer also to `hb_1.dds` in `/examples/Tutorial/express_meas_prj`.

Harmonic balance 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 "[ ]." Multitone HB also produces 1-dimensional data. Individual harmonic components can be indexed as usual by means of "[ ]." However, the "mix" function provides a convenient way to select a particular mixing component (for a list of functions, refer to List of Functions).

**Working with Transient Data**

In ADS, refer also to `tran_1.dsn` and `tran_1.dds` in `/examples/Tutorial/express_meas_prj`.

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 "[ ]."

**Working with Envelope Data**

In ADS, refer also to `env_1.dds` in `/examples/Tutorial/express_meas_prj`.

Envelope analysis produces complex frequency spectra as a function of time. A single envelope analysis can produce 2-dimensional data where the outermost independent variable is time and the innermost is frequency or harmonic number. Indexing can be used to look at a harmonic against time, or a spectrum at a particular time index.
if-then-else Construct

In ADS, refer also to if_then_else_1.dds in /examples/ Tutorial/ express_meas_prj.

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:

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

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:

\[ A = \text{if} ( \text{condition} ) \text{ then } \text{true_expression} \text{ elseif} ( \text{condition2} ) \text{ then } \text{true_expression} \text{ else } \text{false_expression} \]

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

The if-then-else construct can be used in a MeasEqn component on a schematic. It has the following syntax:

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

Generating Data

In ADS, refer also to gen_1.dds in /examples/ Tutorial/ express_meas_prj.

The simulator produces scalars and matrices. When a sweep is being performed it 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.

Operator Precedence

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

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

The operators >=, <=, >, <, ==, !=, AND, OR, EQUALS, and NOT EQUALS also produce logical results, producing a logical TRUE or FALSE upon comparing the values of two expressions. These operators are most often used to compare two real numbers or integers. These operators operate differently in AEL than C with string expressions in that they actually perform the equivalent of strcmp() between the first and second operands, and test the return value against 0 using the specified operator.

<table>
<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)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X(expr,expr)</td>
</tr>
<tr>
<td>[]</td>
<td>sweep indexer, sweep generator</td>
<td>X[expr_list]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[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</td>
</tr>
<tr>
<td></td>
<td>wildcard</td>
<td>start::inc::stop</td>
</tr>
</tbody>
</table>

Table 2-1. Operator Precedence

2-8 Manipulating Simulation Data with Equations
The following constants can be used in expressions.

### Table 2-2. 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>p</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/degree 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–sec</td>
</tr>
<tr>
<td>c0</td>
<td>Speed of light in free space</td>
<td>2.99792e+08 m/sec</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>
Budget Measurement Analysis

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

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

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

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

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

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

The budget measurements can be performed on arbitrary networks with multiple signal paths between the input and output ports. As a result, the measurements can be affected by reflection and noise generated by components placed between the terminal of interest and the output port on the same signal path or by components on different signal paths.

**Reflection and Backward-Traveling Wave Effects**

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

This sampler can be constructed using the equation-based linear three-port S-parameter component. To do this, set the elements of the scattering matrix as follows: \( S_{12} = S_{21} = S_{31} = 1 \), 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.

**MeasEqn**

By placing a MeasEqn (simulation measurement equation) component on an ADS schematic, you can write an equation that can be evaluated, following a simulation, and displayed in a Data Display window.

**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 "User's Guide".

User-Defined Functions
By writing some Application Extension Language (AEL in Advanced Design System) code, you can define your own custom functions. A file called user_defined_fun.ael has been set aside for this purpose in the directory expressions/ael/. By looking at the other_fun.ael files, you can see how to write your 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, in order to have your functions all compile, you will need to include a line such as

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

in the expressions_init.ael file in the same directory.
Measurement Expressions (RFDE Only)

In RF Design Environment, measurement expressions are specified through the Measurement Expressions form. Pre configured measurements are also available though this form in the “Expression Builder (RFDE Only)” on page 2-14. These are designed to help you build relevant expressions for a specific analysis.

Measurements are evaluated after a simulation is run and the results are stored in the dataset. Complex measurement equations are available for both circuit and signal processing simulations. Underlying a measurement is the same generic expressions 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, a Monte Carlo output or a Yield specification is a measurement expression. In general, a measurement expression can reference any other measurement expressions, simulation output, and swept or design variables.

To access the Measurement Expressions form from the main menu in the Circuit Design Environment window:

2. Click the **Build Expression** button. The Expression Builder form appears enabling you to select from a group of pre-defined expressions. For more information, refer to “Expression Builder (RFDE Only)” on page 2-14.

3. Select an enabled Analysis Type (i.e. DC, AC, HB, SP, Env, or Tran) using the button below the Analysis column of the Measurement Expressions form. The **Auto** button enables you to apply your expressions to all enabled analyses.

4. You can enable or disable the Autoplot feature (see Autoplot in the Data Display documentation) and/or a measurement expression by selecting yes or no from the button below the Plot and Enabled columns of the Measurement Expressions form.

5. Click the **Add** button to add your expression to the Expressions list. The completed expression is added to the Measurement Expressions form and numbered sequentially.

6. To remove an unwanted expression, select the expression in the Expressions list and then click the **Delete** button.

7. To modify an existing expression, select the expression in the Expressions list, make your modification, then click the **Update** button. Your expression is updated with the changes.

8. Once you have completed building your list of expressions, click the **OK** button in the Measurement Expressions form.

**Expression Builder (RFDE Only)**

The Expression Builder in RF Design Environment (RFDE) enables you to access a set of pre-defined measurement expressions (see Table 2-6) and can also help in constructing measurement expressions for defining Optimization, Statistical Analysis or Yield goals.

You can access the Expression Builder from the main menu or from several other tools provided in RFDE including:

- Monte Carlo - Tools > MonteCarlo/Yield > Monte Carlo > Add Output
- Yield - Tools > MonteCarlo/Yield > Yield > Add Specification
- Optimization - Tools > Optimization > Add Goal

To access the Expression Builder from the main menu in the Circuit Design Environment window:

2. Click the Build Expression button. The Expression Builder form appears.

3. Use the Analysis Type pull-down menu in the Expression Builder form to select an analysis type (see Table 2-3).

<table>
<thead>
<tr>
<th>Analysis Type</th>
<th>Measurement Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>AC Simulation</td>
</tr>
<tr>
<td>DC</td>
<td>DC Simulation</td>
</tr>
<tr>
<td>HB</td>
<td>Harmonic Balance Simulation</td>
</tr>
<tr>
<td>SP</td>
<td>S-parameter Simulation</td>
</tr>
<tr>
<td>Env</td>
<td>Circuit Envelope Simulation</td>
</tr>
<tr>
<td>Tran</td>
<td>Transient (Time-Domain) Simulation</td>
</tr>
</tbody>
</table>

4. Use the Measurement pull-down menu to select a specific measurement. This pull-down menu contains a set of pre-defined measurement expressions related to the analysis type selected. For detailed information, refer to “Pre-defined Measurement Expressions for Expression Builder” on page 2-16.

Note: Depending on how your environment is configured, you may need to expand the Expression Builder form to display all of the options available for each measurement.

5. Click the Select button next to the desired voltage or current and view the prompt in the schematic window. The prompt displayed will tell you what to select in the schematic window. The prompts are described in Table 2-4.
6. Select the signal necessary for the function. The net name will be added to the field for the item you selected.

7. For certain outputs, you can select a specific Window Type. Windowing is used to reduce discrete Fourier transform (DFT) leakage. For more information on Window Types, refer to “ts()” on page 3-219.

Depending on your selection, additional options are listed which enable you to further define your outputs. Several example options are shown in Table 2-5.

Table 2-5. Example Additonal Options

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Points</td>
<td>Set the number of points to be displayed. This field accepts either an integer or a string variable.</td>
</tr>
<tr>
<td>Stop Time</td>
<td>Sets the stop time. This field accepts either a Real number or a string variable.</td>
</tr>
<tr>
<td>Harmonic Index</td>
<td>This field accepts a Vector in the form [0, 2].</td>
</tr>
<tr>
<td>S-parameter</td>
<td>For Probe Circuit, this field is a selection field. For Expression Builder, string variables are accepted. e.g. S11 or S(12,12)</td>
</tr>
</tbody>
</table>

8. After selecting all of your voltages and/or currents and entering any other appropriate information, click OK in the Expression Builder form.

Pre-defined Measurement Expressions for Expression Builder

The pre-defined measurement expressions available in RF Design Environment Expression Builder are listed in Table 2-6. Each measurement expression includes a link to the function detail.
### Table 2-6. RF Design Environment Measurement Expressions

<table>
<thead>
<tr>
<th>Analysis Type</th>
<th>Measurement</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>DC Current</td>
<td>User Defined or Selected.</td>
</tr>
<tr>
<td></td>
<td>DC Voltage</td>
<td>User Defined or Selected.</td>
</tr>
<tr>
<td></td>
<td>DC Voltage Difference</td>
<td>User Defined or Selected.</td>
</tr>
<tr>
<td>AC</td>
<td>Signal to Noise Ratio</td>
<td>“snr()” on page 3-194</td>
</tr>
<tr>
<td></td>
<td>Voltage/Current Imaginary Part</td>
<td>“imag()” on page 3-116</td>
</tr>
<tr>
<td></td>
<td>Voltage/Current Magnitude</td>
<td>“mag()” on page 3-132</td>
</tr>
<tr>
<td></td>
<td>Voltage/Current Magnitude in dB</td>
<td>“dB()” on page 3-77</td>
</tr>
<tr>
<td></td>
<td>Voltage/Current Magnitude in dBm</td>
<td>“dBm()” on page 3-78</td>
</tr>
<tr>
<td></td>
<td>Voltage/Current Phase</td>
<td>“phase()” on page 3-159</td>
</tr>
<tr>
<td></td>
<td>Voltage/Current Real Part</td>
<td>“real()” on page 3-172</td>
</tr>
</tbody>
</table>
### Table 2-6. RF Design Environment Measurement Expressions

<table>
<thead>
<tr>
<th>Analysis Type</th>
<th>Measurement</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>HB</td>
<td>Carrier to IMD Power Ratio</td>
<td>“carr_to_im()” on page 3-50</td>
</tr>
<tr>
<td></td>
<td>DC-to-RF Efficiency</td>
<td>“dc_to_rf()” on page 3-81</td>
</tr>
<tr>
<td></td>
<td>Frequency-Selective Current</td>
<td>“ifc()” on page 3-114</td>
</tr>
<tr>
<td></td>
<td>Frequency-Selective Power</td>
<td>“pfc()” on page 3-157</td>
</tr>
<tr>
<td></td>
<td>Frequency-Selective Voltage</td>
<td>“vfc()” on page 3-223</td>
</tr>
<tr>
<td></td>
<td>Input Third-Order-Intercept Point</td>
<td>“ip3_in()” on page 3-122</td>
</tr>
<tr>
<td></td>
<td>Output Nth-Order-Intercept Point</td>
<td>“ipn()” on page 3-124</td>
</tr>
<tr>
<td></td>
<td>Output Third-Order-Intercept Point</td>
<td>“ip3_out()” on page 3-123</td>
</tr>
<tr>
<td></td>
<td>Power Spectrum</td>
<td>“pspec()” on page 3-166</td>
</tr>
<tr>
<td></td>
<td>Power-Added Efficiency</td>
<td>“pae()” on page 3-154</td>
</tr>
<tr>
<td></td>
<td>Signal to Noise Ratio</td>
<td>“snr()” on page 3-194</td>
</tr>
<tr>
<td></td>
<td>Spurious-Free Dynamic Range</td>
<td>“sfdr()” on page 3-182</td>
</tr>
<tr>
<td></td>
<td>Time-Domain Current</td>
<td>“it()” on page 3-126</td>
</tr>
<tr>
<td></td>
<td>Time-Domain Voltage</td>
<td>“vt()” on page 3-231</td>
</tr>
<tr>
<td></td>
<td>Time-Domain Waveform</td>
<td>“ts()” on page 3-219</td>
</tr>
<tr>
<td></td>
<td>Total Power</td>
<td>“pt()” on page 3-169</td>
</tr>
<tr>
<td></td>
<td>Voltage/Current Magnitude</td>
<td>“mag()” on page 3-132</td>
</tr>
<tr>
<td></td>
<td>Voltage/Current Magnitude in dB</td>
<td>“dB()” on page 3-77</td>
</tr>
<tr>
<td></td>
<td>Voltage/Current Magnitude in dBm</td>
<td>“dBm()” on page 3-78</td>
</tr>
</tbody>
</table>
### Table 2-6. RF Design Environment Measurement Expressions

<table>
<thead>
<tr>
<th>Analysis Type</th>
<th>Measurement</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>Deviation From Linear Phase</td>
<td>“dev_lin_phase()” on page 3-84</td>
</tr>
<tr>
<td></td>
<td>Gain Circle</td>
<td>“ga_circle()” on page 3-101</td>
</tr>
<tr>
<td></td>
<td>Gain Ripple</td>
<td>“ripple()” on page 3-175</td>
</tr>
<tr>
<td></td>
<td>Group Delay Ripple</td>
<td>“ripple()” on page 3-175</td>
</tr>
<tr>
<td></td>
<td>Integrated Noise Power</td>
<td>“ns_pwr_int()” on page 3-151</td>
</tr>
<tr>
<td></td>
<td>Load Stability Circle</td>
<td>“l_stab_circle()” on page 3-127</td>
</tr>
<tr>
<td></td>
<td>Load-Mismatch Gain Circle</td>
<td>“gl_circle()” on page 3-105</td>
</tr>
<tr>
<td></td>
<td>Maximum Available and Stable Gain</td>
<td>“max_gain()” on page 3-136</td>
</tr>
<tr>
<td></td>
<td>Noise Figure Circle</td>
<td>“ns_circle()” on page 3-150</td>
</tr>
<tr>
<td></td>
<td>Noise Power in a Reference Bandwidth</td>
<td>“ns_pwr_ref_bw()” on page 3-152</td>
</tr>
<tr>
<td></td>
<td>Optimum Admittance for Noise Match</td>
<td>“yopt()” on page 3-236</td>
</tr>
<tr>
<td></td>
<td>Optimum Impedance for Noise Match</td>
<td>“zopt()” on page 3-243</td>
</tr>
<tr>
<td></td>
<td>Port Input Admittance</td>
<td>“yin()” on page 3-235</td>
</tr>
<tr>
<td></td>
<td>Port Input Impedance</td>
<td>“zin()” on page 3-242</td>
</tr>
<tr>
<td></td>
<td>Power Gain</td>
<td>“pwr_gain()” on page 3-170</td>
</tr>
<tr>
<td></td>
<td>Power Gain Circle</td>
<td>“gp_circle()” on page 3-106</td>
</tr>
<tr>
<td></td>
<td>Rollett Stability Factor</td>
<td>“stab_fact()” on page 3-203</td>
</tr>
<tr>
<td></td>
<td>S-parameter Magnitude</td>
<td>“mag()” on page 3-132</td>
</tr>
<tr>
<td></td>
<td>S-parameter Magnitude in dB</td>
<td>“dB()” on page 3-77</td>
</tr>
<tr>
<td></td>
<td>S-parameter Phase</td>
<td>“phase()” on page 3-159</td>
</tr>
<tr>
<td></td>
<td>S-parameter Smith Chart</td>
<td>Complex Number.</td>
</tr>
</tbody>
</table>
### Table 2-6. RF Design Environment Measurement Expressions

<table>
<thead>
<tr>
<th>Analysis Type</th>
<th>Measurement</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SP</strong> (cont.)</td>
<td>Simultaneous-Match Input-Admittance</td>
<td>“sm_y1()” on page 3-190</td>
</tr>
<tr>
<td></td>
<td>Simultaneous-Match Input- Impedance</td>
<td>“sm_z1()” on page 3-192</td>
</tr>
<tr>
<td></td>
<td>Simultaneous-Match Input-Reflection Coef.</td>
<td>“sm_gamma1()” on page 3-188</td>
</tr>
<tr>
<td></td>
<td>Simultaneous-Match Output-Admittance</td>
<td>“sm_y2()” on page 3-191</td>
</tr>
<tr>
<td></td>
<td>Simultaneous-Match Output- Impedance</td>
<td>“sm_z2()” on page 3-193</td>
</tr>
<tr>
<td></td>
<td>Simultaneous-Match Output-Reflection Coef.</td>
<td>“sm_gamma2()” on page 3-189</td>
</tr>
<tr>
<td></td>
<td>Source Mapping Circles Port1 to Port2</td>
<td>“map1_circle()” on page 3-133</td>
</tr>
<tr>
<td></td>
<td>Source Mapping Circles Port2 to Port1</td>
<td>“map2_circle()” on page 3-134</td>
</tr>
<tr>
<td></td>
<td>Source Stability Circle</td>
<td>“s_stab_circle()” on page 3-177</td>
</tr>
<tr>
<td></td>
<td>Source-Mismatch Gain Circle</td>
<td>“gs_circle()” on page 3-107</td>
</tr>
<tr>
<td></td>
<td>Stability Factor for the Load</td>
<td>“mu()” on page 3-148</td>
</tr>
<tr>
<td></td>
<td>Stability Factor for the Source</td>
<td>“mu_prime()” on page 3-149</td>
</tr>
<tr>
<td></td>
<td>Stability Measure</td>
<td>“stab_meas()” on page 3-204</td>
</tr>
<tr>
<td></td>
<td>VSWR</td>
<td>“vswr()” on page 3-230</td>
</tr>
<tr>
<td></td>
<td>Voltage Gain</td>
<td>“volt_gain()” on page 3-225</td>
</tr>
<tr>
<td><strong>Env</strong></td>
<td>Baseband Signal in Time-Domain</td>
<td>“real()” on page 3-172</td>
</tr>
<tr>
<td></td>
<td>Magnitude of the Carrier in Time-Domain</td>
<td>“mag()” on page 3-132</td>
</tr>
<tr>
<td></td>
<td>Phase of the Carrier in Time-Domain</td>
<td>“phase()” on page 3-159</td>
</tr>
<tr>
<td></td>
<td>Spectrum of the Carrier in dB</td>
<td>“dB()” on page 3-77</td>
</tr>
<tr>
<td></td>
<td>Spectrum of the Carrier in dBm</td>
<td>“dBm()” on page 3-78</td>
</tr>
<tr>
<td>Analysis Type</td>
<td>Measurement</td>
<td>Expression</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Tran</td>
<td>Frequency-Selective Current</td>
<td>&quot;ifc_tran()&quot; on page 3-115</td>
</tr>
<tr>
<td></td>
<td>Frequency-Selective Power</td>
<td>&quot;pfc_tran()&quot; on page 3-158</td>
</tr>
<tr>
<td></td>
<td>Frequency-Selective Voltage</td>
<td>&quot;vfc_tran()&quot; on page 3-224</td>
</tr>
<tr>
<td></td>
<td>Transient Current Spectrum</td>
<td>&quot;ispec_tran()&quot; on page 3-125</td>
</tr>
<tr>
<td></td>
<td>Transient Power Spectrum</td>
<td>&quot;pspec_tran()&quot; on page 3-167</td>
</tr>
<tr>
<td></td>
<td>Transient Voltage Difference</td>
<td>User Defined or Selected.</td>
</tr>
<tr>
<td></td>
<td>Transient Voltage Spectrum</td>
<td>&quot;vspec_tran()&quot; on page 3-229</td>
</tr>
<tr>
<td></td>
<td>Transient Voltage/Current</td>
<td>User Defined or Selected.</td>
</tr>
</tbody>
</table>
Using the MeasEqn Function Reference
Chapter 3: MeasEqn Function Reference

This chapter lists and describes the MeasEqn functions that are available within the Advanced Design System. These functions include mathematical functions such as those for matrix conversion, trigonometry, absolute value, and the like. They also include functions specific to simulation, such as S-parameter functions and budget measurement components.

The tables in this chapter indicate whether or not a function is available as a built-in measurement from a palette in the design window. Although they have been designed to make simulation convenient, the built-in measurement items can also be edited by the user to meet specific requirements.

For more information on how to interpret this material, see Chapter 2, Using the MeasEqn Function Reference.
MeasEqn Function Reference

abcdtoh()

Purpose
Performs ABCD-to-H conversion

Synopsis
abcdtoh(A)

where A is the chain (ABCD) matrix of a 2-port network.

Examples
h=abcdtoh(a)

Used in
Small-signal and large-signal S-parameter simulations.

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

See also
htoabcd(), abcdtoh(), abcdtoz()

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

Purpose
Performs ABCD-to-S conversion

Synopsis
abcdtos(A, zRef)

where A is the chain (ABCD) matrix of a 2-port network and zRef is a reference impedance.

Examples
s=abcdtos(a, 50)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

See also
stoabcd(), abcdtoy(), abcdtoz()

Description
This measurement transforms the chain (ABCD) matrix of a 2-port network to a scattering matrix.
abcdtoy()

**Purpose**
Performs ABCD-to-Y conversion

**Synopsis**
abcdtoy(A)

where A is the chain (ABCD) matrix of a 2-port network.

**Examples**
y = abcdtoy(a)

**Used in**
Small-signal and large-signal S-parameter simulations

**Available as measurement component?**
This equation can be entered by means of a MeasEqn component in S_Param. Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

**Defined in**
$\texttt{HPEESOF_DIR/expressions/ael/network_fun.ael}$

**See also**
ytoabcd(), abcdtoz(), abcdtoh()

**Description**
This measurement transforms the chain (ABCD) matrix of a 2-port network to an admittance matrix.
abcdtoz()

Purpose
Performs ABCD-to-Z conversion

Synopsis
abcdtoz(A)

where A is the chain (ABCD) matrix of a 2-port network.

Examples
z = abcdtoz(a)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

Defined in
$HPEE_SOF_DIR/expressions/ael/network_fun.ael

See also
ztoabcol(), abcdtoy(), abcdtoh()

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

Returns the absolute value of a real number or an integer. In the case of a complex number, the abs function:

- accepts one complex argument.
- returns a positive real number.
- returns the magnitude of its complex argument.

Synopsis

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

where \( x \) is an integer or real number.

Examples

\[ a = \text{abs}(-45) \]
returns 45

Used in

Not applicable

Available as measurement component?

Not applicable

Defined in

Built in

See also

cint(), exp(), float(), int(), log(), log10(), pow(), sgn(), sqrt()
acos()

Purpose
Returns the inverse cosine, or arc cosine, in radians, of a real number or integer

Synopsis
y = acos(x)
where x is an integer or real number, and y ranges from 0 to pi.

Examples
a = acos(-1)
returns 3.142

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
asin(), atan(), atan2()
acpr_vi()

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

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

where
voltage is the single complex voltage spectral component (for example, the fundamental) across a load versus time;
current is the single complex current spectral component into the same load versus time;
mainCh is the two-dimensional vector defining the main channel frequency limits (as an offset from the single voltage and current spectral component);
lowerAdjCh is the two-dimensional vector defining the lower adjacent-channel frequency limits (as an offset from the single voltage and current spectral component);
upperAdjCh is the two-dimensional vector defining the upper adjacent channel frequency limits (as an offset from the single voltage and current spectral component);
winType is an optional window type and must be one of the following: Kaiser, Hamming, Gaussian, 8510, or NoWindow (leaving this field blank is the equivalent of NoWindow); and
winConst is an optional parameter that affects the shape of the applied window. The default window constants are:

- Kaiser: 7.865
- Hamming: 0.54
- Gaussian: 0.75
- 8510: 6 (The 8510 window is the same as a Kaiser window with a window constant of 6.)

Examples

Example equations
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.

Example file
examples/RF_Board/NADC_PA_prj/NADC_PA_ACPRtransmitted.dds

Used in
Adjacent-channel power computations

Available as measurement component?
Equations listed under Description can be entered by means of a MeasEqn component in the Schematic window. There is no explicit ACPR measurement function.

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

See also
acpr_vr(), channel_power_vi(), channel_power_vr(), relative_noise_bw()

Description
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 nonperiodic data.
MeasEqn Function Reference

acpr_vr()

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

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

where

voltage is the single complex voltage spectral component (for example, the fundamental) across a resistive load versus time;

resistance is the load resistance in ohms (default is 50 ohms);

mainCh is the two-dimensional vector defining the main-channel frequency limits (as an offset from the single voltage and current spectral component);

lowerAdjCh is the two-dimensional vector defining the lower adjacent-channel frequency limits (as an offset from the single voltage spectral component);

upperAdjCh is the two-dimensional vector defining the upper adjacent-channel frequency limits (as an offset from the single voltage spectral component);

winType is an optional window type and must be one of the following: Kaiser, Hamming, Gaussian, 8510, or NoWindow (leaving this field blank is the equivalent of NoWindow); and

winConst is an optional parameter that affects the shape of the applied window. The default window constants are:

- Kaiser: 7.865
- Hamming: 0.54
- Gaussian: 0.75
- 8510: 6 (The 8510 window is the same as a Kaiser window with a window constant of 6.)

Examples

Example equations

Vfund = vOut[1]
mainlimits = {-1.2288 MHz/2, 1.2288 MHz/2}
UpChlimits = \{885 \text{ kHz}, 915 \text{ kHz}\}
LoChlimits = \{-915 \text{ kHz}, -885 \text{ kHz}\}
TransACPR = \text{acpr}_\text{vr}(\text{VloadFund}, 50, \text{mainlimits}, \text{LoChlimits}, \text{UpChlimits}, \"Kaiser\")

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

**Note**

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

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

---

**Example file**

examples/Tutorial/ModSources_prj/IS95RevLinkSrc.dds

**Used in**

Adjacent-channel power computations

**Available as measurement component?**

Equations listed under Description can be entered by means of a MeasEqn component in the Schematic window. There is no explicit ACPR measurement function.

**Defined in**

$\text{HPEESOF_DIR/expressions/ael/digital\_wireless\_fun.ael}$

**See also**

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

**Description**
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 nonperiodic data.
add_rf()

Purpose
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)

Synopsis
y = add_rf(T1, T2)
where T1 and T2 are two Timed Complex Envelope signals at two distinct carrier frequencies Fc1 and Fc2.

Examples
y = add_rf(T1, T2)

Used in
Signal processing designs that output Timed Signals using Timed Sinks

Available as measurement component?
Not applicable

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

See also
Not applicable

Description
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.
asin()

Purpose
Returns the inverse sine, or arc sine, in radians, of a real number or integer

Synopsis
\( y = \text{asin}(x) \)
where \( x \) is an integer or real number and \( y \) ranges from \(-\pi/2\) to \(\pi/2\).

Examples
\( a = \text{asin}(-1) \)
returns \(-1.571\)

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
\( \text{acos()} \), \( \text{atan()} \), \( \text{atan2()} \)
atan()

Purpose
Returns the inverse tangent, or arc tangent, in radians, of a real number or integer

Synopsis
y = atan(x)
where x is a real number or integer and y ranges from -pi/2 to pi/2.

Examples
a = atan(-1)
returns -0.785

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
acos(), asin(), atan2()
atan2()

Purpose
Returns the inverse tangent, or arc tangent, of the rectangular coordinates y and x

Synopsis
\[ w = \text{atan2}(y, x) \]
where y and x are integer or real number coordinates, and w ranges from \(-\pi\) to \(\pi\).
\(\text{atan2}(0, 0)\) defaults to \(-\pi/2\).

Examples
a = \text{atan2}(1, 0)
returns 1.571

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
\(\text{acos()}, \text{asin()}, \text{atan}()\)
ber_pi4dqpsk()

Purpose
Returns the symbol probability of error versus signal-to-noise ratio per bit for pi/4 DQPSK modulation

Synopsis
data = ber_pi4dqpsk(vIn, vOut, symRate, noise{, samplingDelay, rotation, tranDelay, pathDelay})

where
vIn and vOut are the complex envelope voltage signals at the input and output nodes, respectively, symRate is the symbol rate (real) of the modulation signal, and noise is the RMS noise vector.

The remaining arguments are optional and will be calculated if not specified by the user: pathDelay is the delay from input to output in seconds, rotation is the carrier phase in radians, and samplingDelay is the clock phase in seconds.

tranDelay is an optional time in seconds that causes this time duration of symbols to be eliminated from the bit error rate calculation. Usually the filters in the simulation have transient responses, and the bit error rate calculation should not start until these transient responses have finished.

Note that ber_pi4dqpsk returns a list of data:

- data[0] = symbol probability of error versus Eb / N0
- data[1] = path delay in seconds
- data[2] = carrier phase in radians
- data[3] = clock phase in seconds
- data[4] = complex(I sample, Q sample)

Examples
y = ber_pi4dqpsk(videal[1], vout[1], 0.5e6, {0.1::-0.01::0.02})

Used in
Circuit Envelope simulation, Data Flow simulation

Available as measurement component?
Not applicable
MeasEqn Function Reference

Defined in
$\text{HPEESOF} \_\text{DIR/expressions/ael/digital\_wireless\_fun.ael}$

See also
ber_qpsk(), constellation()

Description
The arguments $v_{\text{In}}$ and $v_{\text{Out}}$ 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.
ber_qpsk()

Purpose
Returns the symbol probability of error versus signal-to-noise ratio per bit for QPSK modulation

Synopsis
\[
data = \text{ber}_\text{qpsk}(vln, vOut, symRate, noise\{, \text{samplingDelay, rotation, tranDelay, pathDelay}\})
\]

where

- \(vln\) and \(vOut\) are the complex envelope voltage signals at the input and output nodes, respectively, \(symRate\) is the symbol rate (real) of the modulation signal, and \(noise\) is the RMS noise vector.
- The remaining arguments are optional and will be calculated if not specified by the user: \(pathDelay\) is the delay from input to output in seconds, \(rotation\) is the carrier phase in radians, and \(samplingDelay\) is the clock phase in seconds.
- \(tranDelay\) is an optional time in seconds that causes this time duration of symbols to be eliminated from the bit error rate calculation. Usually the filters in the simulation have transient responses, and the bit error rate calculation should not start until these transient responses have finished.

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)

Examples

\[
y = \text{ber}_\text{qpsk}(\text{videal}[1], \text{vout}[1], 1e6, \{0.15::-0.01::0.04\})
\]

Used in
Circuit Envelope simulation, Data Flow simulation

Available as measurement component?
Not applicable
Defined in
$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See also
ber_pi4dqpsk(), constellation()

Description
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.
**bud_freq()**

**Purpose**
Returns the frequency plan of a network

**Synopsis**

```
bud_freq(freqIn, pinNumber, “simName”)  
or  
bud_freq(planNumber, pinNumber)
```

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.

**Note**
To use bud_freq() in AC simulation, the AC controller FreqConversion flag must be set to “yes”.

When using this function with HB data, the planNumber is required. The planNumber is an integer which represents the chosen frequency plan.

For both analyses, the second parameter is the pinNumber, which is used to choose which pins of each network element are referenced. 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.

“simName” is the simulation instance name, such as “AC1” or “HB1”, used to qualify the data when multiple simulations are performed.

**Examples**

```plaintext
x = bud_freq()
Returns frequency plan for AC analysis.

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

\[ x = \text{bud}_\text{freq}(2) \]

For HB, returns a selected frequency plan, 2, with respect to pin 1 of every network element.

**Used in**

AC and harmonic balance simulations

**Available as measurement component?**

BudFreq

**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.

**Defined in**

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

**See also**

Not applicable

**Description**

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
$v_{Out} = \alpha \cdot \text{conj}(v_{In})$ 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.
bud_gain()

Purpose
Returns budget transducer-power gain

Synopsis
bud_gain(vIn, iIn{, Zs, Plan, pinNumber, “simName”})
or
bud_gain(“SourceName”, SrcIndx, Zs, Plan)

where vIn and iIn are the input voltage and the input current (flowing into the input port), respectively. “SourceName” is the component name at the input port, and SrcIndx is the frequency index that corresponds to the source frequency to determine which frequency to use from a multitone source as the reference signal. The input source port impedance Zs is an optional parameter. If not specified, Zs is set to 50.0 ohms. Plan is the number of the selected frequency plan, which is only needed for HB.

Note that for AC simulation, both the SrcIndx and Plan arguments must not be specified; these are for HB only.

pinNumber is used to choose which pins of each network element are referenced. 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.

“simName” is the simulation instance name, such as “AC1” or “HB1”, used to qualify the data when multiple simulations are performed.

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

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

or

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

**Used in**

AC and harmonic balance simulations

**Available as measurement component?**

BudGain

**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.

**Defined in**

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

**See also**

bud_gain_comp()

**Description**

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

**Note**

Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.
**bud_gain_comp()**

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

**Synopsis**
bud_gain_comp(vIn, iIn{, Zs, Plan, freqIndex, pinNumber, “simName”})
or
bud_gain_comp(“SourceName”, SrcIndx{, Zs, Plan, freqIndex, pinNumber, “simName”})

where vIn and iIn are the input voltage and the input current (flowing into the input port), respectively. SrcIndx is the frequency index that corresponds to the source frequency to determine which frequency to use from a multitone source as the reference signal. Zs is an optional input port impedance whose default value is 50.0 ohms. Plan is the number of the selected frequency plan, which is only needed for HB.

If Plan is not selected, the gain compression is calculated at the harmonic frequency selected by freqIndex

pinNumber is used to choose which pins of each network element are referenced. 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.

“simName” is the simulation instance name, such as “AC1” or “HB1”, used to qualify the data when multiple simulations are performed.

**Example**

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

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

**Used in**
Harmonic balance simulation with sweep
Available as measurement component?
BudGainComp

Budget Path measurements
This function does not support the budget path feature.

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

See also
bud_gain()

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

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

Purpose
Returns the budget reflection coefficient

Synopsis
bud_gamma(Zref, Plan, pinNumber, “simName”)

where Zref is the reference impedance, set to 50.0 ohms by default. Plan is the number of the selected frequency plan, which is only needed for HB.

pinNumber is used to choose which pins of each network element are referenced. 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.

“simName” is the simulation instance name, such as “AC1” or “HB1”, used to qualify the data when multiple simulations are performed.

Examples
x = bud_gamma()
returns reflection coefficient at all frequencies.

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

Used in
AC and harmonic balance simulations

Available as measurement component?
BudGamma

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.

Defined in
$HPEESOF_DIR/expressions/ael/budget_fun.ael
See also
bud_vswr()

Description
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.
bud_ip3_deg()

Purpose
Returns the budget third-order intercept point degradation

Synopsis
bud_ip3_deg(vOut, LinearizedElement, fundFreq, imFreq{, zRef})

where vOut is the signal voltage at the output, LinearizedElement is the variable containing the names of the linearized components, fundFreq and imFreq are the harmonic frequency indices for the fundamental and intermodulation frequencies, respectively, and zRef is the reference impedance, set to 50.0 ohms by default.

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

Used in
Harmonic balance simulation with the BudLinearization Controller

Available as measurement component?
BudIP3Deg

Budget Path measurements
This function does not support the budget path feature.

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

See also
ip3_out(), ipn()

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

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

Purpose
Returns the budget noise figure

Synopsis
bud_nf(vIn, iIn, noisevIn, { Zs, BW, pinNumber, “simName”})
or
bud_nf(“SourceName”)

where vIn, iIn, and noisevIn are the signal voltage and current (flowing into the input port) and the noise voltage at the input port, respectively. The input port impedance and the bandwidth are optional parameters. If not specified, Zs and BW are set to 50.0 ohms and 1 Hz, respectively. BW must be set as the value of Bandwidth used on the noise page of the AC controller.

pinNumber is used to choose which pins of each network element are referenced. 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.

“simName” is the simulation instance name, such as “AC1” or “HB1”, used to qualify the data when multiple simulations are performed.

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

Used in
AC simulation

Available as measurement component?
BudNF.

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

See also
bud_nf_deg(), bud_tn()

Description
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.
bud_nf_deg()

**Purpose**

Returns budget noise figure degradation

**Synopsis**

bud_nf_deg(vIn, iIn, vOut, iOut, vOut.NC.vnc, vOut.NC.name{, Zs, BW})

or

bud_nf_deg("PORT1", "Term1", "vOut")

where vIn and iIn are the voltage and current at the input port, and vOut and iOut are the voltage and current at the output port. vOut.NC.vnc and vOut.NC.name are the noise contributions and the corresponding component names at the output port, respectively. The input port impedance, bandwidth, and temperature are optional parameters.

If not specified, Zs and BW are set to 50.0 ohms and 1 Hz, respectively. BW must be set as the value of Bandwidth used on the noise page of the AC controller.

**Example**

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

x = bud_nf_deg("PORT1", "Term1", "vOut")
```

**Used in**

AC simulation

**Available as measurement component?**

BudNFDeg

**Budget Path measurements**

This function does not support the budget path feature.

**Defined in**

$HPEESOF_DIR/expressions/ael/budget_fun.ael

**See also**

bud_nf(), bud_tn()

**Description**
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.
MeasEqn Function Reference

**bud_noise_pwr()**

**Purpose**
Returns the budget noise power

**Synopsis**
bud_noise_pwr(Zref, Plan, pinNumber, “simName”)

where Zref is the reference impedance and Plan is the number of the selected frequency plan, which is only needed for HB.

pinNumber is used to choose which pins of each network element are referenced. 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.

“simName” is the simulation instance name, such as “AC1” or “HB1”, used to qualify the data when multiple simulations are performed.

**Example**
x = bud_noise_pwr()
returns the noise power at all frequencies

y = bud_noise_pwr(75, 1)
returns the noise power at reference frequencies in plan 1

**Used in**
AC and harmonic balance simulations

**Available as measurement component?**
BudNoisePwr

**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.

**Defined in**
$HPEESOF_DIR/expressions/ael/budget_fun.ael
See also

bud_pwr()

Description

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

---

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

Purpose
Returns the budget signal power in dBm

Synopsis
bud_pwr(Plan, pinNumber, “simName”)

where Plan is the number of the selected frequency plan, which is only needed for HB.

pinNumber is used to choose which pins of each network element are referenced. 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.

“simName” is the simulation instance name, such as “AC1” or “HB1”, used to qualify the data when multiple simulations are performed.

Example
x = bud_pwr() returns the signal power at all frequencies when used in AC or HB simulations

y = bud_pwr(50, 1) returns the signal power at reference frequencies in plan 1 when used for HB simulations

Used in
AC and harmonic balance simulations

Available as measurement component?
Not applicable

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.

Defined in
$HPEESOF_DIR/expressions/ael/budget_fun.ael
See also

bud_noise_pwr()

Description

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.
bud_pwr_inc()  

Purpose  
Returns the budget incident power  

Synopsis  
bud_pwr_inc({Zref, Plan, pinNumber, "simName"})

where Zref is the reference impedance, set to 50.0 ohms by default. Plan is the number of the selected frequency plan, which is only needed for HB.

pinNumber is used to choose which pins of each network element are referenced. 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.

"simName" is the simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.

Example  
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  

Used in  
AC and harmonic balance simulations  

Available as measurement component?  
BudPwrInc  

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.

Defined in  
$HPEESOF_DIR/expressions/ael/budget_fun.ael
See also

bud_pwr_refl()

Description

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.
bud_pwr_refl()

Purpose
Returns the budget reflected power

Synopsis
bud_pwr_refl(Zref, Plan, pinNumber, “simName”)

where Zref is the reference impedance, set to 50.0 ohms by default. Plan is the number of the selected frequency plan, which is only needed for HB.

pinNumber is used to choose which pins of each network element are referenced. 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.

“simName” is the simulation instance name, such as “AC1” or “HB1”, used to qualify the data when multiple simulations are performed.

Example
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

Used in
AC and Harmonic balance simulations

Available as measurement component?
BudPwrRefl

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.

Defined in
%HPEESOF_DIR/expressions/ael/budget_fun.ael
See also

bud_pwr_inc()

Description

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.
bud_snr()

Purpose
Returns the budget signal-to-noise-power ratio

Synopsis
bud_snr(Plan, pinNumber, "simName")

where Plan is the number of the selected frequency plan, which is only needed for HB.

pinNumber is used to choose which pins of each network element are referenced. 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.

"simName" is the simulation instance name, such as “AC1” or “HB1”, used to qualify the data when multiple simulations are performed.

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

or

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

Used in
AC and harmonic balance simulations

Available as measurement component?
BudSNR

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.

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

See also

Not applicable

Description

This is the SNR (in dB) at the terminal(s) of each component, looking into that component. Note that the fundamental frequency at different pins can in general be different, and therefore values are given for all frequencies unless a Plan is referenced. The noise analysis control parameter in the AC and Harmonic Balance Simulation components must be selected. For the AC Simulation component select: “Calculate Noise” and “Include port noise.” For the source, the parameter “Noise” should be set to yes. In Harmonic Balance select the “Nonlinear noise” option.

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

Purpose
Returns the budget equivalent output-noise temperature

Synopsis
bud_tn(vIn, iIn, noisevIn{, Zs, BW, pinNumber, “simName”})

or
bud_tn(“SourceName”)

where vIn, iIn, and noisevIn are the signal voltage and current (flowing into the input port) and the noise voltage at the input port, respectively. The input port impedance, the bandwidth, and the temperature are optional parameters.

If not specified, Zs and BW are set to 50.0 ohms and 1 Hz, respectively. If the values of BW or Temp used in the simulation are different from their default values, be sure to use their correct values in the budget function. BW must be set as the value of Bandwidth used on the noise page of the AC controller.

pinNumber is used to choose which pins of each network element are referenced. 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.

“simName” is the simulation instance name, such as “AC1” or “HB1”, used to qualify the data when multiple simulations are performed.

Example
x = bud_tn(PORT1.t1.v, PORT1.t1.i, PORT1.t1.v.noise)

Used in
AC simulation

Available as measurement component?
BudTN

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.

**Defined in**

$HPEESOF_DIR/expressions/ael/budget_fun.ael

**See also**

`bud_nf()`, `bud_nf_deg()`

**Description**

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

**Note** Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.
bud\_vswr()

**Purpose**
Returns the budget voltage-standing-wave ratio

**Synopsis**
\[
bud\_vswr(\{Zref, Plan, pinNumber, \"simName\"\})
\]

where \(Zref\) is the reference impedance, set to 50.0 ohms by default. \(Plan\) is the number of the selected frequency plan, which is only needed for HB.

\(pinNumber\) is used to choose which pins of each network element are referenced. 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.

\("simName"\) is the simulation instance name, such as “AC1” or “HB1”, used to qualify the data when multiple simulations are performed.

**Example**
\[
x = bud\_vswr()
\]
returns the vswr at all frequencies

\[
y = bud\_vswr(75, 1)
\]
returns the vswr at reference frequencies in plan 1

**Used in**
AC and harmonic balance simulations

**Available as measurement component?**
BudVSWR

**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.

**Defined in**
$HPEESOF\_DIR\/expressions/ael/budget\_fun.ael
See also

bud_gamma()

Description

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.
carr_to_im()

Purpose
Returns the ratio of carrier signal power to IMD power

Synopsis
carr_to_im(vOut, fundFreq, imFreq)

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

Used in
Harmonic balance simulation

Available as measurement component?
CarrToIM

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

See also
ip3_out()

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

Purpose
Returns the cumulative distribution function

Synopsis
```
cdf(data, numBins, minBin, maxBin)
```

where \( x \) is the signal, \( \text{numBins} \) is the number of subintervals or bins used to measure CDF, and \( \text{minBin} \) and \( \text{maxBin} \) are the beginning and end, respectively, of the evaluation of the CDF.

Example
```
cdf(data)
cdf(data, 20)
```

Used in
Not applicable

Available as measurement component?
This function can only be entered by means of a \texttt{Eqn} component in the Data Display window. There is no measurement component in schematic window.

Defined in
\$HPEESOF\_DIR/expressions/ael/statistical\_fun.ael

See also
\texttt{histogram()}, \texttt{pdf()}, \texttt{yield\_sens()}

Description
This function measures the cumulative distribution function of a signal. The default values for \( \text{minBin} \) and \( \text{maxBin} \) are the minimum and the maximum values of the data, and \( \text{numBins} \) is set to \( \log(\text{numOfPts})/\log(2.0) \) by default.
MeasEqn Function Reference

cdrange()

Purpose
Returns compression dynamic range

Synopsis
cdrange(nf, inpwr_lin, outpwr_lin, outpwr)

where nf is noise figure at the output port, inpwr_lin and outpwr_lin are input and the output power, respectively, in the linear region, and outpwr is output power at 1 dB compression.

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

Used in
XDB simulation

Available as measurement component?
CDRange

Defined in
$HPEESOF_DIR/expressions/ael_rf_system_fun.ael

See also
sfdr()

Description
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.
channel_power_vi()

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

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

where
voltage is the single complex voltage spectral component (for example, the
fundamental) across a load versus time;
current is the single complex current spectral component into the same load
versus time;
mainCh is the two-dimensional vector defining channel frequency limits (as an
offset from the single voltage and current spectral component (note that these
frequency limits do not have to be centered on the voltage and current spectral
component frequency);
winType is an optional window type and must be one of the following: Kaiser,
Hamming, Gaussian, 8510, or NoWindow (leaving this field blank is the
equivalent of NoWindow); and
winConst is an optional parameter that affects the shape of the applied window.
The default window constants are:
  Kaiser: 7.865
  Hamming: 0.54
  Gaussian: 0.75
  8510: 6 (The 8510 window is the same as a Kaiser window with a window
constant of 6.)

Examples
Example equations
VloadFund = vload[1]
IloadFund = iload.i[1]
mainlimits = {-16.4 kHz, 16.4 kHz}
MeasEqn Function Reference

\[
\text{Main\_Channel\_Power} = \text{channel\_power\_vi}(V\text{loadFund}, I\text{loadFund}, \text{mainlimits}, \text{Kaiser})
\]

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

Use the equation

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

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

**Example file**

examples/RF\_Board/NADC\_PA\_prj/NADC\_PA\_ACPRtransmitted.dds

**Used in**

Channel power computations

**Available as measurement component?**

Equations listed under **Description** can be entered by means of a MeasEqn component in the Schematic window. There is no explicit channel-power measurement function for use with Circuit Envelope data.

**Defined in**

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

**See also**

acpr\_vi(), acpr\_vr(), channel\_power\_vr()

**Description**

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 nonperiodic data.
channel_power_vr()

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

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

where
- voltage is the single complex voltage spectral component (for example, the fundamental) across a resistive load versus time;
- resistance is the load resistance in ohms (default is 50 ohms);
- mainCh is the two-dimensional vector defining channel frequency limits (as an offset from the single voltage and current spectral component (note that these frequency limits do not have to be centered on the voltage and current spectral component frequency);
- winType is an optional window type and must be one of the following: Kaiser, Hamming, Gaussian, 8510, or NoWindow (leaving this field blank is the equivalent of NoWindow); and
- winConst is an optional parameter that affects the shape of the applied window.

The default window constants are:
- Kaiser: 7.865
- Hamming: 0.54
- Gaussian: 0.75
- 8510: 6 (The 8510 window is the same as a Kaiser window with a window constant of 6.)

Example

Example equations
Vmain_fund = Vmain[1]
mainlimits = {-16.4 kHz, 16.4 kHz}
Main_Channel_Power = channel_power_vr(Vmain_fund, 50, mainlimits, Kaiser)
where $V_{\text{main}}$ 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 equation

$$\text{Main Channel Power dBm} = 10 \times \log(\text{Main Channel Power}) + 30$$

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

**Example file**

examples/RF_Board/NADC_PA_prj/NADC_PA_ACPRreceived.dds

**Used in**

Channel power computations

**Available as measurement component?**

Equations listed under Description can be entered by means of a MeasEqn component in the Schematic window. There is no explicit channel-power measurement function for use with Circuit Envelope data.

**Defined in**

$\text{HPEESOF DIR/expressions/ael/digital_wireless_fun.ael}$

**See also**

acpr_vi(), acpr_vr(), channel_power_vi()

**Description**

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 nonperiodic data.
chop()

Purpose
Replace numbers in x with magnitude less than dx with 0

Synopsis
y=chop(x {, dx})
then y=x if mag(x) >= mag(dx)
and y=0 if mag(x) < mag(dx)
dx is optional, default is 1e-10.
Actually this function is more complicated; it acts independently on the real and complex components of x, comparing each to mag(dx)

Example
chop(1)
1
chop(1e-12)
0
chop(1+1e-12i)
1+0i

Used in
Not applicable

Available as measurement component?
Not applicable

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

See also
None
**chr()**

**Purpose**
Returns the character representation of an integer

**Synopsis**
y = chr(x)

where x is a valid ASCII string representing a character.

**Examples**
a = chr(64)
"@"

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
Not applicable
circle()

Purpose
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.

Synopsis
a = circle(x, y, z)

where x is the center coordinate (can be a complex number), y is the radius, and z is the number of points

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

Used in
Data Display

Available as measurement component?
Not applicable

Defined in
AEL

See also
Not applicable
cint()

Purpose
Given a noninteger real number, returns a rounded integer

Synopsis
y = cint(x)
where x is a real number to be rounded to an integer.

Note 0.5 rounds up, -0.5 rounds down (up in magnitude).

Examples
a = cint(45.6)
46

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
abs(), exp(), float(), int(), log(), log10(), pow(), sgn(), sqrt()
**cmplx()**

**Purpose**
Given two real numbers representing the real and imaginary components of a complex number, returns a complex number.

**Note** Use the real and imag functions to retrieve the real and imaginary components, respectively. The basic math functions operate on complex numbers.

**Synopsis**
y = cmplx(x, y)
where x is the real component and y is the imaginary component.

**Examples**
a = cmplx(2, -1)
2 - 1j

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
imag(), real()
conj()
Purpose
Returns the conjugate of a complex number
Synopsis
\[ y = \text{conj}(x) \]
where \( x \) is a complex number.
Examples
\[ a = \text{conj}(3-4*\text{j}) \]
3.000 + j4.000
or 5.000 / 53.130 in magnitude / degrees
Used in
Not applicable
Available as measurement component?
Not applicable
Defined in
Built in
See also
mag()
const_evm()

Purpose
Takes the results of a Circuit Envelope simulation and generates 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

Synopsis

\[
data = \text{const_evm}(\text{vfund\\_ideal}, \text{vfund\\_dist}, \text{symbol\\_rate}, \text{sampling\\_delay}, \text{rotation}, \text{transient\\_duration}, \text{path\\_delay})
\]

where

vfund\\_ideal is a 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().

vfund\\_dist is a 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().

symbol\\_rate is the symbol rate of the modulation signal.

sampling\\_delay (if nonzero) throws away the first delay = N seconds of data from
the constellation and trajectory plots. It is also used to interpolate between
simulation time points, which is necessary if the optimal symbol-sampling instant
is not exactly at a simulation time point. Usually this parameter must be nonzero
to generate a constellation diagram with the smallest grouping of sample points.

rotation is a user-selectable parameter that rotates the constellations by that
many radians. It does not need to be entered, and it will not affect the
error-vector-magnitude calculation, because both the ideal and distorted
constellations will be rotated by the same amount.

transient\\_duration is an optional time in seconds that causes this time duration of
symbols to be eliminated from the error-vector-magnitude calculation. Usually the
filters in the simulation have transient responses, and the error-vector-magnitude
calculation should not start until these transient responses have finished.

path\\_delay is an optional time in seconds of the sum of all delays in the signal
path. If the delay is 0, this parameter may be omitted. If it is non-zero, enter the
delay value. This can be calculated by using the function delay\\_path().

Note that const\\_evm returns a list of data. So in the above example,
MeasEqn Function Reference

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

Please refer to the example file listed below to see how these data are plotted.

**Example**

**Example equations**

\[
\text{rotation} = -0.21
\]

\[
\text{sampling
delay} = 1/\text{sym	rate}[0, 0] - 0.5 \ast \text{tstep}[0, 0]
\]

\[
\text{vfund}_\text{ideal} = \text{vOut}_\text{ideal}[1]
\]

\[
\text{vfund}_\text{dist} = \text{vOut}_\text{dist}[1]
\]

\[
\text{symbol	rate} = \text{sym	rate}[0, 0]
\]

\[
data = \text{const	evm} (\text{vfund}_\text{ideal}, \text{vfund}_\text{dist}, \text{symbol	rate}, \text{sampling
delay}, \text{rotation},
1.5\text{ms}, \text{path	delay})
\]

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 it so that

\[
\text{Other} = \text{OutVar} = \text{sym	rate} \text{OutVar} = \text{tstep}
\]

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

**Example files**

- examples/RF_Board/NADC_PA_prj/NADC_PA_Test.dsn and ConstEVM.dds
- examples/Tutorial/Env_BER_prj/timing_doc.dds

**Used in**

Constellation and trajectory diagram generation and error-vector-magnitude calculation

**Available as measurement component?**
Equations listed under Description can be entered by means of a MeasEqn component in the Schematic window. There is no explicit constellation or error-vector-magnitude measurement function.

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

See also
c constellation(), delay_path(), sample_delay_pi4dqpsk(), sample_delay_qpsk()

Description
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.
**constellation()**

**Purpose**
Generates the constellation diagram from Circuit Envelope, Transient, or Ptolemy simulation I and Q data.

**Synopsis**
Const = constellation(i_data, q_data, symbol_rate, delay)

where
i_data is the in-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) (this could be a baseband signal instead, but in either case it must be real-valued versus time);
q_data is the quadrature-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) (this could be a baseband signal instead, but in either case it must be real valued versus time);
symbol_rate is the symbol rate of the modulation signal; and delay (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.

**Example**
**Example equations**
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 ADS notation of "vOut[1]".

---

**Example files**

examples/RF_Board/NADC_PA_prj/ConstEVMslow.dds
examples/Tutorial/ModSources_prj/QAM_16_ConstTraj.dds

**Used in**

Constellation diagram generation

**Available as measurement component?**

Equations listed under Description can be entered by means of a MeasEqn component in the Schematic window. There is no explicit constellation measurement function.

**Defined in**

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

**See also**

const_evm()
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.
contour()

Purpose
Generates contour levels on surface data

Synopsis
\[ y = \text{contour}(\text{data} \{ \text{contour\_levels} \}) \]

where data is the data to be contoured, which must be at least two-dimensional
real number or integer or implicit, and contour\_levels is an optional
one-dimensional quantity specifying the levels of the contours, which is normally
specified by the sweep generator “[ ],” but can also be specified as a vector. If not
provided, contour\_levels defaults to six levels equally spaced between the
maximum and the minimum of the data.

Examples
a = \text{contour}(\text{dB}(S_{11}), [1::3::10])
or
a = \text{contour}(\text{dB}(S_{11}), \{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.

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
contour\_polar()

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

**Purpose**
Generates contour levels on polar or Smith chart surface data

**Synopsis**

```math
y = contour_polar(data \{ contour_levels\})
```

where `data` is the polar or Smith chart data to be contoured, (and therefore is surface data), and `contour_levels` is an optional one-dimensional quantity specifying the levels of the contours, which is normally specified by the sweep generator "[]," but can also be specified as a vector. If not provided, `contour_levels` defaults to six levels equally spaced between the maximum and the minimum of the data.

**Examples**

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

or

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

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

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**

```
$HPEESOF_DIR/expressions/ael/display_fun.ael
```

**See also**

`contour()`
cos()

Purpose
Returns the cosine of a real number or integer

Synopsis
\[ y = \cos(x) \]
where \( x \) is the real number or integer, in radians.

Examples
\[ a = \cos(pi/3) \]
0.500

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
\( \sin(), \tan() \)
cosh()

Purpose
hyperbolic cosine

Synopsis
cosh()

Example
cosh(0)
1
cosh(1)
1.543

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
sinh(), tanh()
cross()

Purpose
Computes the zero crossings of a signal and the interval between successive zero crossings. The independent axis returns the time when the crossing occurred. The dependent axis returns the time interval since the last crossing.

Synopsis
cross(signal, direction)
If direction = +1, compute positive going zero crossings.
If direction = -1, compute negative going zero crossings.
If direction = 0, compute all zero crossings (default).

Example
period=cross(vosc-2.0, 1)
This computes the period of each cycle of the vosc signal. The period is measured from each positive-going transition through 2.0V.

Used In
Not applicable

Available in measurement component?
Not applicable

Defined In
Built in

See Also
None
cross_corr()

Purpose
Returns the cross-correlation

Synopsis
cross_corr(v1, v2)
    where v1 and v2 are 1-dimensional data

Example
v1 = qpsk..videal[1]
v2 = qpsk..vout[1]
x_corr_v1v2 = cross_corr(v1, v2)
auto_corr_v1 = cross_corr(v1, v1)

Used in
Not applicable

Available as measurement component?
Not applicable

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

See also
None
cum_prod()

Purpose
Returns the cumulative product

Synopsis
cum_prod(x)
The function takes a single argument, so enclose a sequence of numbers in brackets "[x, y, ...]"

Example
cum_prod(1)
1
cum_prod([1, 2, 3])
[1, 3, 6]
cum_prod([i, i])
[i, i^2]

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
cum_sum(), max(), mean(), min(), prod(), sum()
cum_sum()

Purpose
Returns the cumulative sum

Synopsis
cum_sum(x)
The function takes a single argument, so enclose a sequence of numbers in brackets “[x, y, ...]”

Example
cum_sum([1, 2, 3])
[1,3,6]
cum_sum([i, i]
[i,2i]

Used in
Not Applicable

Available as measurement component?
Not Applicable

Defined in
Built in

See also
cum_prod(), max(), mean(), min(), prod(), sum()
**dB()**

**Purpose**

Returns the decibel measure of a voltage ratio

**Synopsis**

\[ \text{dB}(r \{ z1, z2 \}) = 20 \log(\text{mag}(r)) - 10 \log(\frac{z\text{Outfactor}}{z\text{Infactor}}) \]

where \( r \) is a voltage ratio (\( \frac{v\text{Out}}{v\text{In}} \)), \( z1 \) is the source impedance (default is 50), \( z\text{Outfactor} = \text{mag}(z2)^2 / \text{real}(z2) \), \( z2 \) is the load impedance (default is 50), and \( z\text{Infactor} = \text{mag}(z1)^2 / \text{real}(z1) \).

**Examples**

- \( \text{dB}(100) \)
  
  40

- \( \text{dB}(8-6^j) \)
  
  20

**Used in**

Not applicable

**Available as measurement component?**

Not applicable

**Defined in**

Built in

**See also**

\( \text{dBm}(), \text{pae}() \)
dBm()

Purpose

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

Synopsis

\[ \text{dBm}(v, \ z) = 20 \log(\text{mag}(v)) - 10 \log(\text{real}(\text{zOutfactor}/50)) + 10 \]

where \( v \) is a voltage (the peak voltage), \( z \) is an impedance (default is 50), and \( \text{zOutfactor} = \frac{\text{mag}(z)^2}{\text{real}(z)} \).

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:

\[ \text{noise\_power} = \text{dBm}(\text{vout\_noise} \cdot \sqrt{2}); \]

For more information, refer to “Understanding the dBm() function” on page 3-79.

Examples

- dBm(100)
  - 50
- dBm(8-6*j)
  - 30

Used in

Not applicable

Available as measurement component?

Not applicable

Defined in

Built in

See also

dB(), pae()
Understanding the dBm() function

Given a power Po in Watts, the power in dB is:

\[ Po_{dB} = 10 \times \log(\text{mag}(\frac{Po}{1\ W})) \]

while the power in dBm is:

\[ Po_{dBm} = 10 \times \log(\text{mag}(\frac{Po}{1\ mW})) + 30 \]
\[ = Po_{dB} + 30 \]

Given a voltage Vo in Volts, the voltage in dB is:

\[ V_{dB} = 20 \times \log(\text{mag}(\frac{Vo}{1\ V})) \]

This is the dB() function in ADS - voltage in dB relative to 1 V.

Given a real impedance Zo, the power-voltage relation is:

\[ Po = \frac{(Vo)^2}{2 \times Zo} \]

Using the above, Po in dBm is then:

\[ Po_{dBm} = 10 \times \log(\text{mag}(\frac{Po}{1\ W})) + 30 \]
\[ = 10 \times \log(\text{mag}(\frac{(Vo(1 V))^2}{2 \times Zo(1 Ohm)})) + 30 \]
\[ = 10 \times \log(\text{mag}(\frac{(Vo(1 V))^2}{2 \times Zo(1 Ohm)})) - 10 \times \log(\text{mag}(2 \times Zo(1 Ohm))) + 30 \]
\[ = 20 \times \log(\text{mag}(Vo(1 V))) - 10 \times \log(\text{mag}(Zo(50 Ohm))) + 10 \]

This is the dBm() function in ADS - voltage in dBm in a Zo environment.
dBmtoW()

Purpose
Converts dBm to watts.

Synopsis
\[
\text{dBmtoW}(P) = (1\text{mW}) \times 10^{P/10}
\]
where \( P \) is the power expressed in dBm.

Examples
\[
\begin{align*}
d\text{BmtoW}(0) & = 0.001 \text{ W} \\
d\text{BmtoW}(-10) & = 1.000 \times 10^{-4} \text{ W}
\end{align*}
\]

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
dBm()
dc_to_rf()

Purpose
Returns DC-to-RF efficiency

Synopsis
dc_to_rf(vPlusRF, vMinusRF, vPlusDC, vMinusDC, currentRF, currentDC, freq)

where vPlusRF and vMinusRF are RF voltages at the negative terminals,
vPlusDC and vMinusDC are DC voltages at the negative terminals, currentRF
and currentDC are the RF and DC currents for power calculation, and freq is
harmonic index of the RF frequency at the output port.

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

Used in
Harmonic balance simulation

Available as measurement component?
DCtoRF

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

See also
None

Description
This measurement computes the DC-to-RF efficiency of any part of the network.
deg()

Purpose
Converts radians to degrees

Synopsis
deg(x)

Example
deg(1.5708)
90
deg(pi)
180

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
rad()
**delay_path()**

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

**Synopsis**

delay_path(vin, vout)

where vin is the complex envelope (I + j * Q) at the input node and vout is I + j * Q at the output node.

**Example**

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

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

or

x = delay_path(T1, T2)

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

**Used in**
Circuit envelope simulation, Ptolemy simulation.

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
ber_pi4dqpsk(), ber_qpsk(), const_evm(), cross_corr()

**Description**
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.
\textbf{dev\_lin\_phase()}

\textbf{Purpose}
Returns deviation (in degrees) from linear phase.

\textbf{Synopsis}
\texttt{dev\_lin\_phase(voltGain)}
where \texttt{voltGain} is a function of frequency.

\textbf{Example}
a = dev\_lin\_phase(S21)

\textbf{Used in}
Small-signal and large-signal S-parameter simulations

\textbf{Available as measurement component?}
DevLinPhase

\textbf{Defined in}
$\texttt{HPEESOF\_DIR/expressions/ael/rf\_system\_fun.ael}$

\textbf{See also}
diff(), phasedeg(), phaserad(), pwr\_gain(), ripple(), unwrap(), volt\_gain()

\textbf{Description}
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.
diff()

Purpose
Returns the numerical difference

Synopsis
y = diff(data)
returns the numerical difference against the inner independent variable associated
with the data.

Examples
group_delay = -diff(unwrap(phaserad(S21),pi))/ (2*pi)

Used in
Not applicable

Available as measurement component?
Not applicable

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

See also
dev_lin_phase(), integrate(), phasedeg(), phaserad(), ripple(), unwrap()

Description
Calculates a simple numerical difference against the inner independent variable
associated with the data. Can be used to calculate group delay.
erf()

Purpose
Returns the error function

Synopsis
y = erf(x)
where x is real.

Examples
a = erf(0.1)
0.112
a = erf(0.2)
0.223

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
erfc()

Description
Calculates the error function, the area under the Gaussian curve exp(−x**2).
erfc()

Purpose
Returns the complementary error function

Synopsis
\[ y = \text{erfc}(x) \]
where \( x \) is real.

Examples
a = -erfc(0.1)
0.888
a = -erfc(0.2)
0.777

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
erf()

Description
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.
exp()

Purpose
Given an integer or real number as an exponent, returns $e^{\text{exponent}}$.

Synopsis
$y = \exp(x)$
where $x$ is the exponent of $e$

Examples
$a = \exp(1)$
2.71828

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
abs(), cint(), float(), int(), log(), log10(), pow(), sgn(), sqrt()
eye()

Purpose
Creates data for an eye diagram plot

Synopsis
eye(data, symbolRate{, Cycles{, Delay}})
data is either numeric data or a time domain waveform typically from the I or Q data
cchannel. symbolRate is the 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. Cycles is optional and is the number of cycles to repeat, default is 1. Delay
is an optional sampling delay, default is 0.

Example
eye(I_data, symbol_rate)

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
cconstellation()

description
Refer also to the eye tutorial in analysis.dds in examples/Tutorial/express_meas_prj.
The cycle parameter is used to display more than one cycle of the eye. The delay
parameter is used to adjust the position of the eye opening. Note that delay is not
used to remove a transient in the eye diagram. To remove an initial transient, you
must use explicit indexing on the original data. Following this, you may want to use a
delay to realign the eye opening.
fft()

Purpose
Performs the discrete Fourier transform

Synopsis
y = fft(x, length)

Example
fft([1, 1, 1, 1])
[4+0i, 0+0i]
fft([1, 0, 0, 0])
[1+0i, 1+0i]
fft(1, 4)
[1+0i, 1+0i]

Used in
Not applicable

Available as measurement component?
Not Applicable

Defined in
Built in

See also
fs(), ts()

Description
fft(x) is the discrete Fourier transform of x computed with the fast Fourier transform algorithm. fft() 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.
**find_index()**

**Purpose**
Finds the closest index for a given search value

**Synopsis**
```python
index = find_index(data_sweep, search_value)
```
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.

**Examples**
Given S-parameter data swept as a function of frequency, find the value of S11 at 1 GHz:
```python
index = find_index(freq, 1GHz)
a = S11[index]
```

**Used in**
Use with all simulation data

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
mix()
float()

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

Note: To convert a real to an integer, use int.

Synopsis
y = float(x)
    where x is the integer to convert.

Examples
a = float(10)

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
abs(), cint(), float(), int(), log10(), pow(), sgn(), sqrt()
fs()

Purpose
Performs a time-to-frequency transform

Synopsis
fs(x, fstart, fstop, numfreqs, dim, windowType, windowConst, tstart, tstop, interpOrder, transformMethod)
See detailed Description below.

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

y=fs(vOut)
returns the spectrum (0, 0.2GHz, ... , 25.6GHz), evaluated from 0 to 5 ns.

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

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

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

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

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

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
fft(), fspot()

Description

fs(x) returns the frequency spectrum of the vector x by using a chirp-z transform. The values returned are peak, complex values.

x will typically be data 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 nonuniformly 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.

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

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.

The data to be transformed may be windowed. The window is specified by windowType, with an optional window constant windowConst. 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

The default time-to-frequency transform is done by means of a chirp-z transform. This may be changed by using transformMethod:

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

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, \( y = \text{fs}([0], 0, 25e3, 251) \). 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 nor Ptolemy simulation because voltage data from Transient and baseband data from Ptolemy are real.
fspot()

Purpose
Performs a single-frequency time-to-frequency transform

Synopsis
fspot(x, fund, harm, windowType, windowConst, interpOrder, tstart)
See detailed Description below.

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:

fspot(vOut)
returns the 200-MHz component, integrated from 0 to 5 ns.

fspot(vOut, , 5)
returns the 1-GHz component, integrated from 0 to 5 ns.

fspot(vOut, 1GHz, 1)
returns the 1-GHz component, integrated from 4 to 5 ns.

fspot(vOut, 0.5GHz, 2, , , 2.5ns)
returns the 1-GHz component, integrated from 2.5 to 4.5 ns.

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.

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.

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
fft(), fs()

Description

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.

fund must be greater than zero. It is used to specify the period 1/fund for the Fourier transform. fund defaults to a period that matches the length of the independent axis of x.

harm may be any positive number. harm defaults to 1. Specifying harm=0 will compute the dc component of x.

The data to be transformed may be windowed. The window is specified by windowType, with an optional window constant windowConst. 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

windowType can be specified either by the number or by the name.
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 nonuniformly sampled data from a transient simulation.

Transient simulation uses a variable timestep and variable order algorithm. The user sets an upper limit on the allowed timestep, but the simulator will control the timestep so the local truncation error of the integration is controlled. If the Gear integration algorithm is used, the order can also be changed during simulation. fspot can use all of this information when performing the Fourier transform. The time data are not resampled; the Fourier integration is performed from timestep to timestep of the original data.

When the order varies, the Fourier integration will adjust the order of the polynomial it uses to compute the shape of the data between timepoints.

This variable order integration depends on the presence of a special dependent variable, tranorder, which is output by the transient simulator. If this variable is not present, or if the user wishes to override the interpolation scheme, then interpOrder may be set to a nonzero value:

1 = use only linear interpolation
2 = use quadratic interpolation
3 = use cubic polynomial interpolation

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.
fun_2d_outer()

Purpose
Applies a function to the outer dimension of two-dimensional data.

Synopsis
fun_2d_outer(data, fun)

where data must be two-dimensional data, and fun is some function (usually mean, max, or min) that will be applied to the outer dimension of the data.

Example
fun_2d_outer(data, min)

Used in
max_outer, mean_outer, min_outer functions.

Available as measurement component?
No, but the function can be used on a schematic page, in a measurement equation.

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

See also
max_outer(), mean_outer(), min_outer()

Description
Functions such as mean, max, and min operate on the inner dimension of two-dimensional data. The function fun_2d_outer enables these functions to be applied to the outer dimension. As an example, assume that a Monte Carlo simulation of an amplifier was run, with 151 random sets of parameter values, and that for each set the S-parameters were simulated over 26 different frequency points. S21 becomes a [151 Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being frequency, and the outer dimension being Monte Carlo index. Now, assume that it is desired to know the mean value of the S-parameters at each frequency. Inserting an equation mean(S21) computes the mean value of S21 at each Monte Carlo iteration. If S21 is simulated from 1 to 26 GHz, it computes the mean value over this frequency range, which usually is not very useful. The function fun_2d_outer allows the mean to be computed over each element in the outer dimension.
ga_circle()

Purpose
Generates an available-gain circle

Synopsis
\[ \text{ga_circle}(S \{ \text{gain}, \text{numOfPts} \}) \]

where \( S \) is the scattering matrix of a 2-port network, \( \text{gain} \) is the specified gain in dB, and \( \text{numOfPts} \) is the desired number of points per circle. The default value for \( \text{gain} \) is \( \min(\max_{\text{gain}}(S)) - \{1, 2, 3\} \), and the default value for \( \text{numOfPts} \) is 51.

Example
\[
\begin{align*}
\text{circleData} &= \text{ga_circle}(S, 2, 51) \\
\text{circleData} &= \text{ga_circle}(S, \{2, 3, 4\}, 51) \\
&\text{return the points on the circle(s).}
\end{align*}
\]

Used in
Small-signal S-parameter simulations

Available as measurement component?
GaCircle

Defined in
$\text{HPEESOF_DIR/expressions/ael/circle_fun.ael}$

See also
gl_circle(), gp_circle(), gs_circle()

Description
This expression 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.
gain_comp()

Purpose
Returns gain compression

Synopsis

gain_comp(Sji)
    where Sji is a power-dependent complex transmission coefficient obtained from
    large-signal S-parameter simulation.

Example

gc = gain_comp(S21[:, 0])

Used in
Large-signal S-parameter simulations

Available as measurement component?
GainComp

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

See also
phase_comp()

Description
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.
generate()

Purpose
Generates a sequence of real numbers

Synopsis
generate(start, stop, npts)
where start is the first number, stop is the last number, and npts is the number of
numbers in the sequence.

Example
a = generate(9, 4, 6)
return the sequence 9., 8., 7., 6., 5., 4.

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
Not applicable

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

Purpose
Gets a data attribute

Synopsis

\[
a = \text{get_attr}(\text{data, "attr\_name"}{, \text{eval}})
\]

where data is a frequency swept variable, attr_name is the name of an attribute,
and eval is true or false as to whether to evaluate the attribute.

Example

get_attr(data, "fc", true)
10GHz

get_attr(data, "dataType")
"TimedData"

get_attr(data, "TraceType", false)
"Spectral"

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
set_attr()

Description
This function only works with frequency swept variables.
gl_circle()

Purpose
Generates a load-mismatch gain circle

Synopsis
gl_circle(S{, gain, numOfPts})
where S is the scattering matrix of a 2-port network, gain is the specified gain in dB, and numOfPts is the desired number of points per circle. The default value for gain is $10 \times \log(1 / (1 - \text{mag}(S22)^2))$ and the default value for numOfPts is 51.

Example
circleData = gl_circle(S, 2, 51)
circleData = gl_circle(S, {2, 3, 4}, 51)
return the points on the circle(s).

Used in
Small-signal S-parameter simulations

Available as measurement component?
GlCircle

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

See also
ga_circle(), gp_circle(), gs_circle()

Description
This expression 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.
gp_circle()

Purpose
Generates a power gain circle

Synopsis

\[
gp\_\text{circle}(S\{\text{gain, numOfPts}})\]

where \( S \) is the scattering matrix of a 2-port network, \( \text{gain} \) is the specified gain in dB, and \( \text{numOfPts} \) is the desired number of points per circle. The default value for \( \text{gain} \) is \( \min(\max\_\text{gain}(S)) - \{1, 2, 3\} \) and the default value for \( \text{numOfPts} \) is 51.

Example

\[
circleData = gp\_\text{circle}(S, 2, 51)
circleData = gp\_\text{circle}(S, \{2, 3, 4\}, 51)
\]

return the points on the circle(s).

Used in
Small-signal S-parameter simulations

Available as measurement component?
GpCircle

Defined in

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

See also

g\text{a\_circle()}, gl\text{\_circle()}, gs\text{\_circle()}

Description
This expression 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.
gs_circle()

Purpose
Returns a source-mismatch gain circle

Synopsis
gs_circle(S\{ gain, numOfPts\})

where S is the scattering matrix of a 2-port network, gain is the specified gain in dB, and numOfPts is the desired number of points per circle. The default value for gain is $10\times\log\left(1 / (1 - \text{mag}(S11)^2)\right)$ – \{1, 2, 3\} and the default value for numOfPts is 51.

Example
circleData = gs_circle(S, 2, 51)
circleData = gs_circle(S, \{2, 3, 4\}, 51)
return the points on the circle(s).

Used in
Small-signal S-parameter simulations

Available as measurement component?
GsCircle

Defined in
$\text{HPEESOF_DIR/expressions/ael/circle_fun.ael}$

See also
ga_circle(), gl_circle(), gp_circle()

Description
This expression 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.
histogram()

Purpose
Generates a histogram representation

Synopsis
histogram(x, numBins, minBin, maxBin)

where x is the signal, numBins is number of subintervals or bins used to measure the histogram, and minBin and maxBin are the beginning and end, respectively, of the evaluation of the histogram.

Example

```python
y = histogram(data)
y = histogram(data, 20)
```

Used in
Not applicable

Available as measurement component?
This function can only be entered by means of a Eqn component in the Data Display window. There is no measurement component in schematic window.

Defined in
Built in

See also
cdf(), pdf(), yield_sens()

Description
This function creates a histogram that represents data. The default values for minBin and maxBin are the minimum and the maximum values, respectively, of the data, and numBins is set to log(numOfPts)/log(2.0) by default.
htoabcd()

Purpose
Performs H-to-ABCD conversion

Synopsis
htoabcd(H)
   where H is the hybrid matrix of a 2-port network.

Example
a = htoabcd(h)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param
Simulation and LSSP Simulation palettes in the Schematic window. There is no
explicit measurement component.

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

See also
abcdtoh(), htoz(), ytoh()

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

**Purpose**
Performs H-to-S conversion

**Synopsis**
htos(H, Z)

where H is the hybrid matrix of a 2-port network, and Z is the reference impedance.

**Example**
s = htos(h, 50)

**Used in**
Small-signal and large-signal S-parameter simulations

**Available as measurement component?**
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

**See also**
htoy(), htoz(), stoh()

**Description**
This measurement transforms the hybrid matrix of a 2-port network to a scattering matrix.
htoy()

Purpose
Performs H-to-Y conversion

Synopsis
htoy(H)
where H is the hybrid matrix of a 2-port network

Example
Y = htoy(H)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

See also
htos(), htoz(), ytoh()

Description
This measurement transforms the hybrid matrix of a 2-port network to an admittance matrix.
htoz()

Purpose
Performs H-to-Z conversion

Synopsis
htoz(H)

   where H is the hybrid matrix of a 2-port network

Example
z = htoz(h)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

Defined in
$HPEESOF_DIR/expressions/acl/network_fun.acl

See also
htos(), htoy(), ytoh()

Description
This measurement transforms the hybrid matrix of a 2-port network to an impedance matrix.
identity()

Purpose
Returns the identity matrix

Synopsis
Y = identity(2)
Y = identity(2, 3)

The identity matrix is defined as follows. 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 \times cols is returned, again with ones on the diagonal.

Example
a = identity(2)

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
ones(), zeros()
ifc()

**Purpose**
Returns frequency-selective current in Harmonic Balance analysis

**Synopsis**
ifc(iOut, harm_freq_index)

where iOut is the current through a branch, and harm_freq_index is the harmonic index of the desired frequency. 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.

**Example**
The following example is for two tones in the harmonic balance controller:
ifc(I_Probe1.i, {1, 0})

**Used in**
Harmonic Balance simulation

**Available as measurement component?**
Ifc

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

**See also**
pfc(), vfc()

**Description**
This measurement gives the RMS current value of one frequency-component of a harmonic balance waveform.
ifc_tran()

Purpose

Returns frequency-selective current in Transient analysis

Synopsis

ifc_tran(iOut, fundFreq, harmNum)

where iOut is the current through a branch, fundFreq is the fundamental frequency
and harmNum is the harmonic number of the fundamental frequency (positive
integer value only).

Example

ifc_tran(I_Probe1.i, 1GHz, 1)

Used in

Transient simulation

Available as measurement component?

IfcTran

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See also

pfc_tran(), vfc_tran()

Description

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
containing the lowest frequency in the waveform. harmNum is the harmonic
number of the fundamental frequency at which the current is requested.
imag()

Purpose
Returns the imaginary component of a complex number

Synopsis
\[ y = \text{imag}(x) \]
where \( x \) is a complex number.

Examples
\[ a = \text{imag}(1-1\cdot j) \]
\[ -1.000 \]

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
cmplx(), real()
indep()

Purpose
Returns the independent attached to the data

Synopsis
Y = indep(x)
Y = indep(x, dimension)
Y = indep(x, "indep_name")

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

Example
Given S-parameters versus frequency and power: Frequency is the innermost independent, so its index is 1. Power has index 2.
freq = indep(S, 1)
freq = indep(S, "freq")
power = indep(S, 2)
power = indep(S, "power")

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
find_index()
int()

Purpose
Returns the largest integer not greater than a given real value

Synopsis
y = int(x)
   where x is the real value.

Examples
a = int(4.3);
4

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
abs(), cint(), exp(), float(), log10(), pow(), sgn(), sqrt()
integrate()

Purpose
Returns the integral of data

Synopsis
\[ y = \text{integrate(data\{ start, stop\{ incr}}}) \]
returns the integral of data from start to stop with increment incr.

Examples
\[ x = [0::0.01::1.0] \]
\[ y = \text{vs(2*exp(-x*x) / sqrt(pi), x)} \]
\[ z = \text{integrate(y, 0.1, 0.6, 0.001)} \]

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
$\text{HPEESOF_DIR/expressions/acl/circuit_fun.ael}$

See also
diff()

Description
Returns the integral of data from start to stop with increment incr using the composite trapezoidal rule on uniform subintervals. The default values for start and stop are the first and last points of the data, respectively. The default value for incr is “(stop - start) / (nPts - 1)” where nPts is the number of original data points between start and stop, inclusively.
**interp()**

**Purpose**
Returns linearly interpolated data

**Synopsis**

```plaintext
y = interp(data, start, stop, incr)
```

returns linearly interpolated data between start and stop with increment incr.

**Examples**

```plaintext
y = interp(data, start, stop, incr)
```

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
None

**Description**
Returns linearly interpolated data between start and stop with increment incr. The default values for start and stop are the first and last points of the data, respectively. The default value for incr is “(stop - start) / (nPts - 1)” where nPts is the number of original data points between start and stop, inclusively.
inverse()

Purpose
Performs a matrix inverse

Synopsis
y = inverse(x)

inversion of real and complex general matrices.

Example
inverse({[1, 2], [3, 4]});
{[-2, 1], [1.5, -0.5]}

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
None
ip3_in()

Purpose
Returns the input third-order intercept (TOI) point

Synopsis
ip3_in(vOut, ssGain, fundFreq, imFreq, zRef)

where vOut is the signal voltage at the output, ssGain is the small signal gain in
dB, fundFreq and imFreq are the harmonic frequency indices for the fundamental
and intermodulation frequencies, respectively, and zRef is the reference
impedance.

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

Used in
Harmonic balance simulation

Available as measurement component?
IP3in

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

See also
ip3_out(), ipn()

Description
This measurement determines the input third-order intercept point (in dBm) at the
input port with reference to a system output port.
ip3_out()

Purpose
Returns the output third-order intercept (TOI) point

Synopsis
ip3_out(vOut, fundFreq, imFreq, zRef)
where vOut is the signal voltage at the output, fundFreq and imFreq are the
harmonic frequency indices for the fundamental and intermodulation frequencies,
respectively, and zRef is the reference impedance.

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

Used in
Harmonic balance simulation

Available as measurement component?
IP3out

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

See also
ip3_in(), ipn()

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

ipn()

Purpose
Returns the output nth-order intercept (TOI) point

Synopsis
ipn(vPlus, vMinus, iOut, fundFreq, imFreq, n)

where vPlus and vMinus are the voltages at the positive and negative output terminals, respectively; iOut is the current through a branch; fundFreq and imFreq are the harmonic indices of the fundamental and intermodulation frequencies, respectively; and n is the order of the intercept.

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

Used in
Harmonic balance simulation

Available as measurement component?
IPn

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

See also
ip3_in(), ip3_out()

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

Purpose
Returns current spectrum

Synopsis
ispec_tran(iOut, fundFreq, numHarm)

   where iOut is the current through a branch, fundFreq is the fundamental
   frequency value and numHarm is the number of harmonics of fundamental
   frequency (positive integer value only).

Example
y=ispec_tran(l_Probe1.i, 1GHz, 8)

Used in
Transient simulation

Available as measurement component?
IspecTran

Defined in
$/HP/EESOF_DIR/expressions/ael/circuit_fun.ael$

See also
pspec_tran(), vspec_tran()

Description
This measurement gives a current spectrum for a specified branch. The measurement
 gives a set of RMS current values at each frequency. 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. numHarm is
 the number of harmonics of fundamental frequency to be included in the currents
 spectrum.
it()

**Purpose**
Returns time-domain current waveform

**Synopsis**
\[ \text{it}(\text{iOut}, \text{tmin}, \text{tmax}, \text{numOfPnts}) \]

where \text{iOut} is the current through a branch, \text{tmin} and \text{tmax} are start time and stop time, respectively, and \text{numOfPnts} is the number of points (integer values only).

**Example**
\[ y = \text{it}(\text{I_Probe1.i}, 0, 10\text{ns}, 201) \]

**Used in**
Harmonic balance simulation

**Available as measurement component?**
Yes

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

**See also**
vt()

**Description**
This measurement converts a harmonic-balance current frequency spectrum to a time-domain current waveform.
l_stab_circle()

Purpose
Returns a load (output) stability circle

Synopsis
l_stab_circle(S{, numOfPts})
where S is the scattering matrix of a 2-port network and numOfPts is the desired
number of points per circle and is set to 51 by default.

Example
circleData=l_stab_circle(S, 51)
returns the points on the circle(s).

Used in
Small-signal S-parameter simulations

Available as measurement component?
L_StabCircle

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

See also
l_stab_region(), s_stab_circle(), s_stab_region()

Description
The expression 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.
MeasEqn Function Reference

l_stab_region()

Purpose
Indicates the region of stability of the load (output) stability circle

Synopsis
l_stab_region(S)

where S is the scattering matrix of a 2-port network.

Example
region = l_stab_region(S)
returns “Outside” or “Inside”.

Used in
Small-signal S-parameter simulations

Available as measurement component?
Not applicable

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

See also
l_stab_circle(), s_stab_circle(), s_stab_region()

Description
This expression returns a string identifying the region of stability of the corresponding load stability circle.
**ln()**

**Purpose**
Returns the natural logarithm (ln) of an integer or real number

**Synopsis**
y = ln(x)

where x is the integer or real number.

**Examples**
a = ln(e);
returns 1

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
abs(), cint(), exp(), float(), int(), pow(), sgn(), sqrt()
log()

Purpose
Returns the base 10 logarithm of an integer or real number

Note: \textit{log10(x) perform the same operation.}

Synopsis
\begin{verbatim}
y = \text{log}(x)
\end{verbatim}
where \( x \) is the integer or real number.

Examples
\begin{verbatim}
a = \text{log}(10)
a = \text{log}(0+0i)
\end{verbatim}
1
\begin{verbatim}
NULL and an error message "log of zero"
\end{verbatim}

Available as measurement component?
Not applicable

Defined in
Built in

See also
\begin{verbatim}
abs(), cint(), exp(), log10(), float(), int(), pow(), sgn(), sqrt()
\end{verbatim}
**log10()**

**Purpose**
Returns the base 10 logarithm of an integer or real number

**Note:** log(x) perform the same operation.

**Synopsis**
y = log10(x)
where x is the integer or real number.

**Examples**
a = log10(10)
1
a = log10(0+0i)
NULL and an error message "log of zero"

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
abs(), cint(), exp(), log(), float(), int(), pow(), sgn(), sqrt()
mag()

Purpose
Returns the magnitude of a complex number

Synopsis
\( y = \text{mag}(x) \)
where \( x \) is a complex number.

Examples
\[
a = \text{mag}(3-4\text{j}) \\
5.000
\]

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
\text{conj()}
map1_circle()

Purpose
Returns source-mapping circles from port 1 to port 2

Synopsis
circleData=map1_circle(S, numOfPts)

where S is the scattering matrix of a 2-port network and numOfPts is the desired
number of points per circle and is set to 51 by default.

Example
circleData=map1_circle(S, 51)
returns the points on the circle(s).

Used in
Small-signal S-parameter simulations

Available as measurement component?
Map1Circle

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

See also
map2_circle()

Description
The expression maps the set of terminations with unity magnitude at port 1 to port 2.
The circles are defined by the loci of terminations on one port as seen at the other
port.

A source-mapping circle is created for each value of the swept variable(s). This
measurement is supported for 2-port networks only.
map2_circle()

Purpose
Returns source-mapping circles, from port 2 to port 1

Synopsis

\[
\text{circleData} = \text{map2_circle}(S, \text{numOfPts})
\]

where \( S \) is the scattering matrix of a 2-port network and \( \text{numOfPts} \) is the desired number of points per circle and is set to 51 by default.

Example

\[
\text{circleData} = \text{map2_circle}(S, 51)
\]
returns the points on the circle(s).

Used in
Small-signal S-parameter simulations

Available as measurement component?
Map2Circle

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

See also
map1_circle()

Description
The expression maps the set of terminations with unity magnitude at port 2 to port 1. The circles are defined by the loci of terminations on one port as seen at the other port.

A source-mapping circle is created for each value of the swept variable(s). This measurement is supported for 2-port networks only.
max()

Purpose
Returns the maximum value

Synopsis
Y = max(x)
The function takes a single argument, so enclose a sequence of numbers in brackets “[x, y, ...]”

Example
a = max([1, 2, 3])
3

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
cum_prod(), cum_sum(), mean(), min(), prod(), sum()
max_gain()

Purpose
Returns the maximum available and stable gain

Synopsis
max_gain(S)
    where S is a scattering matrix of 2-port network.

Example
y=max_gain(S)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
MaxGain

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

See also
sm_gamma1(), sm_gamma2(), stab_fact(), stab_meas()

Description
Given a 2 x 2 scattering matrix, this measurement returns the maximum available and stable gain between the input and the measurement ports.
max_index()

Purpose
Returns the index of the maximum

Synopsis
max_index(x)
The function takes a single argument, so enclose a sequence of numbers in brackets “[x, y, ...]”

Example
max_index([1, 2, 3])
2
max_index([3, 2, 1])
0

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built-in

See also
min_index()
**max_outer()**

**Purpose**
Computes the maximum across the outer dimension of two-dimensional data.

**Synopsis**

```
max_outer(data)
```

where data must be two-dimensional data.

**Example**

```
max_outer(data)
```

**Used in**
Not applicable

**Available as measurement component?**
No, but the function can be used on a schematic page, in a measurement equation.

**Defined in**

`$HPEESOF_DIR/expressions/ael/statistcal_fun.ael`

**See also**

`fun_2d_outer(), mean_outer(), min_outer()`

This function can be applied to the data in the example:

`.../examples/Tutorial/DataAccess_prj/Truth_MonteCarlo.dds`

**Description**

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.
**mean()**

**Purpose**
Returns the mean

**Synopsis**
```plaintext
mean(x)
```
The function takes a single argument, so enclose a sequence of numbers in brackets “[x, y, ...]”

**Example**
```plaintext
mean([1, 2, 3])
```
2

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
cum_prod(), cum_sum(), max(), min(), prod(), sum()
mean_outer()

Purpose
Computes the mean across the outer dimension of two-dimensional data.

Synopsis
mean_outer(data)

where data must be two-dimensional data.

Example
mean_outer(data)

Used in
Not applicable

Available as measurement component?
No, but the function can be used on a schematic page, in a measurement equation.

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

See also
fun_2d_outer(), max_outer(), min_outer()

This function can be applied to the data in the example:
.../examples/Tutorial/DataAccess_prj/Truth_MonteCarlo.dds.

Description
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()

Purpose
Returns the median

Synopsis
median(x)

The function takes a single argument, so enclose a sequence of numbers in brackets “[x, y, …]”

Example
median([1, 2, 3, 4])
2.5

Used in
Not applicable

Available as measurement component?
This function can only be entered by means of a Eqn component in the Data Display window. There is no explicit measurement component.

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

See also
mean(), sort()
min()

**Purpose**
Returns the minimum value of a swept parameter

**Synopsis**
y = min(x)
The function takes a single argument, so enclose a sequence of numbers in brackets “[x, y, ...]”

**Examples**
a = min([1, 2, 3]);
1

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
cum_prod(), cum_sum(), max(), mean(), prod(), sum()
min_index()

Purpose
Returns the index of the minimum

Synopsis
y = min_index(x)
The function takes a single argument, so enclose a sequence of numbers in brackets “[x, y, ...]”

Example
min_index([3, 2, 1])
2
min_index([1, 2, 3])
0

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
max_index()
min_outer()

Purpose
Computes the minimum across the outer dimension of two-dimensional data.

Synopsis
min_outer(data)

where data must be two-dimensional data.

Example
min_outer(data)

Used in
Not applicable

Available as measurement component?
No, but the function can be used on a schematic page, in a measurement equation.

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

See also
fun_2d_outer(), max_outer(), mean_outer()

This function can be applied to the data in the example:
.../examples/Tutorial/DataAccess_prj/Truth_MonteCarlo.dds.

Description
The min function operates on the inner dimension of two-dimensional data. The
min_outer function just calls the fun_2d_outer function, with min being the applied
operation. As an example, assume that a Monte Carlo simulation of an amplifier was
run, with 151 random sets of parameter values, and that for each set the
S-parameters were simulated over 26 different frequency points. S21 becomes a [151
Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being
frequency, and the outer dimension being Monte Carlo index. Now, assume that it is
desired to know the minimum value of the S-parameters at each frequency. Inserting
an equation min(S21) computes the minimum value of S21 at each Monte Carlo
iteration. If S21 is simulated from 1 to 26 GHz, it computes the minimum value over
this frequency range, which usually is not very useful. Inserting an equation
min_outer(S21) computes the minimum value of S21 at each Monte Carlo iteration.
mix()

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

Synopsis
mix(xOut, harmIndex{, Mix})

where xOut is a voltage or a current spectrum and harmIndex is the desired vector of harmonic frequency indices (mixing terms). Mix is a variable consisting of all possible vectors of harmonic frequency indices (mixing terms) in the analysis.

Example
y = mix(vOut, [2, -1])
z = mix(vOut+vOut/50, [2, -1], Mix)

Used in
Harmonic balance simulation

Available as measurement component?
This equation can be entered by means of a MeasEqn component in Harmonic balance simulation palette in the Schematic window. There is no explicit Mix measurement component.

Defined in
Built in

See also
find_index()

Description
This function returns the mixing component of a voltage or a current spectrum corresponding to particular harmonic-frequency indices or mixing terms. Note that the third argument, Mix, is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.
moving_average()

Purpose
Returns the moving_average of a sequence of data

Synopsis
moving_average(data, numPoints)
    where data is a one-dimensional sequence of numbers in brackets “[x, y, ...]”, and
    numPoints is the number of points to be averaged together.

Example
moving_average([1, 2, 3, 7, 5, 6, 10], 3)
[1, 2, 4, 5, 6, 7, 10]

Used in
Not applicable

Available as measurement component?
There is no explicit measurement component, but the function can be used on a schematic page.

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

See also
Not applicable

Description
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.
mu()

Purpose
Returns the geometrically derived stability factor for the load

Synopsis
mu(S)
   where S is a scattering matrix of a 2-port network.

Examples
x=mu(S)

Used in
Small-signal and large-signal S-parameter simulations.

Available as measurement component?
Mu

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

See also
mu_prime()

Description
This measurement gives the distance from the center of the Smith chart to the nearest output (load) stability circle.

This stability factor is given by
mu = {1-|S11|^2} / { |S22 - conj(S11)*Delta | + |S12*S21| }

where Delta is the determinant of the S-parameter matrix. Having mu > 1 is the single necessary and sufficient condition for unconditional stability of the 2-port network.

Reference
mu_prime()

Purpose
Returns the geometrically derived stability factor for the source

Synopsis
mu_prime(S)
where S is a scattering matrix of 2-port network.

Examples
y=mu_prime(S)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
MuPrime

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

See also
mu()

Description
This measurement gives the distance from the center of the Smith chart to the
nearest unstable-input (source) stability circle.
This stability factor is given by
mu_prime = {1-|S22|**2} / {|S11 - conj(S22)*Delta | + |S21*S12| }
where Delta is the determinant of the S-parameter matrix. Having mu_prime > 1
is the single necessary and sufficient condition for unconditional stability of the
2-port network.

Reference
using geometrically derived parameters”, IEEE Transactions on Microwave
ns_circle()

Purpose

Returns noise-figure circles

Synopsis

ns_circle(nf, NFmin, Sopt, rn{, numOfPts})

where nf is the specified noise figure and is set by default to \( \max(NF \text{ min}) + \{0, 1, 2, 3\} \) \( NF \text{ min} \) is the minimum noise figure, Sopt is the optimum mismatch, \( rn \) is the equivalent normalized noise resistance of a 2-port network (\( rn = Rn/z\text{Ref} \) where \( Rn \) is the equivalent noise resistance and \( z\text{Ref} \) is the reference impedance), and numOfPts is the desired number of points per circle and is set to 51 by default.

Example

\[
\begin{align*}
\text{circleData}= & \text{ns}\_\text{circle}(0+NF \text{ min}, NF \text{ min}, \text{Sopt}, Rn/50, 51) \\
\text{circleData}= & \text{ns}\_\text{circle}(0, 1+NF \text{ min}, NF \text{ min}, \text{Sopt}, Rn/50, 51)
\end{align*}
\]

returns the points on the circle(s).

Used in

Small-signal S-parameter simulations

Available as measurement component?

NsCircle

Defined in

$\text{HPEESOF DIR/expressions/ael/circle_fun.ael}$

See also

Not applicable

Description

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. \( NF \text{ min} \), Sopt, and \( Rn \) are generated from noise analysis. A circle is created for each value of the swept variable(s).
ns_pwr_int()

Purpose
Returns the integrated noise power

Synopsis
ns_pwr_int(Sji, nf, resBW)

where Sji is the complex transmission coefficient, nf is noise figure at the output port (in dB), and resBW is the user-defined resolution bandwidth.

Example
Y=ns_pwr_int(S21, nf2, 1MHz)

Used in
Small-signal S-parameter simulation

Available as measurement component?
NsPwrInt

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

See also
ns_pwr_ref_bw(), snr()

Description
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()

Purpose
Returns noise power in a reference bandwidth

Synopsis
Y = ns_pwr_ref_bw(Sji, nf, resBW)
where Sji is the complex transmission coefficient, nf is noise figure at the output
port (in dB), and resBW is the user-defined resolution bandwidth.

Example
Y = ns_pwr_ref_bw(S21, nf2, 1MHz)
returns the noise power with respect to the reference bandwidth.

Used in
Small-signal S-parameter simulation

Available as measurement component?
NsPwrRefBW

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

See also
ns_pwr_int(), snr()

Description
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.
ones()

Purpose
Returns a matrix of ones

Synopsis
Y = ones(2)
This is the ones matrix. If only one argument is supplied, then a square matrix is returned. If two are supplied, then a matrix of ones with size rows × cols is returned.

Example
a = ones(2)
{1, 1; 1, 1}

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
identity(), zeros()
**pae()**

**Purpose**

Returns power-added efficiency

**Synopsis**

\[
pae(v_{\text{PlusOut}}, v_{\text{MinusOut}}, v_{\text{PlusIn}}, v_{\text{MinusIn}}, v_{\text{PlusDC}}, v_{\text{MinusDC}}, i_{\text{Out}}, i_{\text{In}}, i_{\text{DC}}, \text{outFreq}, \text{inFreq})
\]

where \(v_{\text{PlusOut}}\) and \(v_{\text{MinusOut}}\) are output voltages at the positive and negative terminals; \(v_{\text{PlusIn}}\) and \(v_{\text{MinusIn}}\) are input voltages at the positive and negative terminals; \(v_{\text{PlusDC}}\) and \(v_{\text{MinusDC}}\) are DC voltages at the positive and negative terminals; \(i_{\text{Out}}, i_{\text{In}}, \) and \(i_{\text{DC}}\) are the output, input, and DC currents, respectively; and \(\text{outFreq}\) and \(\text{inFreq}\) are harmonic indices of the fundamental frequency at the output and input port, respectively.

**Example**

\[
y = \text{pae}(v_{\text{Out}}, 0, v_{\text{In}}, 0, v_{1}, 0, l_{\text{Probe1}.i}, l_{\text{Probe2}.i}, l_{\text{Probe3}.i}, 1, 1)
\]

**Used in**

Harmonic balance simulation

**Available as measurement component?**

PAE

**Defined in**

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

**See also**

\(\text{dB}(), \text{dBm}()\)

**Description**

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

Purpose
Returns a probability density function

Synopsis
pdf(x, numBins, minBin, maxBin)

where x is the signal, numBins is number of subintervals or bins used to measure PDF, and minBin and maxBin are the beginning and end, respectively, of the evaluation of the PDF.

Example
y = pdf(data)
y = pdf(data, 20)

Used in
Not applicable

Available as measurement component?
This function can be entered by means of a Eqn component in the Data Display window. There is no measurement component in schematic window

Defined in
AEL statistical_fun.ael

See also
cdf(), histogram(), yield_sens()

Description
This function measures the probability density function of a signal. The default values for minBin and maxBin are the minimum and the maximum values of the data and numBins is set to log(numOfPts)/log(2.0) by default.
permute()

Purpose
Permutes data based on the attached independents

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

where \( \text{data} \) is any \( N \)-dimensional square data (all inner independents must have the same value \( N \)) and \( \text{permute\_vector} \) is any permutation vector of the numbers 1 through \( N \). The \( \text{permute\_vector} \) defaults to \{\( N \):1\} representing a complete reversal of the data with respect to its independent variables. If \( \text{permute\_vector} \) 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
\begin{align*}
a &= \text{permute}(\text{data}) \\
a &= \text{permute}(\text{data}, \{3, 2, 1\}) \\
\end{align*}

reverses the (three inner independents of) the data.

\begin{align*}
a &= \text{permute}(\text{data}, \{1, 2, 3\}) \\
\end{align*}

preserves the data.

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
None
pfc()

Purpose
Returns frequency-selective power

Synopsis
pfc(vPlus, vMinus, iOut, harm_freq_index)

where vPlus is the voltage at the positive and negative terminals, iOut is the
current through a branch, and harm_freq_index is the harmonic index of the
desired frequency. 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.

Example
The following example is for two tones in the harmonic balance controller:

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

Used in
Harmonic balance simulation

Available as measurement component?
Pfc

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

See also
ifc(), vfc()

Description
This measurement gives the RMS power value of one frequency component of a
harmonic balance waveform.
pfc_tran()

**Purpose**

Returns frequency-selective power

**Synopsis**

pfc_tran(vPlus, vMinus, iOut, fundFreq, harmNum)

where vPlus and vMinus are the voltages at the positive terminals, iOut is the current through a branch measured for power calculation, and fundFreq is fundamental frequency and harmNum is the harmonic number of the fundamental frequency (positive integer value only).

**Example**

\[ y = pfc\_tran(v1, v2, I\_Probe1.i, 1GHz, 1) \]

**Used in**

Transient simulation

**Available as measurement component?**

PfcTran

**Defined in**

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

**See also**

ifc_tran(), vfc_tran()

**Description**

This measurement gives RMS power, delivered to any part of the circuit at a particular frequency of interest. fundFreq determines the portion of the time-domain waveform to be converted to frequency domain. This is typically one full period corresponding to the lowest frequency in the waveform. harmNum is the harmonic number of the fundamental frequency at which the power is requested.
phase()

Purpose
Phase in degrees

Synopsis
y = phase(x)

Example
phase(1i)
90
phase(1+1i)
45

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built-in

See also
phaserad()
phase_comp()

Purpose
Returns the phase compression (phase change)

Synopsis
Y = phase_comp(Sji)
where Sji is a power-dependent parameter obtained from large-signal S-parameters simulation.

Example
y = phase_comp(S21[:, 0])

Used in
Large-signal S-parameter simulations

Available as measurement component?
PhaseComp

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

See also
gain_comp()

Description
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.
phasedeg()

Purpose
Phase in degrees

Synopsis
y = phasedeg(x)

Example
phase(1i)
90
phase(1+1i)
45

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built-in

See also
dev_lin_phase(), diff(), phase(), phasedeg(), phaserad(), ripple(), unwrap()
phaserad()

Purpose
Phase in Radians

Synopsis
\[ y = \text{phaserad}(x) \]

Example
phaserad(1i)
1.5708
phaserad(1+1i)
0.785398

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
\text{dev_lin_phase()}, \text{diff()}, \text{phase()}, \text{phasedeg()}, \text{phaserad()}, \text{ripple()}, \text{unwrap()}
plot_vs()

Purpose
Attaches an independent to data for plotting

Synopsis
plot_vs(dependent, independent)

where dependent is any N-dimensional square data (all inner independents must have the same value N) and permute_vector is any permutation vector of the numbers 1 through N. 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.

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

Builds c with independent b, and dependent a.

Used in
Not applicable

Available as measurement component?
Not applicable

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

See also
indep(), vs()
polar()

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

Synopsis
polar(mag, angle)

Example
polar(1, 90)
0+1i
polar(1, 45)
0.707107+0.707107i

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
None
pow()

Purpose
 Raises an integer or real number to a given power

Synopsis
 \[ z = \text{pow}(x, y) \]
 where \( x \) is the integer or real number and \( y \) is the exponent of that number.

Examples
 \[ a = \text{pow}(4, 2); \]
 returns 16

Used in
 Not applicable

Available as measurement component?
 Not applicable

Defined in
 Built in

See also
 abs(), cint(), exp(), float(), int(), log10(), sgn(), sqrt()
pspec()

Purpose
Returns power frequency spectrum

Synopsis
pspec(vPlus, vMinus, iOut)
where vPlus and vMinus are voltages at the positive terminals, and iOut is the current through a branch measured for power calculation.

Example
y=pspec(vOut, 0, I_Probe1.i)

Used in
Harmonic balance simulation

Available as measurement component?
Pspec

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

See also
Not applicable

Description
This measurement gives a power frequency spectrum in harmonic balance analyses.
pspec_tran()

Purpose
Returns transient power spectrum

Synopsis
pspec_tran(vPlus, vMinus, iOut, fundFreq, numHarm)
where vPlus and vMinus are the voltages at the positive and negative terminals, iOut is the current through a branch measured for power calculation, fundFreq is the fundamental frequency, and numHarm is the number of harmonics of the fundamental frequency (positive integer value only).

Example
y=pspec_tran(v1, v2, i_Probe1.i, 1GHz, 8)

Used in
Transient simulation

Available as measurement component?
Pspectran

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

See also
ispec_tran(), vspec_tran()

Description
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. fundFreq is the fundamental frequency 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). numHarm is the number of harmonics of the fundamental frequency to be included in the power spectrum.
prod()

**Purpose**
Returns the product

**Synopsis**
prod(x)

**Example**
prod([1, 2, 3])
6
prod([4, 4, 4])
64

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built-in

**See also**
sum()
pt()

Purpose
Returns total power

Synopsis
pt(vPlus, vMinus, iOut)

where vPlus and vMinus are the voltages at the positive and negative terminals, respectively, and iOut is the current through a branch.

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

Used in
Harmonic balance simulation

Available as measurement component?
Pt

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

See also
pspec()

Description
This measurement calculates the total power of a harmonic balance frequency spectrum.
pwr_gain()

Purpose
Returns power gain

Synopsis
y=pwr_gain(S, Zs, Zl { Zref})

where S is the 2 x 2 scattering matrix, and Zs and Zl are the input and output impedances, respectively. Zref is the reference impedance, set by default to the port impedance.

Example
y=pwr_gain(S, 50, 75)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
PwrGain

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

See also
stos(), volt_gain(), volt_gain_max()

Description
This measurement is used to determine the power gain (in dB), i.e. the power delivered to the load minus the power available from the source (where power is in dBm).
rad()

**Purpose**
Degrees to radians

**Synopsis**
rad(x)

**Example**
rad(90)
1.5708
rad(45)
0.785398

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
deg()
real()

Purpose
Returns the real part of a complex number

Synopsis
\[ y = \text{real}(x) \]
where \( x \) is a complex number.

Examples
\[ a = \text{real}(1-1j); \]
returns 1

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
\text{cmplx()}, \text{imag()}
relative_noise_bw()

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

Synopsis
RelNoiseBW=relative_noise_bw(winType, winConst)

where
winType is a window type and must be one of the following: Kaiser, Hamming, Gaussian, 8510, or NoWindow (leaving this field blank is the equivalent of NoWindow); and

winConst is an optional parameter that affects the shape of the applied window. The default window constants are as follows:
   Kaiser: 7.865
   Hamming: 0.54
   Gaussian: 0.75
   8510: 6 (The 8510 window is the same as a Kaiser window with a window constant of 6.)

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

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

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

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

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

**Available as measurement component?**
Equations listed under Description can be entered by means of a MeasEqn component in the Schematic window. There is no explicit relative noise bandwidth measurement function.

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

**See also**
acpr_vi(), acpr_vr(), channel_power_vi(), channel_power_vr(), fs()  

**Description**
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.
ripple()

Purpose
Returns deviation from the average

Synopsis
ripple(x)
    where x can be a gain or group delay data over a given frequency range.

Example
y=ripple(pwr_gain(S21))

Used in
Not applicable

Available as measurement component?
GainRipple

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

See also
dev_lin_phase(), diff(), mean(), phasedeg(), phaserad(), unwrap()

Description
This function measures the deviation of x from the average of x.
round()

Purpose
Rounds to the nearest integer

Synopsis
round(x)

Example
round(0.1)  0
round(0.5)  1
round(0.9)  1
round(-0.1) 0
round(-0.5) -1
round(-0.9) -1

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
int()
s_stab_circle()

Purpose
Returns source (input) stability circles

Synopsis
s_stab_circle(S{, numOfPts})
where S is the scattering matrix of a 2-port network and numOfPts is the desired
number of points per circle and is set to 51 by default.

Example
circleData = s_stab_circle(S, 51)
returns the points on the circle(s).

Used in
Small-signal S-parameter simulations

Available as measurement component?
S_StabCircle

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

See also
l_stab_circle(), l_stab_region(), s_stab_region()

Description
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.
s_stab_region()

**Purpose**
Indicates the region of stability of the source (input) stability circle

**Synopsis**

s_stab_region(S)

where S is the scattering matrix of a 2-port network.

**Example**

```plaintext
region = s_stab_region(S)
```
returns “Outside” or “Inside”.

**Used in**
Small-signal S-parameter simulations

**Available as measurement component?**
Not applicable

**Defined in**

$HPEESOF_DIR/expressions/ael/circle_fun.ael$

**See also**

l_stab_circle(), l_stab_region(), s_stab_circle()

**Description**
This expression returns a string identifying the region of stability of the corresponding source stability circle.
sample_delay_pi4dqpsk()

Purpose
This function calculates the optimal sampling point within a symbol for a given pi4dqpsk waveform.

Synopsis
sample_delay_pi4dqpsk(vlQ, symbolRate, delay, timeResolution)

where
vlQ is the complex envelope (I + j * Q) of a pi/4 DQPSK signal.
symbolRate is the symbol rate of the pi/4 DQPSK signal.
path is the time delay on the waveform before the sampling starts. 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().
timeResolution is the time step (typically one-tenth of a symbol time or less) used to search for the best sampling point in a given symbol period.

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

Used in
Envelope simulation

Available as measurement component?
Not applicable

Defined in
Built in

See also
ber_pi4dqpsk(), ber_qpsk(), const_evm()

Description
Calculates the optimal sampling point for a given waveform. "Optimal" is defined as the sampling point that provides the lowest bit error rate.
sample_delay_qpsk()

Purpose
This function calculates the optimal sampling point within a symbol for a given QPSK waveform.

Synopsis
sample_delay_qpsk(vlQ, symbolRate, delay, timeResolution)
where
vlQ is the complex envelope (I + j * Q) of a QPSK signal.
symbolRate is the symbol rate of the QPSK signal.
path is the time delay on the waveform before the sampling starts. 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().
timeResolution is the time step (typically one-tenth of a symbol time or less) used to search for the best sampling point in a given symbol period.

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

Used in
Envelope simulation

Available as measurement component?
Not applicable

Defined in
Built in

See also
ber_pi4dqpsk(), ber_qpsk(), const_evm()

Description
Calculates the optimal sampling point for a given waveform. "Optimal" is defined as the sampling point that provides the lowest bit error rate.
**set_attr()**

**Purpose**
Sets a data attribute

**Synopsis**
```
a = set_attr(data, "attr_name", attribute_value)
```

**Example**
```
set_attr(data, "TraceType", "Spectral")
set_attr(data, "TraceType", 10GHz)
```

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
`get_attr()`
sfdr()

Purpose
Returns the spurious-free dynamic range

Synopsis
sfdr(vOut, ssgain, nf, noiseBW, fundFreq, imFreq, zRef)

where vOut is the output voltage, ssgain is the small-signal gain (in dB), nf is the
noise figure at the output port, noiseBW is the noise bandwidth, fundFreq and
imFreq are the harmonic frequency indices for the fundamental and
intermodulation frequencies, respectively, and zRef is the reference impedance.

Example
y=sfdr(vIn, 12, nf2, , {1, 0}, {2, -1}, 50)

Used in
In a Harmonic Balance simulation, sfdr is used in Small-signal S-parameter
simulations. It appears in the HB Simulation palette.

Available as measurement component?
SFDR

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

See also
ip3_out()

Description
This measurement determines the spurious-free dynamic-range ratio for noise power
with respect to the reference bandwidth. zRef is an optional parameter that, if not
specified, is set to 50.0 ohms.
sgn()

Purpose
Returns the integer sign of an integer or real number, as either 1 or –1

Synopsis
y = sgn(x)
where x is an integer or real number.

Examples
a = sgn(-1)
returns -1
a = sgn(1)
returns 1

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
abs(), cint(), exp(), float(), int(), log10(), pow(), sqrt()
**sin()**

**Purpose**
Returns the sine of an integer or real number

**Synopsis**
y = sin(x)

where x is an integer or real number, in radians.

**Examples**
a = sin(pi/2)
returns 1

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
cos(), tan()
sinc()  
Purpose  
Returns the sinc of an integer or real number  
Synopsis  
y = sinc(x)  
    where x is an integer or real number, in radians.  
Examples  
a = sinc(0.5)  
0.637  
Used in  
Not applicable  
Available as measurement component?  
Not applicable  
Defined in  
Built in  
See also  
sin()  
Description  
The sinc function is defined as sinc(x) = sin(pi*x) / (pi*x) and sinc(0)=1.
**sinh()**

**Purpose**
hyperbolic sin

**Synopsis**

```
sinh()
```

**Example**

```
sinh(0)
0
```

```
sinh(1)
1.1752
```

**Used in**

Not applicable

**Available as measurement component?**

Not applicable

**Defined in**

Built in

**See also**

cosh(), tanh()
size()

Purpose
Returns the row and column size of a vector or matrix

Synopsis
Y = size(X)

Example
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:

size(S) returns {2, 2}
sweep_size(S) returns 10

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
sweep_size()
sm_gamma1()

Purpose
Returns the simultaneous-match input-reflection coefficient

Synopsis
sm_gamma1(S)

where S is a scattering matrix of 2-port network.

Example
y=sm_gamma1(S)

Used in
Small-signal and large-signal S-parameter simulations.

Available as measurement component?
SmGamma1

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

See also
max_gain(), sm_gamma2(), stab_fact(), stab_meas()

Description
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 zero. It is, in effect, undefined when stab_fact(S) < 1.
sm_gamma2()

Purpose
Returns the simultaneous-match output-reflection coefficient

Synopsis
sm_gamma2(S)
where S is a scattering matrix of 2-port network.

Example
y = sm_gamma2(S)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
SmGamma2

Defined in
$HPEESOF_DIR/expressions/acl/rl_system_fun.ael

See also
max_gain(), sm_gamma1(), stab_fact(), stab_meas()

Description
This complex measurement determines the reflection coefficient that must be
presented to the output (port 2) of the network to achieve simultaneous input and
output reflections. If the Rollett stability factor stab_fact(S) is less than unity for the
analyzed circuit, then sm_gamma2(S) returns zero. It is, in effect, undefined when
stab_fact(S) < 1.
sm_y1()

Purpose
Returns the simultaneous-match input admittance

Synopsis
sm_y1(S, Z)
    where S is a scattering matrix of a 2-port network, and Z is a port impedance.

Example
y = sm_y1(S, 50)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
SmY1

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

See also
sm_y2()

Description
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.
**sm_y2()**

**Purpose**
Returns the simultaneous-match output admittance

**Synopsis**
sm_y2(S, Z)

where S is a scattering matrix of 2-port network and
Z is a port impedance.

**Example**
y=sm_y2(S, 50)

**Used in**
Small-signal and large-signal S-parameter simulations

**Available as measurement component?**
SmY2

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

**See also**
sm_y1()

**Description**
This complex measurement determines the admittance that must be presented to the input (port 2) of the network to achieve simultaneous input and output reflections.
**sm_z1()**

**Purpose**
Returns the simultaneous-match input impedance

**Synopsis**

\[ \text{sm}_z1(S, Z) \]

where \( S \) is a scattering matrix of a 2-port network, and \( Z \) is a port impedance.

**Example**

\[ y = \text{sm}_z1(S, 50) \]

**Used in**
Small-signal and large-signal S-parameter simulations

**Available as measurement component?**
SmZ1

**Defined in**

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

**See also**

`sm_z2()`

**Description**

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.
sm_z2()

Purpose
Returns the simultaneous-match output impedance

Synopsis
Y = sm_z2(S, Z)

where S is a scattering matrix of 2-port network, and Z is a port impedance.

Example
y=sm_z2(S, 50)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
SmZ2

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

See also
sm_z1()

Description
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.
**snr()**

**Purpose**
Returns the signal-to-power noise ratio

**Synopsis**

\[
\text{snr}(\text{vOut}, \text{vOut}.\text{noise}\{\text{fundFreq}\})
\]

where \(\text{vOut}\) and \(\text{vOut}.\text{noise}\) are the signal and noise voltages at the output port, and \(\text{fundFreq}\) is the harmonic frequency index for the fundamental frequency. Note that \(\text{fundFreq}\) is not optional; it is required for harmonic balance simulations, but it is not applicable in AC simulations.

**Example**

\[
y = \text{snr}(\text{vOut}, \text{vOut}.\text{noise}, \{1, 0\})
\]

returns the signal-to-power noise ratio for a harmonic balance simulation.

\[
y = \text{snr}(\text{vOut}, \text{vOut}.\text{noise})
\]

returns the signal-to-power noise ratio for an AC simulation.

**Used in**
Harmonic balance simulations

**Available as measurement component?**
SNR

**Defined in**

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

**See also**
ns_pwr_int(), ns_pwr_ref_bw()

**Description**

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).
sort()

Purpose
Returns a sorted variable

Synopsis
sort(data, sortOrder, indepName)

where data is a multidimensional scalar variable, sortOrder is the sorting order, 
{"ascending", "decending"}. (If not specified, it is set to “ascending.”) indepName is 
used to specify the name of the independent variable for sorting. (If not specified, 
the sorting is done on the dependent.)

Example
y = sort(data)
y = sort(data, “decending”, “freq”)

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
None

Description
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.
MeasEqn Function Reference

**spec_power()**

**Purpose**

Returns the integrated signal power (dBm) of a spectrum

**Synopsis**

`spec_power(sinkInstanceName{, lowerFrequencyLimit, upperFrequencyLimit})`

where `sinkInstanceName` is the instance of the FFTAnalyzer or SpecAnalyzer sink in the DSP schematic window (values in dBm). `lowerFrequencyLimit` and `upperFrequencyLimit` are optional and define the lower and upper frequency limits to be used in calculating the integrated power. The frequency unit used must match that used in SpecAnalyzer. The default unit is MHz. The entire spectral frequency range will be used if `lowerFrequencyLimit` and `upperFrequencyLimit` are not specified.

**Example**

```
total_power = spec_power(Mod_Spectrum, 60, 71)
```

returns the integrated power between 60 and 71 MHz

```
total_power = spec_power(Mod_Spectrum, indep(m1), indep(m2))
```

returns the integrated power between markers 1 and 2

where `Mod_Spectrum` is the instance ID of an FFTAnalyzer or SpecAnalyzer sink

**Used in**

Agilent Ptolemy simulations

**Available as measurement component?**

This function is available for use in a MeasEqn component.

**Defined in**

`$HPEESOF_DIR/expressions/ael/signal_proc_fun.ael`

**See also**

None

**Description**

This function will return the total integrated power (dBm) of a spectrum. The frequency window over which the integrated power will be calculated can be specified, otherwise the entire spectral frequency range will be used.
The FFTAnalyzer and SpecAnalyzer sinks are valid for this measurement. These sinks should have a termination resistor shunted to ground at their input for a measurement referenced to an impedance such as 50 Ohms. This termination resistor value should be set to the Ref Resistance value specified in the FFTAnalyzer or SpecAnalyzer sink. The display parameter of the sinks must be set to dBm.
MeasEqn Function Reference

**spur_track()**

**Purpose**
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.

**Synopsis**

```plaintext
IFspur=spur_track(vs(vout, freq), if_low, if_high, rout)
```

where `vout` is the IF output node name, `if_low` is the lowest frequency in the IF band, `if_high` is the highest frequency in the IF band, `rout` is the load resistance connected to the IF port, necessary for computing power delivered to the load. 

`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.

**Example**

```plaintext
IFspur=spur_track(vs(HB.VIF1, freq), Fiflow[0, 0], Fifhigh[0, 0], 50)
```

where `VIF1` is the named node at the IF output, `Fiflow` is the lowest frequency in the IF band, `Fifhigh` is the highest frequency in the IF band, and 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.

**Used in**
Receiver spurious response simulations

**Available as measurement component?**
No, but the function can be used on a schematic page, in a measurement equation.

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

**See also**
`spur_track_with_if()`
This function can be applied to the data in the example:
.../examples/Com_Sys/Spur_Track_prj/MixerSpurs2MHz.dds.

**Description.**

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()

Purpose
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.

Synopsis
IFspur=spur_track_with_if(vs(vout, freq), if_low, if_high, rout)

where vout is the IF output node name, if_low is the lowest frequency in the IF band, if_high is the highest frequency in the IF band, rout is the load resistance connected to the IF port, necessary for computing power delivered to the load. 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.

Example
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, and 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.

Used in
Receiver spurious response simulations

Available as measurement component?
No, but the function can be used on a schematic page, in a measurement equation.

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

See also
spur_track()
This function can be applied to the data in the example:

`.../examples/Com_Sys/Spur_Track_prj/MixerSpurs2MHz.dds`

**Description**

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.
sqrt()

Purpose
Returns the square root of a positive integer or real number

Synopsis
y = sqrt(x)
where x is a positive integer or real number.

Examples
a = sqrt(4)
returns 2

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
abs(), cint(), exp(), float(), int(), log10(), pow(), sgn()
stab_fact()

Purpose
Returns the Rollett stability factor

Synopsis
stab_fact(S)
    where S is the scattering matrix of a 2-port network.

Example
k = stab_fact(S)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
StabFact

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

See also
max_gain(), sm_gamma1(), sm_gamma2(), stab_meas()

Description
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()

Purpose
Returns the stability measure

Synopsis
stab_meas(S)

where S is the scattering matrix of a 2-port network.

Example
b = stab_meas(S)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
StabMeas

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

See also
max_gain(), sm_gamma1(), sm_gamma2(), stab_fact()

Description
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 + |S11|^2 - |S22|^2 - |S11*S22 - S12*S21|^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
stddev()

Purpose
Returns the standard deviation

Synopsis
stddev(x{, flag})

   where x is the data and flag is used to indicate how stddev normalizes. By default, flag is set to 0, which means that stddev normalizes by N-1, where N is the length of the data sequence. Otherwise, stddev normalizes by N.

Example
y = stddev(data)
y = stddev(data, 1)

Used in
Not applicable

Available as measurement component?
This function can only be entered by means of a Eqn component in the Data Display window.

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

See also
mean()

Description
This function calculates the standard deviation of the data.
**stoabcd()**

**Purpose**
Performs S-to-ABCD conversion

**Synopsis**
stoabcd(S, zRef)

where S is a scattering matrix of a 2-port network and zRef is a reference impedance.

**Example**
a = stoabcd(S, 50)

**Used in**
Small-signal and large-signal S-parameter simulations

**Available as measurement component?**
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

**See also**
abcdtoh(), stoh(), stoy()

**Description**
This measurement transforms the scattering matrix of a 2-port network to a chain (ABCD) matrix.
stoh()

Purpose
Performs S-to-H conversion

Synopsis
stoh(S, zRef)

where S is a scattering matrix of a 2-port network and zRef is a reference impedance.

Example
h = stoh(S, 50)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

See also
htos(), stoabcd(), stoy()

Description
This measurement transforms the scattering matrix of a 2-port network to a hybrid matrix.
stos()

Purpose
Performs S-to-S conversion

Synopsis
stos(S, zRef, zNew, zy)

where $S$ is a scattering matrix, $z_{Ref}$ is a normalizing impedance, $z_{New}$ is a new
normalizing impedance, and $zy$ is an optional parameter for directing the
conversion through the Z- or Y-matrix. 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.

Example
$y = stos(S, 50, 75, 1)$

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param
Simulation and LSSP Simulation palettes in the Schematic window. There is no
explicit measurement component.

Defined in
$\text{HPEESOF_DIR/expressions/ael/network_fun.ael}$

See also
stoy(), stoz()

Description
This function changes the normalizing impedance of a scattering matrix.
stoy()

Purpose
Performs S-to-Y conversion

Synopsis
stoy(S, zRef)

where S is a scattering matrix of a 2-port network and zRef is a reference impedance.

Example
y = stoy(S, 50.0)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

See also
stoh(), stoz(), ytos()

Description
This measurement transforms a scattering matrix to an admittance matrix.
MeasEqn Function Reference

**stoz()**

**Purpose**
Performs S-to-Z conversion

**Synopsis**

\[
\text{stoz}(S, Z0)
\]

where \( S \) is a scattering matrix of a 2-port network and \( Z0 \) is a reference impedance.

**Example**

\[
z = \text{stoz}(S, 50)
\]

**Used in**
Small-signal and large-signal S-parameter simulations

**Available as measurement component?**
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

**Defined in**

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

**See also**

stoh(), stoy(), ztos()

**Description**
This measurement transforms a scattering matrix to an impedance matrix.
sum()

Purpose
Returns the sum

Synopsis
Y = sum(X)

Example
a = sum([1, 2, 3])
returns 6

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
max(), mean(), min()
sweep_dim()

Purpose
Returns the dimensionality of the data

Synopsis
sweep_dim(x)

Example
sweep_dim(1)
0
sweep_dim([1, 2, 3])
1

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
sweep_size()
sweep_size()

Purpose

Returns the sweep size of a data object

Synopsis

Y = sweep_size(X)

This function returns a vector with an entry corresponding to the length of each sweep.

Example

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

```matlab
a = sweep_size(S)
returns 10
size(S)
returns {2, 2}
```

Used in

Not applicable

Available as measurement component?

Not applicable

Defined in

Built in

See also

size(), sweep_dim()
\textbf{tan()}

\textbf{Purpose}

Returns the tangent of an integer or real number

\textbf{Synopsis}

\[ y = \tan(x) \]

where \( x \) is an integer or real number, in radians.

\textbf{Examples}

\begin{itemize}
  \item \( a = \tan(\pi/4) \)
  returns 1
  \item \( a = \tan(\pm\pi/2) \)
  returns \( \pm1.633E16 \)
\end{itemize}

\textbf{Used in}

Not applicable

\textbf{Available as measurement component?}

Not applicable

\textbf{Defined in}

Built in

\textbf{See also}

\( \cos(), \sin() \)
tanh()

**Purpose**
hyperbolic tangent

**Synopsis**
tanh(x)

**Example**
tanh(0)  
0  
tanh(1)  
0.761594  
tanh(-1)  
-0.761594

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**

sinh(), cosh()
trajectory()

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

Synopsis
Traj = trajectory(i_data, q_data)

where

i_data is the in-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental). This could be a baseband signal instead, but in either case it must be real valued versus time.

q_data is the quadrature-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental). This could be a baseband signal instead, but in either case it must be real valued versus time.

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

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

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

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

---

**Used in**
Trajectory diagram generation

**Available as measurement component?**
Equations listed under Description can be entered by means of a MeasEqn component in the Schematic window. There is no explicit trajectory measurement function.

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

**See also**
constellation(), const_evm()

**Description**
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.
transpose()

Purpose
Transposes a matrix

Synopsis
Y = transpose(y)

This function transposes a matrix, but does not perform a conjugate transpose for complex matrices.

Example
a = {{1, 2}, {3, 4}}
b = transpose(a)
returns {{1, 3}, {2, 4}}

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
None
ts()

Purpose
Performs a frequency-to-time transform

Synopsis
\[ ts(x, \text{tstart}, \text{tstop}, \text{numtpts}, \text{dim}, \text{windowType}, \text{windowConst}, \text{nptsspec}) \]
See detailed Description below.

Example
The following examples of ts assume that a harmonic balance simulation was performed with a fundamental frequency of 1 GHz and order = 8:
\begin{itemize}
  \item \( Y = ts(vOut) \) returns the time series (0, 20ps, ..., 2ns), 0 to 5 ns.
  \item \( Y = ts(vOut, 0, 1ns) \) returns the time series (0, 10ps, ..., 1ns).
  \item \( Y = ts(vOut, 0, 10ns, 201) \) returns the time series (0, 50ps, ..., 10ns).
  \item \( Y = ts(vOut, , , , , , , 3) \) returns the time series (0, 20ps, ..., 2ns), but only uses harmonics from 1 to 3 GHz.
\end{itemize}

Used in
Harmonic Balance and Circuit Envelope simulations

Available as measurement component?
Not applicable

Defined in
Built in

See also
fft(), fs(), fspot()

Description
ts(x) returns the time domain waveform from a frequency spectrum. When x is a multidimensional vector, the transform is evaluated for each vector in the specified dimension. For example, if x is a matrix, then ts(x) applies the transform to every row of the matrix. If x is three dimensional, then ts(x) is applied in the lowest dimension over the remaining two dimensions. The dimension over which to apply the transform
may be specified by dimension; the default is the lowest dimension (dimension=1).
ts() originated in MDS and is similar to vt().
x must be numeric. It will typically be data from a harmonic balance analysis.
By default, two cycles of the waveform are produced with 101 points, starting at time
zero, based on the lowest frequency in the input spectrum. These may be changed by
setting tstart, tstop, or numtpts.
All of the harmonics in the spectrum will be used to generate the time domain
waveform. When the higher-order harmonics are known not to contribute
significantly to the time domain waveform, only the first n harmonics may be
requested for the transform, by setting nptsspec = n.
The data to be transformed may be windowed by a window specified by windowType,
with an optional window constant windowConst. 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

windowType can be specified either by the number or by the name.
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.
**type()**

**Purpose**
Returns the type of the data

**Synopsis**
type(x)
Returns a string, which is one of “Integer”, “Real”, “Complex” or “String”

**Example**
type(1)
"Integer"
type(1i)
"Complex"
type("type")
"String"

**Used in**
Not applicable

**Available as measurement component?**
Not applicable

**Defined in**
Built in

**See also**
what()
unwrap()

Purpose
Unwraps phase

Synopsis
\[ y = \text{unwrap}(\text{phase}, \text{jump}) \]
where phase is a swept real variable and jump is the absolute jump. By default, jump is set to 180.

Example
unwrap(phase(S21))
unwrap(phaserad(S21), pi)

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built-in

See also
dev_lin_phase(), diff(), phase(), phasedeg(), phaserad(), ripple(), unwrap()

Description
This measurement unwraps a phase by changing an absolute jump greater than jump to its 2*jump complement.
vfc()

Purpose
Returns frequency-selective voltage

Synopsis
vfc(vPlus, vMinus, harm_freq_index)

where vPlus and vMinus are the voltages at the positive and negative terminals, and harm_freq_index is the harmonic index of the desired frequency. 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.

Example
The following example is for two tones in the harmonic balance controller:

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

Used in
Harmonic balance simulation

Available as measurement component?
Vfc

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

See also
ifc(), pfc()

Description
This measurement gives the RMS voltage value of one frequency-component of a harmonic balance waveform.
vfc_tran()

Purpose
Returns the transient frequency-selective voltage

Synopsis
vfc_tran(vPlus, vMinus, fundFreq, harmNum)

where vPlus and vMinus are the voltages at the positive and negative terminals, fundFreq is the fundamental frequency, and harmNum is the harmonic number of the fundamental.

Example
y=vfc_tran(vOut, 0, 1GHz, 1)

Used in
Transient simulations

Available as measurement component?
VfcTran

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

See also
ifc_tran(), pfc_tran()

Description
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).
volt_gain()

Purpose
Returns the voltage gain

Synopsis
y = volt_gain(S, Zs, Zl \{ Zref\})

where S is the $2 \times 2$ scattering matrix measured with equal terminations of Zref,
and Zs and Zl are the input and output impedances, respectively for voltage gain
calculation. Zref is the reference impedance for s-matrix measurement, set by
default to the value of 50 ohms.

Example
y = volt_gain(S, 50, 75)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
VoltGain

Defined in
circuit_fun.ael

See also
pwr_gain(), volt_gain_max()

Description
This measurement determines the ratio of the voltage across the load to the voltage
available from the source. The network-parameter transformation function “stos” can
be used to change the normalizing impedance of the scattering matrix.

\[
\text{volt\_gain} = \frac{S_{21}}{2} \cdot \frac{(1 - \Gamma_S) \cdot (1 + \Gamma_L)}{(S_{11} \cdot \Gamma_S - 1) \cdot (S_{22} \cdot \Gamma_L - 1) - S_{12} \cdot S_{21} \cdot \Gamma_S \cdot \Gamma_L}
\]  

(3-1)

Figure 3-1 illustrates the volt_gain measurement.
Figure 3-1. Signal Flow Diagram used for volt_gain Calculation
volt_gain_max()

Purpose

Returns the voltage gain at maximum power transfer

Synopsis

Y = volt_gain_max(S, Zs, Zl{, Zref})

where S is the 2 × 2 scattering matrix, and Zs and Zl are the input and output impedances, respectively. Zref is the reference impedance, set by default to the port impedances.

Example

y = volt_gain_max(S, 50, 75)

Used in

Small-signal and large-signal S-parameter simulations

Available as measurement component?

Not available

Defined in

$HPEESOF_DIR/expressions/acl/rf_system_fun.acl

See also

pwr_gain(), volt_gain()

Description

This measurement determines the ratio of the voltage across the load to the voltage available from the source at maximum power transfer. The network-parameter transformation function “stos” can be used to change the normalizing impedance of the scattering matrix.
vs()

Purpose
Attaches an independent to data

Synopsis
vs(dependent, independent)

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

Builds c with independent b, and dependent a.

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
indep()
vspec_tran()

Purpose
Returns the transient voltage spectrum

Synopsis
vspec_tran(vPlus, vMinus, fundFreq, numHarm)

where vPlus and vMinus are the voltages at the positive and negative terminals,
fundFreq is the fundamental frequency, and numHarm is the number of
harmonics of the fundamental frequency (positive integer value only).

Example
y=vspec_tran(v1, v2, 1GHz, 8)

Used in
Transient simulation

Available as measurement component?
VspecTran

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

See also
ispec_tran(), pspec_tran()

Description
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. numHarm is the number of harmonics of the fundamental frequency to be
included in the voltage spectrum.
vswr()

Purpose
Returns the voltage standing-wave ratio (VSWR)

Synopsis
vswr(Sii)

where Sii is the complex reflection coefficient.

Example
y=vswr(S11)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
VSWR

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

See also
yin(), zin()

Description
Given a complex reflection coefficient, this measurement returns the voltage standing wave ratio.
vt()

Purpose
Returns time-domain voltage waveform

Synopsis
vt(vPlus, vMinus, tmin, tmax, numOfPnts)

where vPlus and vMinus are the voltages at the positive and negative nodes, respectively, tmin and tmin are the start time and stop time, respectively, and numOfPts is the number of points (integer values only).

Example
y=vt(vOut, 0, 0, 10nsec, 201)

Used in
Harmonic balance simulation

Available as measurement component?
Vt

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

See also
it()

Description
This measurement converts a harmonic-balance voltage frequency spectrum to a time-domain voltage waveform. vt() originated in SIV and is similar to ts().
vt_tran()

Purpose
Returns the transient time-domain voltage waveform

Synopsis
Y = vt_tran(vPlus, vMinus)
    where vPlus and vMinus are the terminals across which the voltage is measured.

Example
y = vt_tran(v1, v2)

Used in
Transient simulations

Available as measurement component?
VtTran

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

See also
vt()

Description
This measurement produces a transient time-domain voltage waveform for specified nodes. vPlus and vMinus are the nodes across which the voltage is measured.
what()

Purpose
Returns size and type of data

Synopsis
what(x)

Example
None

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
type()

Description
what() is used to find out 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).
yield_sens()

Purpose
Returns the yield as a function of a design variable

Synopsis
yield_sens(pf_data, numBins)
where pf_data is a binary-valued scalar data set indicating the pass/fail status of
each value of a companion independent variable, and numBins is the number of
subintervals or bins used to measure yield_sens.

Example
yield_sens(pf_data)
yield_sens(pf_data, 20)

Used in
Monte Carlo simulation

Available as measurement component?
This function can only be entered by means of a Eqn component in the Data Display
window (or by choosing Insert > Equation, or clicking the Eqn button on the left side
of the window). There is no measurement component in schematic window.

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

See also
cdf(), histogram(), pdf()

Description
This function measures the yield as a function of a design variable. The default value
for numBins is set to \(\log(\text{numOfPts})/\log(2.0)\) by default. For more information and an
example refer to "Creating a Sensitivity Histogram" on page 3-18 in the Tuning,
Optimization and Statistical Design manual.
yin()

Purpose
Returns the port input admittance

Synopsis
yin(Sii, Z)

where Sii is a complex reflection coefficient and Z is a reference impedance.

Example
y=yin(S11, 50)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
Yin

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

See also
vswr(), zin()

Description
Given a reflection coefficient and the reference impedance, this measurement returns the input admittance looking into the measurement ports.
yopt()

Purpose
Returns optimum admittance for noise match

Synopsis
yopt(gammaOpt, zRef)

where gammaOpt is a optimum reflection coefficient and zRef is a reference impedance.

Example
y = yopt(Sopt, 50)

Used in
Small-signal S-parameter simulations

Available as measurement component?
Yopt

Defined in
Built in

See also
zopt()

Description
This complex measurement produces the optimum source admittance for noise matching. gammaOpt is the optimum reflection coefficient that must be presented at the input of the network to realize the minimum noise figure (NFmin).
ytoabcd()

Purpose
Performs Y-to-ABCD conversion

Synopsis
ytoabcd(Y)
    where Y is an admittance matrix.

Example
a = ytoabcd(Y)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

See also
abcdtoh(), htoabcd()

Description
This measurement transforms an admittance matrix of a 2-port network into a hybrid matrix.
ytoh()

Purpose
Performs Y-to-H conversion

Synopsis
ytoh(Y)

where Y is an admittance matrix.

Example
h = ytoh(Y)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

See also
htoy(), ytoabcd()

Description
This measurement transforms an admittance matrix of a 2-port network into a hybrid matrix.
ytos()

Purpose
Performs Y-to-S conversion

Synopsis
\[ S = \mathrm{ytos}(Y, z_{\text{Ref}}) \]

where \( Y \) is an admittance matrix and \( z_{\text{Ref}} \) is a reference impedance.

Example
\[ s = \mathrm{ytos}(Y, 50.0) \]

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

Defined in
\$\text{HPEESOF}\_\text{DIR}/\text{expressions/ael/network\_fun.ael}

See also
stoy(), ytoz()

Description
This measurement transforms an admittance matrix into a scattering matrix.
ytoz()

**Purpose**
Performs Y-to-Z conversion

**Synopsis**

\[ Z = ytoz(Y) \]

where \( Y \) is an admittance matrix

**Example**

\[ z = ytoz(Y) \]

**Used in**
Small-signal and large-signal S-parameter simulations

**Available as measurement component?**
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

**Defined in**

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

**See also**

ythos(), ztoy()

**Description**

This measurement transforms an admittance matrix to an impedance matrix.
zeros()

Purpose
Returns a matrix of zeros

Synopsis
Y = zeros(2)
Y = zeros(2, 3)

This is the zeros matrix. If only one argument is supplied, then a square matrix is returned. If two are supplied, then a matrix of zeros with size rows × cols is returned.

Example
a=zeros(2);
returns {{0, 0}, {0, 0}}

b=(2, 3)
returns {{0, 0, 0}, {0, 0, 0}}

Used in
Not applicable

Available as measurement component?
Not applicable

Defined in
Built in

See also
identity(), ones()
zin()

Purpose
Returns the port input impedance

Synopsis
zin(Sii, Z)

where Sii is a complex reflection coefficient and Z is a reference impedance.

Example
y=zin(S11, 50.0)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
Zin

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

See also
vswr(), yin()

Description
Given a reflection coefficient and the reference impedance, this measurement returns
the input impedance looking into the measurement ports.
zopt()

Purpose
Returns the optimum impedance for noise matching

Synopsis
zopt(gammaOpt, zRef)
where gammaOpt is an optimum reflection coefficient and zRef is a reference impedance.

Example
y = zopt(Sopt, 50)

Used in
Small-signal S-parameter simulations

Available as measurement component?
Zopt

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

See also
yopt()

Description
This complex measurement produces the optimum source impedance for noise matching. gammaOpt is the optimum reflection coefficient that must be presented at the input of a network to realize the minimum noise figure (NF min).
ztoabcd()

Purpose
Performs Z-to-ABCD conversion

Synopsis
ztoabcd(Z)

where Z is an impedance matrix.

Example
a = ztoabcd(Z)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

See also
abcdtoz(), ytoabcd(), ztoh()

Description
This measurement transforms an impedance matrix of a 2-port network into a chain (ABCD) matrix.
ztoh()

Purpose
Performs Z-to-H conversion

Synopsis
ztoh(Z)
    where Z is an impedance matrix.

Example
h = ztoh(Z)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

See also
htoz(), ytoh(), ztoabcd()

Description
This measurement transforms an impedance matrix of a 2-port network into a hybrid matrix.
MeasEqn Function Reference

ztos()

Purpose
Performs Z-to-S conversion

Synopsis
ztos(Z, zRef)
    where Z is an impedance matrix and zRef is a reference impedance.

Example
s = ztos(Z, 50.0)

Used in
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

See also
stoz(), ytos(), ztoy()

Description
This measurement transforms an impedance matrix to a scattering matrix.
ztoy()

Purpose
Performs Z-to-Y conversion

Synopsis
ztoy(Z)

where Z is an impedance matrix.

Example
y = ztoy(Z)

Used in...
Small-signal and large-signal S-parameter simulations

Available as measurement component?
This equation can be entered by means of a MeasEqn component in S_Param Simulation and LSSP Simulation palettes in the Schematic window. There is no explicit measurement component.

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

See also
stoz(), ytos(), ztoy()

Description
This measurement transforms an impedance matrix to an admittance matrix.
Chapter 4: Simulator Expression Reference

This chapter lists and describes the expressions (functions) that are available at simulation runtime. These expressions include mathematical functions such as those for matrix conversion, trigonometry, absolute value, and the like.

Unlike the measurement equations (MeasEqn) described in Chapter 3, MeasEqn Function Reference, these equations are used internally during simulation time, and are known as Simulator Expressions or sometimes as VarEqn expressions. These expressions can be entered into Advanced Design System by means of the VarEqn component or used in place of a parameter for any component: for example in a resistor, R=sin(5). These expressions are evaluated at the start of simulation. If a term is undefined at the start of simulation, such as R=S(1,1), where the results of S(1,1) will not be known until the simulation is complete, an error will be returned.

In Advanced Design System, the VarEqn component is available in the Data Items palette in an Analog/RF Systems Schematic window or from the Controllers palette in a Signal Processing Schematic window.

For information on the general use of VAR components in ADS, place a VAR component on a Schematic, double click it, and then click the Help button in the Component dialog box. Or from any ADS Window choose Help > Topics and Index > Components > Circuit Components > Introduction and Simulation Components > Chapter 1, Introduction > VAR.

In this chapter you will find:

- A list of the Simulator Expressions.
- A list of simulator variables and constants.
- A list of mathematical operators and hierarchy that can be used with these expressions.

How to Enter Simulator Expressions in ADS

The basic ways to enter the equations or expressions that are available at simulation runtime are as follows:

- Place and setup a VarEqn component.
- Place any component and enter a function in place of a component parameter.
• Place and set up a FDD (frequency-defined device) or SDD (symbolically-defined device).

## Simulator Expressions

Simulator Expressions are listed in Table 4-1 and Table 4-2. The same expressions are provided in both tables; however, the expressions in Table 4-1 have been sorted alphabetically. Each expression is accompanied by a brief description.

### Table 4-1. Simulator Expressions (Sorted Alphabetically)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(A0)</td>
<td>absolute value function</td>
</tr>
<tr>
<td>access_all_data(...)</td>
<td>datafile indep+dep lookup/interpolation function</td>
</tr>
<tr>
<td>access_data(...)</td>
<td>datafile dependents' lookup/interpolation function</td>
</tr>
<tr>
<td>acos(A0)</td>
<td>Arc-cosine(x) for x real and -1 &lt;= x &lt;= 1</td>
</tr>
<tr>
<td></td>
<td>Does not accept/generate complex numbers.</td>
</tr>
<tr>
<td></td>
<td>Warning messages are generated whenever the input lies outside the specified</td>
</tr>
<tr>
<td></td>
<td>range. Should only be used in nonlinear model functions.</td>
</tr>
<tr>
<td>acosh(A0)</td>
<td>Arc-hyperbolic-cosine(x) for x real and x &gt;= 1</td>
</tr>
<tr>
<td></td>
<td>Does not accept/generate complex numbers.</td>
</tr>
<tr>
<td></td>
<td>Warning messages are generated whenever the input lies outside the specified</td>
</tr>
<tr>
<td></td>
<td>range. Should only be used in nonlinear model functions.</td>
</tr>
<tr>
<td>amp_harm_coef(A0, A1, A2)</td>
<td>Amplifier polynomial generator function</td>
</tr>
<tr>
<td>arcsinh(A0)</td>
<td>arcsinh function</td>
</tr>
<tr>
<td>arctan(A0)</td>
<td>arctan function</td>
</tr>
<tr>
<td>asin(A0)</td>
<td>Arc-sine(x) for x real and -1 &lt;= x &lt;= 1</td>
</tr>
<tr>
<td></td>
<td>Does not accept/generate complex numbers.</td>
</tr>
<tr>
<td></td>
<td>Warning messages are generated whenever the input lies outside the specified</td>
</tr>
<tr>
<td></td>
<td>range. Should only be used in nonlinear model functions.</td>
</tr>
<tr>
<td>asinh(A0)</td>
<td>Arc-hyperbolic-sine function</td>
</tr>
</tbody>
</table>
### Table 4-1. Simulator Expressions (Sorted Alphabetically)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>atan(y,x)</td>
<td>arctangent function (two real arguments). Same as atan2 function</td>
</tr>
<tr>
<td>atan2(y, x)</td>
<td>arctangent function (two real arguments)</td>
</tr>
<tr>
<td>atanh(A0)</td>
<td>Arc-hyperbolic-tangent(x) for x real and -1 &lt;= x &lt;= 1</td>
</tr>
<tr>
<td></td>
<td>Does not accept/generate complex numbers. Warning messages are generated</td>
</tr>
<tr>
<td></td>
<td>whenever the input lies outside the specified range. Should only be used</td>
</tr>
<tr>
<td></td>
<td>in nonlinear model functions.</td>
</tr>
<tr>
<td>awg_dia(A0)</td>
<td>wire gauge to diameter in meters</td>
</tr>
<tr>
<td>bin(A0)</td>
<td>function convert a binary to integer</td>
</tr>
<tr>
<td>bitseq(time, [clockfreq, trise, tfall, vlow, vhigh, bitseq] )</td>
<td>bit sequence function</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>ceil function for real numbers</td>
</tr>
<tr>
<td>coef_count(A0)</td>
<td>Count the number of coefficients</td>
</tr>
<tr>
<td>conj(x)</td>
<td>complex-conjugate function</td>
</tr>
<tr>
<td>complex(x, y)</td>
<td>real-to-complex conversion function</td>
</tr>
<tr>
<td>compute_poly_coef(A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12)</td>
<td>Compute polynomial coefficients for user specified nonlinearities such TOI, Psat etc</td>
</tr>
<tr>
<td>cos (x)</td>
<td>cosine function</td>
</tr>
<tr>
<td>cosh(x)</td>
<td>hyperbolic cosine function</td>
</tr>
<tr>
<td>cos_pulse(time, [low, high, delay, rise, fall, width, period] )</td>
<td>periodic cosine shaped pulse function</td>
</tr>
<tr>
<td>cot(x)</td>
<td>cotangent function</td>
</tr>
<tr>
<td>coth(x)</td>
<td>hyperbolic cotangent function</td>
</tr>
<tr>
<td>cpx_gain_poly(A0, A1, A2)</td>
<td>Compute complex gain for given polynomial coefficients</td>
</tr>
<tr>
<td>ctof(A0)</td>
<td>convert Celsius to Fahrenheit</td>
</tr>
<tr>
<td>ctk(A0)</td>
<td>convert Celsius to Kelvin</td>
</tr>
<tr>
<td>cxform(A0, A1, A2)</td>
<td>transform complex data</td>
</tr>
<tr>
<td>damped_sin(time, [offset, amplitude, freq, delay, damping, phase] )</td>
<td>damped sin function</td>
</tr>
<tr>
<td>d_atan2(a, b, c, d)</td>
<td>derivative of atan2()</td>
</tr>
</tbody>
</table>
### Table 4-1. Simulator Expressions (Sorted Alphabetically)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db(x)</td>
<td>decibel function</td>
</tr>
<tr>
<td>dbm(A0, A1)</td>
<td>convert voltage and impedance into dBm</td>
</tr>
<tr>
<td>dbmtoa(A0, A1)</td>
<td>convert dBm and impedance into short circuit current</td>
</tr>
<tr>
<td>dbmtov(A0, A1)</td>
<td>convert dBm and impedance into open circuit voltage</td>
</tr>
<tr>
<td>dbmtow(A0)</td>
<td>convert dBm to watts; takes one argument, e.g., dbmtow (-10) returns 1E-4 watts</td>
</tr>
<tr>
<td>dbpolar(x, y)</td>
<td>(dB,angle)-to-rectangular conversion function</td>
</tr>
<tr>
<td>dbwtow(A0)</td>
<td>convert dBW to watts; see dbmtow for more information</td>
</tr>
<tr>
<td>deembed(A0)</td>
<td>de-embedding function</td>
</tr>
<tr>
<td>deg(x)</td>
<td>radian-to-degree conversion function</td>
</tr>
<tr>
<td>delay(A0)</td>
<td>group delay function</td>
</tr>
<tr>
<td>dep_data(A0, A1, [A2])</td>
<td>dependent variable value</td>
</tr>
<tr>
<td>deriv(A0, A1)</td>
<td>derivative function</td>
</tr>
<tr>
<td>dphase(A0, A1)</td>
<td>Continuous phase difference (radians) between A0 and A1</td>
</tr>
<tr>
<td>dsexpr(A0, A1)</td>
<td>Evaluate a dataset expression to an hpvar</td>
</tr>
<tr>
<td>dstoarray(A0, [A1])</td>
<td>Convert an hpvar to an array</td>
</tr>
<tr>
<td>echo(A0)</td>
<td>echo-arguments function</td>
</tr>
<tr>
<td>erf_pulse(time, [low, high, delay, rise, fall, width, period])</td>
<td>periodic error function shaped pulse function</td>
</tr>
<tr>
<td>eval_poly(A0, A1, A2)</td>
<td>polynomial evaluation function</td>
</tr>
<tr>
<td>exp(x)</td>
<td>exponential function</td>
</tr>
<tr>
<td>exp_pulse(time, [low, high, delay1, tau1, delay2, tau2])</td>
<td>exponential pulse function</td>
</tr>
<tr>
<td>floor(x)</td>
<td>floor function for real numbers</td>
</tr>
<tr>
<td>fmod(x)</td>
<td>real number modulo/remainder</td>
</tr>
<tr>
<td>fread(A0)</td>
<td>raw-file reading function</td>
</tr>
<tr>
<td>freq_mult_coef(A0)</td>
<td>frequency multiplier polynomial generator function</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td><code>freq_mult_poly(A0, A1)</code></td>
<td>frequency multiplier polynomial evaluation function</td>
</tr>
<tr>
<td><code>ftoc(A0)</code></td>
<td>convert Fahrenheit to Celsius</td>
</tr>
<tr>
<td><code>ftok(A0)</code></td>
<td>convert Fahrenheit to Kelvin</td>
</tr>
<tr>
<td><code>generate_gmsk_iq_spectra(A0, A1, A2, A3, A4, A5, A6)</code></td>
<td>generate the gmsk <code>i</code> or <code>q</code></td>
</tr>
<tr>
<td><code>generate_gmsk_pulse_spectra(A0, A1, A2, A3, A4, A5, A6, A7)</code></td>
<td>generate a gmsk pulse train spectra</td>
</tr>
<tr>
<td><code>generate_piqpsk_spectra(A0, A1, A2, A3, A4, A5, A6)</code></td>
<td>generate a pi/4 QPSK pulse train spectra</td>
</tr>
<tr>
<td><code>generate_pulse_train_spectra(A0, A1, A2, A3, A4, A5)</code></td>
<td>generate a pulse train spectra</td>
</tr>
<tr>
<td><code>generate_qam16_spectra(A0, A1, A2, A3, A4, A5, A6)</code></td>
<td>generate a 16-QAM pulse train spectra</td>
</tr>
<tr>
<td><code>generate_qpsk_pulse_spectra(A0, A1, A2, A3, A4, A5, A6)</code></td>
<td>generate a QPSK pulse train spectra</td>
</tr>
<tr>
<td><code>gcdata_to_poly(A0, A1)</code></td>
<td>Fit gain compression data to a polynomial</td>
</tr>
<tr>
<td><code>get_array_size(A0)</code></td>
<td>Get the size of the array</td>
</tr>
<tr>
<td><code>get_attribute(...)</code></td>
<td>value of attribute of a set of data</td>
</tr>
<tr>
<td><code>get_block(A0, A1)</code></td>
<td>HPVar tree from block name function</td>
</tr>
<tr>
<td><code>get_fund_freq(A0)</code></td>
<td>Get the frequency associated with a specified fundamental index</td>
</tr>
<tr>
<td><code>get_max_points(A0, A1)</code></td>
<td>maximum points of independent variable</td>
</tr>
<tr>
<td><code>hypot(A0, A1)</code></td>
<td>hypotenuse function</td>
</tr>
<tr>
<td><code>i(...)</code></td>
<td>current function</td>
</tr>
<tr>
<td><code>ilsb(...)</code></td>
<td>small-signal lower-sideband current function</td>
</tr>
<tr>
<td><code>imag(x)</code></td>
<td>imaginary-part function</td>
</tr>
<tr>
<td><code>impulse(...)</code></td>
<td>impulse(CpxFunction(0), Npts, TimeStep, CenterFreq, Window) returns impulse array</td>
</tr>
<tr>
<td><code>imt_hpvar_to_array(A0, A1, A2, A3)</code></td>
<td>Convert IMT hpvar data to an array</td>
</tr>
<tr>
<td><code>imt_hbdata_to_array(A0, A1, A2, A3, A4, A5)</code></td>
<td>Convert 2-tone HB data to an IMT array</td>
</tr>
<tr>
<td><code>index(A0, A1, [A2, A3])</code></td>
<td>get index of name in array</td>
</tr>
<tr>
<td><code>innerprod(...)</code></td>
<td>inner-product function</td>
</tr>
</tbody>
</table>
### Table 4-1. Simulator Expressions (Sorted Alphabetically)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inoise(A0)</td>
<td>noise current function</td>
</tr>
<tr>
<td>int(x)</td>
<td>convert-to-integer function</td>
</tr>
<tr>
<td>internal_generate_gmsk_iq_spectra(A0, A1, A2, A3, A4, A5, A6, A7)</td>
<td>internal generate the gmsk <code>i</code> or <code>q</code></td>
</tr>
<tr>
<td>internal_generate_gmsk_pulse_spectra(A0, A1, A2, A3, A4, A5, A6, A7)</td>
<td>internal generate a gmsk pulse train spectra</td>
</tr>
<tr>
<td>internal_generate_piqpsk_spectra(A0, A1, A2, A3, A4, A5, A6, A7)</td>
<td>internal generate a pi/4 QPSK pulse train spectra</td>
</tr>
<tr>
<td>internal_generate_pULSE_train_spectra(A0, A1, A2, A3, A4, A5, A6, A7)</td>
<td>internal generate a pulse train spectra</td>
</tr>
<tr>
<td>internal_generate_qam16_spectra(A0, A1, A2, A3, A4, A5, A6, A7)</td>
<td>internal generate a 16-QAM pulse train spectra</td>
</tr>
<tr>
<td>internal_generate_qpsk_pulse_spectra(A0, A1, A2, A3, A4, A5, A6, A7)</td>
<td>internal generate a QPSK pulse train spectra</td>
</tr>
<tr>
<td>internal_get_fund_freq(A0, A1)</td>
<td>internal function to get frequency for a specified fundamental index</td>
</tr>
<tr>
<td>internal_window(A0, A1, A2, A3, A4, A5)</td>
<td>internal spectral windowing function</td>
</tr>
<tr>
<td>interp(...)</td>
<td>somewhat generalized scalar interpolation function</td>
</tr>
<tr>
<td>interp1(A0, A1)</td>
<td>interpolation function--one independent variable</td>
</tr>
<tr>
<td>interp2(A0, A1, A2)</td>
<td>interpolation function--two independent variables</td>
</tr>
<tr>
<td>interp3(A0, A1, A2, A3)</td>
<td>interpolation function--three independent variables</td>
</tr>
<tr>
<td>interp4(A0, A1, A2, A3, A4)</td>
<td>interpolation function--four independent variables</td>
</tr>
<tr>
<td>iss(...)</td>
<td>small-signal current function</td>
</tr>
<tr>
<td>itob(A0, [A1])</td>
<td>convert integer to binary</td>
</tr>
<tr>
<td>iusb(...)</td>
<td>small-signal upper-sideband current function</td>
</tr>
<tr>
<td>jn(A0, A1)</td>
<td>Bessel function</td>
</tr>
<tr>
<td>ktoC(A0)</td>
<td>convert Kelvin to Celsius</td>
</tr>
<tr>
<td>ktoF(A0)</td>
<td>convert Kelvin to Fahrenheit</td>
</tr>
<tr>
<td>length(A0)</td>
<td>returns number of elements in array</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>lfsr(A0, A1)</td>
<td>lfsr(taps, seed) returns a string containing complete sequence</td>
</tr>
<tr>
<td>limit_warn([A0, A1, A2, A3, A4])</td>
<td>limit, default and warn function</td>
</tr>
<tr>
<td>list(...)</td>
<td></td>
</tr>
<tr>
<td>ln(x)</td>
<td>natural log function</td>
</tr>
<tr>
<td>log(x)</td>
<td>log base 10 function</td>
</tr>
<tr>
<td>log10(x)</td>
<td>log base 10 function. Same as log(x)</td>
</tr>
<tr>
<td>log_amp(A0, A1, A2, A3)</td>
<td>successive detection logarithmic amplifier</td>
</tr>
<tr>
<td>log_amp_cas(A0, A1, A2, A3)</td>
<td>true logarithmic amplifier</td>
</tr>
<tr>
<td>lookup(...)</td>
<td>data lookup function</td>
</tr>
<tr>
<td>mag(x)</td>
<td>magnitude function</td>
</tr>
<tr>
<td>makearray(...)</td>
<td>(1:real</td>
</tr>
<tr>
<td>max(x, y)</td>
<td>maximum function</td>
</tr>
<tr>
<td>min(x, y)</td>
<td>minimum function</td>
</tr>
<tr>
<td>miximt_coef(A0, A1, A2, A3, A4)</td>
<td>Mixer IMT polynomial generator function</td>
</tr>
<tr>
<td>miximt_poly(A0, A1, A2, A3, A4)</td>
<td>Mixer IMT polynomial evaluation function</td>
</tr>
<tr>
<td>multi_freq(time, amplitude, freq1, freq2, n, [seed])</td>
<td>multifrequency function</td>
</tr>
<tr>
<td>names(A0, A1)</td>
<td>array of names of indepVars and/or depVars in dataset</td>
</tr>
<tr>
<td>nf(A0)</td>
<td>noise figure function</td>
</tr>
<tr>
<td>norm(A0)</td>
<td>norm function</td>
</tr>
<tr>
<td>phase(x)</td>
<td>phase (in degrees) function</td>
</tr>
<tr>
<td>phasedeg(x)</td>
<td>phase (in degrees) function</td>
</tr>
<tr>
<td>phase_noise_pwl(...)</td>
<td>piecewise-linear function for computing phase noise</td>
</tr>
<tr>
<td>phaserad(x)</td>
<td>phase (in radians) function</td>
</tr>
<tr>
<td>polar(x, y)</td>
<td>polar-to-rectangular conversion function</td>
</tr>
<tr>
<td>polarcpx(...)</td>
<td>polar to rectangular conversion function</td>
</tr>
<tr>
<td>pow(A0,A1)</td>
<td>power function, returns A0^A1</td>
</tr>
</tbody>
</table>
### Table 4-1. Simulator Expressions (Sorted Alphabetically)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pulse(time, [low, high, delay, rise, fall, width, period])</td>
<td>periodic pulse function</td>
</tr>
<tr>
<td>pwl(...)</td>
<td>piecewise-linear function</td>
</tr>
<tr>
<td></td>
<td>( y = \text{pwl}(x, x_1, y_1, x_2, y_2, ...) )</td>
</tr>
<tr>
<td></td>
<td>( x = ) name of the independent variable.</td>
</tr>
<tr>
<td></td>
<td>( x_k, y_k = ) pairs of known values.</td>
</tr>
<tr>
<td></td>
<td>( x ) can be the name of any swept variable like the internal variables</td>
</tr>
<tr>
<td></td>
<td>( ) time or freq, or any other variable swept with a ParamSweep</td>
</tr>
<tr>
<td></td>
<td>( ) controller. If the piecewise linear waveform is to be repeated for</td>
</tr>
<tr>
<td></td>
<td>( ) several cycles, the ( \text{pwlr}() ) function can be used.</td>
</tr>
<tr>
<td></td>
<td>Example: see component VtPWL.</td>
</tr>
<tr>
<td>pwlr(...)</td>
<td>piecewise-linear-repeated function</td>
</tr>
<tr>
<td></td>
<td>Repeats the ( \text{pwl}() ) function for several cycles.</td>
</tr>
<tr>
<td></td>
<td>( y = \text{pwlr}(x, N_{cycles}, x_1, y_1, x_2, y_2, ...) )</td>
</tr>
<tr>
<td></td>
<td>( N_{cycles} = ) the number of cycles to be repeated.</td>
</tr>
<tr>
<td></td>
<td>Example: see component VtPWL.</td>
</tr>
<tr>
<td>qinterp(A0, A1)</td>
<td>quick and dirty interpolation function</td>
</tr>
<tr>
<td>rad(x)</td>
<td>degree-to-radian conversion function</td>
</tr>
<tr>
<td>ramp(A0)</td>
<td>ramp function</td>
</tr>
<tr>
<td>rawtoarray(...)</td>
<td>rawfile pointer to sym array conversion function</td>
</tr>
<tr>
<td>readdata(...)</td>
<td>library or rawfile reading routine</td>
</tr>
<tr>
<td>read_data(...)</td>
<td>read_data(&quot;file</td>
</tr>
<tr>
<td>readlib(A0, A1, A2, A3)</td>
<td>rawfile-from-library reading routine</td>
</tr>
<tr>
<td>read_lib(...)</td>
<td>read_lib(&quot;libName&quot;, &quot;item&quot;, &quot;fileType&quot;)</td>
</tr>
<tr>
<td>readraw(A0, A1, A2)</td>
<td>rawfile reading routine</td>
</tr>
<tr>
<td>real(x)</td>
<td>real-part function</td>
</tr>
<tr>
<td>rect(A0, A1, A2)</td>
<td>rectangular pulse function</td>
</tr>
<tr>
<td>rem(...)</td>
<td>remainder function. Examples: ( \text{rem}(10, 4) ) returns 2, because 10</td>
</tr>
<tr>
<td></td>
<td>divided by 4 is 2 with a remainder of 2. ( \text{rem}(5, 5) ) returns 0.</td>
</tr>
<tr>
<td></td>
<td>( \text{rem}(x; y) = x - \text{int}(x/y)*y ). If either ( x ) or ( y</td>
</tr>
<tr>
<td></td>
<td>is a complex number, it’s replaced by its mag() value.</td>
</tr>
</tbody>
</table>
### Table 4-1. Simulator Expressions (Sorted Alphabetically)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ripple(x, y, z, v)</td>
<td>ripple(amplitude, intercept, period, variable) sinusoidal ripple function</td>
</tr>
<tr>
<td>rms(...)</td>
<td>root-mean-square function</td>
</tr>
<tr>
<td>rpsmooth(A0)</td>
<td>rectangular-to-polar smoothing function</td>
</tr>
<tr>
<td>scalearray(A0, A1)</td>
<td>scalar times a vector (array) function</td>
</tr>
<tr>
<td>sens(A0, A1)</td>
<td>sensitivity function</td>
</tr>
<tr>
<td>setDT(A0)</td>
<td>Turns on discrete time transient mode (returns argument)</td>
</tr>
<tr>
<td>sffm(time, [offset, amplitude, carrier_freq, mod_index, signal_freq])</td>
<td>signal frequency FM</td>
</tr>
<tr>
<td>sgn(A0)</td>
<td>signum function</td>
</tr>
<tr>
<td>sin(x)</td>
<td>sine function</td>
</tr>
<tr>
<td>sinc(A0)</td>
<td>sin(x)/x function</td>
</tr>
<tr>
<td>sinh(x)</td>
<td>hyperbolic sine function</td>
</tr>
<tr>
<td>spectrum(...)</td>
<td>spectrum(function(0), Npts, Period, Delay, Window) returns spectral array</td>
</tr>
<tr>
<td>sprintf(A0, A1)</td>
<td>formatted print utility</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>square root function</td>
</tr>
<tr>
<td>step(A0)</td>
<td>step function</td>
</tr>
<tr>
<td>strcat(...)</td>
<td>string concatenation utility</td>
</tr>
<tr>
<td>stypexform(A0, A1, A2, A3)</td>
<td>Y, Z, H, G, to S matrix transform</td>
</tr>
<tr>
<td>sym_set(A0, A1)</td>
<td>set sym variable to a given value</td>
</tr>
<tr>
<td>system(A0)</td>
<td>UNIX system call function</td>
</tr>
<tr>
<td>tan(x)</td>
<td>tangent function</td>
</tr>
<tr>
<td>tanh(x)</td>
<td>hyperbolic tangent function</td>
</tr>
<tr>
<td>thd(A0)</td>
<td>total-harmonic-distortion function</td>
</tr>
<tr>
<td>toi(A0)</td>
<td>third-order-intercept function</td>
</tr>
<tr>
<td>transform(A0, A1, A2, [A3, A4])</td>
<td>transform complex depVars in dataset</td>
</tr>
<tr>
<td>v(...)</td>
<td>voltage function</td>
</tr>
<tr>
<td>value(A0)</td>
<td>print-value function</td>
</tr>
</tbody>
</table>
The Simulator Expressions listed in Table 4-2 are the same expressions listed in Table 4-1; however, they have been sorted by functionality.

### Table 4-2. Simulator Expressions (Sorted by Functionality)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{d_atan2(a, b, c, d)}</td>
<td>derivative of \texttt{atan2()}</td>
</tr>
<tr>
<td>\texttt{sin(x)}</td>
<td>sine function</td>
</tr>
<tr>
<td>\texttt{cos(x)}</td>
<td>cosine function</td>
</tr>
<tr>
<td>\texttt{tan(x)}</td>
<td>tangent function</td>
</tr>
<tr>
<td>\texttt{cot(x)}</td>
<td>cotangent function</td>
</tr>
<tr>
<td>\texttt{sinh(x)}</td>
<td>hyperbolic sine function</td>
</tr>
<tr>
<td>\texttt{cosh(x)}</td>
<td>hyperbolic cosine function</td>
</tr>
<tr>
<td>\texttt{tanh(x)}</td>
<td>hyperbolic tangent function</td>
</tr>
<tr>
<td>\texttt{coth(x)}</td>
<td>hyperbolic cotangent function</td>
</tr>
<tr>
<td>\texttt{exp(x)}</td>
<td>exponential function</td>
</tr>
<tr>
<td>\texttt{ln(x)}</td>
<td>natural log function</td>
</tr>
<tr>
<td>\texttt{log(x)}</td>
<td>log base 10 function</td>
</tr>
<tr>
<td>\texttt{log10(x)}</td>
<td>log base 10 function. Same as \texttt{log(x)}</td>
</tr>
<tr>
<td>\texttt{ceil(x)}</td>
<td>cell function for real numbers</td>
</tr>
<tr>
<td>\texttt{floor(x)}</td>
<td>floor function for real numbers</td>
</tr>
<tr>
<td>\texttt{fmod(x)}</td>
<td>real number modulo/remainder</td>
</tr>
<tr>
<td>\texttt{hypot(A0,A1)}</td>
<td>hypotenuse function</td>
</tr>
<tr>
<td>\texttt{pow(A0,A1)}</td>
<td>power function, returns \texttt{A0}^\texttt{A1}</td>
</tr>
</tbody>
</table>

---

4-10 Simulator Expressions
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqrt(x)</td>
<td>square root function</td>
</tr>
<tr>
<td>conj(x)</td>
<td>complex-conjugate function</td>
</tr>
<tr>
<td>mag(x)</td>
<td>magnitude function</td>
</tr>
<tr>
<td>phaserad(x)</td>
<td>phase (in radians) function</td>
</tr>
<tr>
<td>imag(x)</td>
<td>imaginary-part function</td>
</tr>
<tr>
<td>real(x)</td>
<td>real-part function</td>
</tr>
<tr>
<td>phasedeg(x)</td>
<td>phase (in degrees) function</td>
</tr>
<tr>
<td>phase(x)</td>
<td>phase (in degrees) function</td>
</tr>
<tr>
<td>deg(x)</td>
<td>radian-to-degree conversion function</td>
</tr>
<tr>
<td>rad(x)</td>
<td>degree-to-radian conversion function</td>
</tr>
<tr>
<td>int(x)</td>
<td>convert-to-integer function</td>
</tr>
<tr>
<td>atan2(y, x)</td>
<td>arctangent function (two real arguments)</td>
</tr>
<tr>
<td>complex(x, y)</td>
<td>real-to-complex conversion function</td>
</tr>
<tr>
<td>polar(x, y)</td>
<td>polar-to-rectangular conversion function</td>
</tr>
<tr>
<td>dbpolar(x, y)</td>
<td>(dB,angle)-to-rectangular conversion function</td>
</tr>
<tr>
<td>vswrpolar(x, y)</td>
<td>(VSWR,angle)-to-rectangular conversion function</td>
</tr>
<tr>
<td>ripple(x, y, z, v)</td>
<td>ripple(amplitude, intercept, period, variable)</td>
</tr>
<tr>
<td></td>
<td>sinusoidal ripple function</td>
</tr>
<tr>
<td>db(x)</td>
<td>decibel function</td>
</tr>
<tr>
<td>max(x, y)</td>
<td>maximum function</td>
</tr>
<tr>
<td>min(x, y)</td>
<td>minimum function</td>
</tr>
<tr>
<td>spectrum(...)</td>
<td>spectrum(function(0), Npts, Period, Delay, Window)</td>
</tr>
<tr>
<td></td>
<td>returns spectral array</td>
</tr>
<tr>
<td>impulse(...)</td>
<td>impulse(CpxFunction(0), Npts, TimeStep, CenterFreq, Window)</td>
</tr>
<tr>
<td></td>
<td>returns impulse array</td>
</tr>
<tr>
<td>abs(A0)</td>
<td>absolute value function</td>
</tr>
<tr>
<td>sgn(A0)</td>
<td>signum function</td>
</tr>
<tr>
<td>arcsinh(A0)</td>
<td>arcsinh function</td>
</tr>
<tr>
<td>arctan(A0)</td>
<td>arctan function</td>
</tr>
</tbody>
</table>
Table 4-2. Simulator Expressions (Sorted by Functionality)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acos(A0)</td>
<td>Arc-cosine(x) for x real and -1 &lt;= x &lt;= 1</td>
</tr>
<tr>
<td></td>
<td>Does not accept/generate complex numbers.</td>
</tr>
<tr>
<td></td>
<td>Warning messages are generated whenever the input lies outside the specified range.</td>
</tr>
<tr>
<td></td>
<td>Should only be used in nonlinear model functions.</td>
</tr>
<tr>
<td>asin(A0)</td>
<td>Arc-sine(x) for x real and -1 &lt;= x &lt;= 1</td>
</tr>
<tr>
<td></td>
<td>Does not accept/generate complex numbers.</td>
</tr>
<tr>
<td></td>
<td>Warning messages are generated whenever the input lies outside the specified range.</td>
</tr>
<tr>
<td></td>
<td>Should only be used in nonlinear model functions.</td>
</tr>
<tr>
<td>atan(y,x)</td>
<td>Arctangent function (two real arguments).</td>
</tr>
<tr>
<td></td>
<td>Same as atan2 function</td>
</tr>
<tr>
<td>acosh(A0)</td>
<td>Arc-hyperbolic-cosine(x) for x real and x &gt;= 1</td>
</tr>
<tr>
<td></td>
<td>Does not accept/generate complex numbers.</td>
</tr>
<tr>
<td></td>
<td>Warning messages are generated whenever the input lies outside the specified range.</td>
</tr>
<tr>
<td></td>
<td>Should only be used in nonlinear model functions.</td>
</tr>
<tr>
<td>asinh(A0)</td>
<td>Arc-hyperbolic-sine function</td>
</tr>
<tr>
<td>atanh(A0)</td>
<td>Arc-hyperbolic-tangent(x) for x real and -1 &lt;= x &lt;= 1</td>
</tr>
<tr>
<td></td>
<td>Does not accept/generate complex numbers.</td>
</tr>
<tr>
<td></td>
<td>Warning messages are generated whenever the input lies outside the specified range.</td>
</tr>
<tr>
<td></td>
<td>Should only be used in nonlinear model functions.</td>
</tr>
<tr>
<td>jn(A0, A1)</td>
<td>Bessel function</td>
</tr>
<tr>
<td>i(...)</td>
<td>Current function</td>
</tr>
<tr>
<td>deembed(A0)</td>
<td>De-embedding function</td>
</tr>
<tr>
<td>delay(A0)</td>
<td>Group delay function</td>
</tr>
<tr>
<td>deriv(A0, A1)</td>
<td>Derivative function</td>
</tr>
<tr>
<td>fread(A0)</td>
<td>Raw-file reading function</td>
</tr>
<tr>
<td>interp1(A0, A1)</td>
<td>Interpolation function—one independent variable</td>
</tr>
<tr>
<td>interp2(A0, A1, A2)</td>
<td>Interpolation function--two independent variables</td>
</tr>
</tbody>
</table>
### Table 4-2. Simulator Expressions (Sorted by Functionality)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interp3(A0, A1, A2, A3)</td>
<td>interpolation function--three independent variables</td>
</tr>
<tr>
<td>interp4(A0, A1, A2, A3, A4)</td>
<td>interpolation function--four independent variables</td>
</tr>
<tr>
<td>interp(...)</td>
<td>somewhat generalized scalar interpolation function</td>
</tr>
<tr>
<td>lookup(...)</td>
<td>data lookup function</td>
</tr>
<tr>
<td>access_data(...)</td>
<td>datafile dependents' lookup/interpolation function</td>
</tr>
<tr>
<td>access_all_data(...)</td>
<td>datafile indep+dep lookup/interpolation function</td>
</tr>
<tr>
<td>polarcpx(...)</td>
<td>polar to rectangular conversion function</td>
</tr>
<tr>
<td>scalearray(A0, A1)</td>
<td>scalar times a vector (array) function</td>
</tr>
<tr>
<td>qinterp(A0, A1)</td>
<td>quick and dirty interpolation function</td>
</tr>
<tr>
<td>ramp(A0)</td>
<td>ramp function</td>
</tr>
<tr>
<td>step(A0)</td>
<td>step function</td>
</tr>
<tr>
<td>setDT(A0)</td>
<td>Turns on discrete time transient mode (returns argument)</td>
</tr>
<tr>
<td>readraw(A0, A1, A2)</td>
<td>rawfile reading routine</td>
</tr>
<tr>
<td>readlib(A0, A1, A2, A3)</td>
<td>rawfile-from-library reading routine</td>
</tr>
<tr>
<td>readdata(...)</td>
<td>library or rawfile reading routine</td>
</tr>
<tr>
<td>rect(A0, A1, A2)</td>
<td>rectangular pulse function</td>
</tr>
<tr>
<td>rem(...)</td>
<td>remainder function. Examples: rem(10,4) returns 2, because 10 divided by 4 is 2 with a remainder of 2. rem(5,5) returns 0. rem(x,y) = x - int(x/y)*y. If either x or y is a complex number, it’s replaced by its mag() value.</td>
</tr>
<tr>
<td>limit_warn(...)</td>
<td>limit, default and warn function</td>
</tr>
<tr>
<td>awg_dia(A0)</td>
<td>wire gauge to diameter in meters</td>
</tr>
<tr>
<td>rpsmooth(A0)</td>
<td>rectangular-to-polar smoothing function</td>
</tr>
<tr>
<td>sens(A0, A1)</td>
<td>sensitivity function</td>
</tr>
<tr>
<td>sinc(A0)</td>
<td>sin(x)/x function</td>
</tr>
<tr>
<td>sprintf(A0, A1)</td>
<td>formatted print utility</td>
</tr>
</tbody>
</table>
### Table 4-2. Simulator Expressions (Sorted by Functionality)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strcat(...)</td>
<td>string concatenation utility</td>
</tr>
<tr>
<td>system(A0)</td>
<td>UNIX system call function</td>
</tr>
<tr>
<td>rawtoarray(...)</td>
<td>rawfile pointer to sym array conversion function</td>
</tr>
<tr>
<td>makearray(...)</td>
<td>(1:real</td>
</tr>
<tr>
<td>list(...)</td>
<td></td>
</tr>
<tr>
<td>length(A0)</td>
<td>returns number of elements in array</td>
</tr>
<tr>
<td>v(...)</td>
<td>voltage function</td>
</tr>
<tr>
<td>inoise(A0)</td>
<td>noise current function</td>
</tr>
<tr>
<td>vnoise(A0)</td>
<td>noise voltage function</td>
</tr>
<tr>
<td>nf(A0)</td>
<td>noise figure function</td>
</tr>
<tr>
<td>vss(...)</td>
<td>small-signal voltage function</td>
</tr>
<tr>
<td>vlsb(...)</td>
<td>small-signal lower-sideband voltage function</td>
</tr>
<tr>
<td>vusb(...)</td>
<td>small-signal upper-sideband voltage function</td>
</tr>
<tr>
<td>iss(...)</td>
<td>small-signal current function</td>
</tr>
<tr>
<td>ilsb(...)</td>
<td>small-signal lower-sideband current function</td>
</tr>
<tr>
<td>iusb(...)</td>
<td>small-signal upper-sideband current function</td>
</tr>
<tr>
<td>dphase(A0, A1)</td>
<td>Continuous phase difference (radians) between A0 and A1</td>
</tr>
<tr>
<td>dbm(A0, A1)</td>
<td>convert voltage and impedance into dBm</td>
</tr>
<tr>
<td>dbmtoa(A0, A1)</td>
<td>convert dBm and impedance into short circuit current</td>
</tr>
<tr>
<td>dbmtov(A0, A1)</td>
<td>convert dBm and impedance into open circuit voltage</td>
</tr>
<tr>
<td>dbmtow(A0)</td>
<td>convert dBm to watts; takes one argument, e.g., dbmtow (-10) returns 1E-4 watts; can be combined with other functions, such as polar, where the polar argument is in degrees</td>
</tr>
<tr>
<td>dbwtow(A0)</td>
<td>convert dBW to watts; see dbmtow for more information</td>
</tr>
<tr>
<td>innerprod(...)</td>
<td>inner-product function</td>
</tr>
<tr>
<td>thd(A0)</td>
<td>total-harmonic-distortion function</td>
</tr>
</tbody>
</table>
### Table 4-2. Simulator Expressions (Sorted by Functionality)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>norm(A0)</td>
<td>norm function</td>
</tr>
<tr>
<td>rms(...)</td>
<td>root-mean-square function</td>
</tr>
<tr>
<td>toi(A0)</td>
<td>third-order-intercept function</td>
</tr>
<tr>
<td>freq_mult_coef(A0)</td>
<td>frequency multiplier polynomial generator function</td>
</tr>
<tr>
<td>freq_mult_poly(A0, A1)</td>
<td>frequency multiplier polynomial evaluation function</td>
</tr>
<tr>
<td>window(A0, A1, A2, A3, A4)</td>
<td>spectral windowing function</td>
</tr>
<tr>
<td>internal_window(A0, A1, A2, A3, A4, A5)</td>
<td>internal spectral windowing function</td>
</tr>
<tr>
<td>generate_pulse_train_spectra(A0, A1, A2, A3, A4, A5)</td>
<td>generate a pulse train spectra</td>
</tr>
<tr>
<td>internal_generate_pulse_train_spectra(A0, A1, A2, A3, A4, A5)</td>
<td>internal generate a pulse train spectra</td>
</tr>
<tr>
<td>sym_set(A0, A1)</td>
<td>set sym variable to a given value</td>
</tr>
<tr>
<td>log_amp(A0, A1, A2, A3)</td>
<td>successive detection logarithmic amplifier</td>
</tr>
<tr>
<td>log_amp_cas(A0, A1, A2, A3)</td>
<td>true logarithmic amplifier</td>
</tr>
<tr>
<td>generate_qpsk_pulse_spectra(A0, A1, A2, A3, A4, A5, A6)</td>
<td>generate a QPSK pulse train spectra</td>
</tr>
<tr>
<td>internal_generate_qpsk_pulse_spectra(A0, A1, A2, A3, A4, A5, A6)</td>
<td>internal generate a QPSK pulse train spectra</td>
</tr>
<tr>
<td>generate_piqpsk_spectra(A0, A1, A2, A3, A4, A5, A6)</td>
<td>generate a pi/4 QPSK pulse train spectra</td>
</tr>
<tr>
<td>internal_generate_piqpsk_spectra(A0, A1, A2, A3, A4, A5, A6, A7)</td>
<td>internal generate a pi/4 QPSK pulse train spectra</td>
</tr>
<tr>
<td>bin(A0)</td>
<td>function convert a binary to integer</td>
</tr>
<tr>
<td>itob(A0, [A1])</td>
<td>convert integer to binary</td>
</tr>
<tr>
<td>generate_gmsk_pulse_spectra(A0, A1, A2, A3, A4, A5, A6)</td>
<td>generate a gmsk pulse train spectra</td>
</tr>
<tr>
<td>internal_generate_gmsk_pulse_spectra(A0, A1, A2, A3, A4, A5, A6, A7)</td>
<td>internal generate a gmsk pulse train spectra</td>
</tr>
<tr>
<td>generate_qam16_spectra(A0, A1, A2, A3, A4, A5, A6)</td>
<td>generate a 16-QAM pulse train spectra</td>
</tr>
<tr>
<td>internal_generate_qam16_spectra(A0, A1, A2, A3, A4, A5, A6, A7)</td>
<td>internal generate a 16-QAM pulse train spectra</td>
</tr>
</tbody>
</table>
### Table 4-2. Simulator Expressions (Sorted by Functionality)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>generate_gmsk_iq_spectra(A0, A1, A2, A3, A4, A5, A6)</code></td>
<td>generate the gmsk <code>i</code> or <code>q</code></td>
</tr>
<tr>
<td><code>internal_generate_gmsk_iq_spectra(A0, A1, A2, A3, A4, A5, A6, A7)</code></td>
<td>internal generate the gmsk <code>i</code> or <code>q</code></td>
</tr>
<tr>
<td><code>get_fund_freq(A0)</code></td>
<td>Get the frequency associated with a specified fundamental index</td>
</tr>
<tr>
<td><code>internal_get_fund_freq(A0, A1)</code></td>
<td>internal function to get frequency for a specified fundamental index</td>
</tr>
<tr>
<td><code>ktoc(A0)</code></td>
<td>convert Kelvin to Celsius</td>
</tr>
<tr>
<td><code>ctok(A0)</code></td>
<td>convert Celsius to Kelvin</td>
</tr>
<tr>
<td><code>ftoc(A0)</code></td>
<td>convert Fahrenheit to Celsius</td>
</tr>
<tr>
<td><code>ctof(A0)</code></td>
<td>convert Celsius to Fahrenheit</td>
</tr>
<tr>
<td><code>ftok(A0)</code></td>
<td>convert Fahrenheit to Kelvin</td>
</tr>
<tr>
<td><code>ktof(A0)</code></td>
<td>convert Kelvin to Fahrenheit</td>
</tr>
<tr>
<td><code>get_array_size(A0)</code></td>
<td>Get the size of the array</td>
</tr>
<tr>
<td><code>eval_poly(A0, A1, A2)</code></td>
<td>polynomial evaluation function</td>
</tr>
<tr>
<td><code>pwl(...)</code></td>
<td>piecewise-linear function</td>
</tr>
<tr>
<td></td>
<td>( y = pwl(x, x_1, y_1, x_2, y_2, \ldots) )</td>
</tr>
<tr>
<td></td>
<td>( x = ) name of the independent variable.</td>
</tr>
<tr>
<td></td>
<td>( x_k, y_k = ) pairs of known values.</td>
</tr>
<tr>
<td></td>
<td>( x ) can be the name of any swept variable like the internal variables</td>
</tr>
<tr>
<td></td>
<td>( \text{time or freq, or any other variable swept with a</td>
</tr>
<tr>
<td><code>phase_noise_pwl(...)</code></td>
<td>piecewise-linear function for computing phase noise</td>
</tr>
<tr>
<td><code>pwlr(...)</code></td>
<td>piecewise-linear-repeated function</td>
</tr>
<tr>
<td></td>
<td>Repeats the ( pwl() ) function for several cycles.</td>
</tr>
<tr>
<td></td>
<td>( y = pwlr(x, N_{\text{cycles}}, x_1, y_1, x_2, y_2, \ldots) )</td>
</tr>
<tr>
<td></td>
<td>( N_{\text{cycles}} = ) the number of cycles to be repeated.</td>
</tr>
<tr>
<td></td>
<td>Example: see component VtPWL.</td>
</tr>
</tbody>
</table>
Table 4-2. Simulator Expressions (Sorted by Functionality)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pulse(time, [low, high, delay, rise, fall, width, period])</td>
<td>periodic pulse function</td>
</tr>
<tr>
<td>cos_pulse(time, [low, high, delay, rise, fall, width, period])</td>
<td>periodic cosine shaped pulse function</td>
</tr>
<tr>
<td>erf_pulse(time, [low, high, delay, rise, fall, width, period])</td>
<td>periodic error function shaped pulse function</td>
</tr>
<tr>
<td>damped_sin(time, [offset, amplitude, freq, delay, damping, phase])</td>
<td>damped sin function</td>
</tr>
<tr>
<td>multi_freq(time, amplitude, freq1, freq2, n, [seed])</td>
<td>multifrequency function</td>
</tr>
<tr>
<td>exp_pulse(time, [low, high, delay1, tau1, delay2, tau2])</td>
<td>exponential pulse function</td>
</tr>
<tr>
<td>sffm(time, [offset, amplitude, carrier_freq, mod_index, signal_freq])</td>
<td>signal frequency FM</td>
</tr>
<tr>
<td>bitseq(time, [clockfreq, trise, tfall, vlow, vhigh, bitseq])</td>
<td>bit sequence function</td>
</tr>
<tr>
<td>lfsr(A0, A1)</td>
<td>lfsr(taps, seed) returns a string containing complete sequence</td>
</tr>
<tr>
<td>read_data(...)</td>
<td>read_data(&quot;file</td>
</tr>
<tr>
<td>read_lib(...)</td>
<td>read_lib(&quot;libName&quot;, &quot;item&quot;, &quot;fileType&quot;)</td>
</tr>
<tr>
<td>get_block(A0, A1)</td>
<td>HPvar tree from block name function</td>
</tr>
<tr>
<td>get_max_points(A0, A1)</td>
<td>maximum points of independent variable</td>
</tr>
<tr>
<td>dsexpr(A0, A1)</td>
<td>Evaluate a dataset expression to an hpvar</td>
</tr>
<tr>
<td>dstoarray(A0, [A1])</td>
<td>Convert an hpvar to an array</td>
</tr>
<tr>
<td>get_attribute(...)</td>
<td>value of attribute of a set of data</td>
</tr>
<tr>
<td>dep_data(A0, A1, [A2])</td>
<td>dependent variable value</td>
</tr>
<tr>
<td>names(A0, A1)</td>
<td>array of names of indepVars and/or depVars in dataset</td>
</tr>
<tr>
<td>index(A0, A1, [A2, A3])</td>
<td>get index of name in array</td>
</tr>
<tr>
<td>cxform(A0, A1, A2)</td>
<td>transform complex data</td>
</tr>
<tr>
<td>transform(A0, A1, A2, [A3, A4])</td>
<td>transform complex depVars in dataset</td>
</tr>
<tr>
<td>stypexform(A0, A1, A2, A3)</td>
<td>Y, Z, H, G, to S matrix transform</td>
</tr>
</tbody>
</table>
Simulator Variables and Constants

When you are using Simulator Expressions, keep in mind that certain variables and constants are reserved words in ADS. You can use these variables and constants, but you cannot redefine them to something else. The following table lists the simulator variables/constants available in ADS and a brief description of each.

### Table 4-3. Simulator Variables and Constants

<table>
<thead>
<tr>
<th>Variable/Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time = 0 s</td>
<td>the analysis time</td>
</tr>
<tr>
<td>timestep = 1 s</td>
<td>the analysis time step</td>
</tr>
<tr>
<td>tranorder = 1</td>
<td>the transient analysis integration order</td>
</tr>
<tr>
<td>freq = 1e+006 Hz</td>
<td>the analysis frequency</td>
</tr>
<tr>
<td>Nsample = 0</td>
<td>signal processing analysis sample number</td>
</tr>
<tr>
<td>ScheduleCycle = 0</td>
<td>signal processing schedule cycle number</td>
</tr>
<tr>
<td>DefaultValue = -1</td>
<td>signal processing default parameter value</td>
</tr>
<tr>
<td>noisefreq = 1e+006 Hz</td>
<td>the spectral noise analysis frequency</td>
</tr>
</tbody>
</table>
Table 4-3. Simulator Variables and Constants

<table>
<thead>
<tr>
<th>Variable/Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssfreq = 1e+006 Hz</td>
<td>the small-signal mixer analysis frequency</td>
</tr>
<tr>
<td>temp = 25 C</td>
<td>the analysis temperature</td>
</tr>
<tr>
<td>e = 2.71828</td>
<td>2.71838...</td>
</tr>
<tr>
<td>ln10 = 2.30259</td>
<td>ln(10)</td>
</tr>
<tr>
<td>c0 = 2.99792e-008 m/s</td>
<td>the speed of light</td>
</tr>
<tr>
<td>e0 = 8.85419e-012</td>
<td>vacuum permittivity</td>
</tr>
<tr>
<td>u0 = 1.25664e-006</td>
<td>vacuum permeability</td>
</tr>
<tr>
<td>boltzmann = 1.380666e-023</td>
<td>Boltzmann's constant</td>
</tr>
<tr>
<td>qelectron = 1.60218e-019</td>
<td>the charge of an electron</td>
</tr>
<tr>
<td>planck = 6.62608e-034</td>
<td>Planck's constant</td>
</tr>
<tr>
<td>hugereal = 1.79769e+308</td>
<td>largest real number</td>
</tr>
<tr>
<td>tinyreal = 2.22507e-308</td>
<td>smallest real number</td>
</tr>
<tr>
<td>sourceLevel = 1</td>
<td>used for source-level sweeping</td>
</tr>
<tr>
<td>dcSourceLevel = 1</td>
<td>used for DC source-level sweeping</td>
</tr>
<tr>
<td>logRshunt = 0</td>
<td>used for DC Rshunt sweeping</td>
</tr>
<tr>
<td>logNodesetScale = 0</td>
<td>used for DC nodeset simulation</td>
</tr>
<tr>
<td>logRforce = 0</td>
<td>used for HB Rforce sweeping</td>
</tr>
<tr>
<td>mcindex = 0</td>
<td>index for Monte Carlo sweeps</td>
</tr>
<tr>
<td>doeindex = 0</td>
<td>index for Design of Experiment sweeps</td>
</tr>
<tr>
<td>CostIndex = 0</td>
<td>index for optimization cost plots</td>
</tr>
<tr>
<td>mcTrial = 0</td>
<td>trial counter for Monte Carlo based simulations</td>
</tr>
<tr>
<td>optIter = 0</td>
<td>optimization job iteration counter</td>
</tr>
<tr>
<td>doeIter = 0</td>
<td>doe experiment iteration counter</td>
</tr>
<tr>
<td>DeviceIndex = 0</td>
<td>device Index used for noise contribution or DC OP output</td>
</tr>
<tr>
<td>LinearizedElementIndex = 0</td>
<td>index for BudLinearization sweep</td>
</tr>
<tr>
<td>DF_Value = -1e+009</td>
<td>reference to corresponding value defined in Data Flow controller</td>
</tr>
</tbody>
</table>
Mathematical Operators and Hierarchy

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

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

The operators >=, <=, >, <, ==, !=, AND, OR, EQUALS, and NOT EQUALS also produce logical results, producing a logical TRUE or FALSE upon comparing the values of two expressions. These operators are most often used to compare two real numbers or integers. These operators operate differently in AEL than C with string expressions in that they actually perform the equivalent of strcmp() between the first and second operands, and test the return value against 0 using the specified operator.

### Table 4-3. Simulator Variables and Constants

<table>
<thead>
<tr>
<th>Variable/Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF_ZERO_OHMS = 1e-013</td>
<td>symbol for use as zero ohms</td>
</tr>
<tr>
<td>DF_DefaultInt = -1e+009</td>
<td>reference to default int value defined in Data Flow controller</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4-4. Operator Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operator</strong></td>
</tr>
<tr>
<td>( )</td>
</tr>
<tr>
<td>[ ]</td>
</tr>
<tr>
<td>{}</td>
</tr>
<tr>
<td>**</td>
</tr>
<tr>
<td>!</td>
</tr>
<tr>
<td>Operator/Operator Name</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td><code>*</code> multiply</td>
</tr>
<tr>
<td><code>/</code> divide</td>
</tr>
<tr>
<td><code>.*</code> element-wise multiply</td>
</tr>
<tr>
<td><code>./</code> element-wise divide</td>
</tr>
<tr>
<td><code>+</code> add</td>
</tr>
<tr>
<td><code>-</code> subtract</td>
</tr>
<tr>
<td><code>::</code> sequence operator</td>
</tr>
<tr>
<td><code>wildcard</code></td>
</tr>
<tr>
<td><code>&lt;</code> less than</td>
</tr>
<tr>
<td><code>&lt;=</code> less than or equal to</td>
</tr>
<tr>
<td><code>&gt;</code> greater than</td>
</tr>
<tr>
<td><code>&gt;=</code> greater than or equal to</td>
</tr>
<tr>
<td><code>==</code> equal</td>
</tr>
<tr>
<td><code>!=</code> not equal</td>
</tr>
<tr>
<td><code>&amp;&amp;</code> logical and</td>
</tr>
<tr>
<td>`</td>
</tr>
<tr>
<td><code>%</code> modulo</td>
</tr>
</tbody>
</table>
Simulator Expression Reference
## Index

<table>
<thead>
<tr>
<th>Letter</th>
<th>Topic</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Backward Traveling Waves</td>
<td>2-11</td>
</tr>
<tr>
<td></td>
<td>Budget Measurement</td>
<td>2-10</td>
</tr>
<tr>
<td></td>
<td>Built-In Constants</td>
<td>2-9</td>
</tr>
<tr>
<td>C</td>
<td>Case Sensitivity</td>
<td>2-2</td>
</tr>
<tr>
<td></td>
<td>Component Options</td>
<td>2-12</td>
</tr>
<tr>
<td>D</td>
<td>Display Parameter on schematic</td>
<td>2-12</td>
</tr>
<tr>
<td>E</td>
<td>Envelope Data</td>
<td>2-6</td>
</tr>
<tr>
<td>F</td>
<td>Frequency Plan</td>
<td>2-11</td>
</tr>
<tr>
<td>G</td>
<td>Generating Data</td>
<td>2-7</td>
</tr>
<tr>
<td>H</td>
<td>Harmonic Balance Data</td>
<td>2-6</td>
</tr>
<tr>
<td>I</td>
<td>if-then-else Construct</td>
<td>2-7</td>
</tr>
<tr>
<td></td>
<td>Indexing</td>
<td>2-5</td>
</tr>
<tr>
<td></td>
<td>Instance Name</td>
<td>2-11</td>
</tr>
<tr>
<td>M</td>
<td>mathematical operators</td>
<td>4-20</td>
</tr>
<tr>
<td></td>
<td>Matrices</td>
<td>2-5</td>
</tr>
<tr>
<td></td>
<td>Meas</td>
<td>2-12</td>
</tr>
<tr>
<td></td>
<td>MeasEqn</td>
<td>1-1, 2-11</td>
</tr>
<tr>
<td></td>
<td>Measurement Equations</td>
<td>1-1</td>
</tr>
<tr>
<td></td>
<td>Measurements and Expressions</td>
<td>2-2</td>
</tr>
<tr>
<td></td>
<td>Multidimensional Sweeps</td>
<td>2-5</td>
</tr>
<tr>
<td>O</td>
<td>Operator Precedence</td>
<td>2-7</td>
</tr>
<tr>
<td>P</td>
<td>Parameter Sweeps</td>
<td>2-4</td>
</tr>
<tr>
<td>R</td>
<td>Reflections</td>
<td>2-11</td>
</tr>
<tr>
<td>S</td>
<td>Select Parameter</td>
<td>2-12</td>
</tr>
<tr>
<td></td>
<td>Simulation Data</td>
<td>2-2</td>
</tr>
<tr>
<td></td>
<td>simulator functions</td>
<td>1-1, 4-1</td>
</tr>
<tr>
<td></td>
<td>simulator functions, list</td>
<td>4-2</td>
</tr>
<tr>
<td></td>
<td>simulator functions, variables and constants</td>
<td>4-18</td>
</tr>
<tr>
<td></td>
<td>S-Parameters</td>
<td>2-5</td>
</tr>
<tr>
<td></td>
<td>Stability</td>
<td>3-148, 3-149, 3-203, 3-204</td>
</tr>
<tr>
<td>T</td>
<td>Transient Data</td>
<td>2-6</td>
</tr>
<tr>
<td>U</td>
<td>User-Defined Functions</td>
<td>2-12</td>
</tr>
<tr>
<td>V</td>
<td>VarEqn functions</td>
<td>1-1, 2-3</td>
</tr>
</tbody>
</table>

---

Index-1