STGraph - System defined functions

[Version 23.2.16]

Legend: A=array (or specifically: S=scalar; V=vector; M=matrix); E=expression

- Operators
- Mathematical functions
- Statistical functions
- Control functions
- Array functions

See also:

- Monadic polymorphic functions / operators
- Diadic polymorphic functions / operators
- Boolean operators
- Polymorphic functions handling statistical distributions
- Interpolation functions

Operators

x+y [x,y:A] x plus y

x&&y [x,y:A] logical conjunction: x and y

x#y [x,y:A] x concatenated with y

x##y [x:S, y:A; x:A, y:S] array obtained by removing the first x elements from each vector in the last dimension of the array y or the last y elements from each vector in the last dimension of the array x

{x} [x:E] subexpression that is assigned the variable $wn, where n ranges from 0 to 3 according to the order of evaluation

x-y [x,y:A] x minus y

x==y [x,y:A] logical comparison: is x equal to y?

x>=y [x,y:A] logical comparison: is x greater than or equal to y?

x>y [x,y:A] logical comparison: is x greater than y?

x<=y [x,y:A] logical comparison: is x less than or equal to y?

[x:y] [x,y:S] vector of scalars from x to y

[x:y:z] [x,y,z:S] vector of scalars from x to y separated by z

x<y [x,y:A] logical comparison: is x less than y?

-x [x:A] minus x

x%y [x,y:A] x modulus y

x!=y [x,y:A] logical comparison: is x different from y?

!x [x:A] logical negation: not x

x||y [x,y:A] logical disjunction: x or y

f|x [f:function; x:A] if x is a vector, vector whose first element is obtained by applying the dyadic function / operator f to the first two elements of x, the second element is obtained by applying f to the the second and the third element, and so on; if x is a higher order array, array obtained in the same way by applying f in parallel to the elements of each vector of the last dimension

f|[n]x [f:function; x:A; n:integer] as f|x, where f is applied to the dimension n of x

x^y [x,y:A] x to the power of y

x*y [x,y:A] x times y

x/y [x,y:A] x divided by y

f/x [f:function; x:A] if x is a vector, scalar obtained by applying the dyadic function / operator f to the first two elements of x, then to the result and the third element, and so on; if x is a [n]order array, [n-1]order array obtained in the same way by applying f in parallel to the elements of each vector of the last dimension

f/[n]x [f:function; x:A; n:integer] as f/x, where f is applied to the dimension n of x

f\x [f:function; x:A] if x is a vector, vector whose first element is obtained by applying the dyadic function / operator f to the first two elements of x, the second element is obtained by applying f to the result and the third element, and so on; if x is a higher order array, array obtained in the same way by applying f in parallel to the elements of each vector of the last dimension

f\[n]x [f:function; x:A; n:integer] as f\x, where f is applied to the dimension n of x

Mathematical functions

acos(x) [x:A] arccosine of x

asin(x) [x:A] arcsine of x

atan(x) [x:A] arctangent of x

bline(vx,vy,x) [vx,vy:V, x:A] the y value corresponding to x over the segment from (vx[0],vy[0]) to (vx[1],vy[1]), and constant elsewhere
cos(x) [x:A] cosine of x
deg2rad(x) [x:A] value of x converted from degrees to radians
exp(x) [x:A] exponential of x, i.e., e to the power of x
FFT(x,s) [x:V,M; s:S] if s==1, fast Fourier transform of the vector x; if s==2, inverse fast Fourier transform of the matrix x
int(x) [x:A] integer part of x
integral(x) [x:A] in state transitions of state nodes, iterative sum of x, according to the chosen integration algorithm
line(vx,vy,x) [vx,vy:V, x:A] the y value corresponding to x over the straight line crossing the points vx[0],vy[0] and vx[1],vy[1]
log(x) [x:A] natural logarithm of x
log(x,y) [x,y:A] logarithm of x to the base y
max(x,y) [x,y:A] maximum between x and y
min(x,y) [x,y:A] minimum between x and y
mod(x,y) [x,y:A] x modulus y
pline(vx,vy,x) [vx,vy:V, x:A] the y value corresponding to x over the polyline whose vertices are in the vectors vx and vy
rad2deg(x) [x:A] value of x converted from radians to degrees
round(x,y) [x,y:A] x rounded to y decimals
sigmoid(vx,vy,x) [vx,vy:V, x:A] the y value corresponding to x over the sigmoid controlled by the points vx[0],vy[0] and vx[1],vy[1]
sign(x) [x:A] sign of x, i.e., 1 if x > 0, -1 if x < 0, 0 if x == 0
sin(x) [x:A] sine of x
spline(vx,vy,x) [vx,vy:V, x:A] the y value corresponding to x over the spline whose node points are in the vectors vx and vy
sqrt(x) [x:A] square root of x
tan(x) [x:A] tangent of x
wrap(x,y) [x,y:A] x modulus y also dealing with negative values

Statistical functions

beta(v,x,s) [v:V; x:A; s:S] beta probability distribution of parameters v=[p1,p2], where p1=alpha (default value: 1), p2=beta (default value: 1): if x and s are not specified, random number extracted from the distribution; if s==0, pdf value at the point x; if s==1, cdf value at the point x
binomial(v,x,s) [v:V; x:A; s:S] binomial probability distribution of parameters v=[p1,p2], where p1 =population cardinality (default value: 10), p2=probability (default value: 0.5): if x and s are not specified, random number extracted from the distribution; if s==0, pdf value at the point x; if s==1, cdf value at the point x
chiSquare(v,x,s) [v:V; x:A; s:S] chi square probability distribution of parameters v=[p1], where p1=degrees of freedom (default value: 5): if x and s are not specified, random number extracted from the distribution; if s==0, pdf value at the point x; if s==1, cdf value at the point x; if s==2, inverse cdf value at the probability x
exponential(v,x,s) [v:V; x:A; s:S] exponential probability distribution of parameters v=[p1], where p1 =lambda (p1>0; default value: 1): random number extracted from the distribution if x and s are not specified; value of the pdf, if s==0 or the cdf, if s==1, at the point x

gamma(v,x,s) [v:V; x:A; s:S] gamma probability distribution of parameters `v=[p1,p2]`, where `p1`=alpha (default value: `1`), `p2`=beta (default value: `1`): if `x` and `s` are not specified, random number extracted from the distribution; if `s==0`, pdf value at the point `x`; if `s==1`, cdf value at the point `x`; if `s==2`, inverse cdf value at the probability `x`

gaussian(v,x,s) [v:V; x:A; s:S] gaussian probability distribution of parameters `v=[p1,p2]`, where `p1`=mean (default value: `0`), `p2`=stddev (default value: `1`): if `x` and `s` are not specified, random number extracted from the distribution; if `s==0`, pdf value at the point `x`; if `s==1`, cdf value at the point `x`; if `s==2`, inverse cdf value at the probability `x`

poisson(v,x,s) [v:V; x:A; s:S] Poisson probability distribution of parameters `v=[p1]`, where `p1`=mean (`p1>0`; default value: `5`): random number extracted from the distribution if `x` and `s` are not specified; value of the pdf, if `s==0` or the cdf, if `s==1`, at the point `x`

rand(x,y) [x,y:S] random number extracted from a uniform distribution between `0` and `1`, or between `0` and `x` if only `x` is specified, or between `x` and `y` if both are specified

randInt(x) [x:A] integer random number extracted from a uniform distribution between `0` and `int(x-1)`

tDistribution(v,x,s) [v:V; x:A; s:S] t-distribution probability distribution of parameters `v=[p1]`, where `p1` =degrees of freedom (`p1>0`; default value: `5`): random number extracted from the distribution if `x` and `s` are not specified; value of the pdf, if `s==0` or the cdf, if `s==1`, at the point `x`

uniform(v,x,s) [v:V; x:A; s:S] uniform (rectangular) probability distribution of parameters `v=[p1,p2]`, where `p1`=mean (default value: `0`), `p2`=stddev (default value: `1`): if `x` and `s` are not specified, random number extracted from the distribution; if `s==0`, pdf value at the point `x`; if `s==1`, cdf value at the point `x`; if `s==2`, inverse cdf value at the probability `x`

Control functions

function(x) [x:E] new function, named as the node in which it is used (the node name must start with an underscore) and defined by the expression `x`

getCProp(n,x) [n:node name; x:string] value of the custom property named `x`, either of the node `n` or of the current node if `n` is not specified

getPhases() [ ] vector of phases as specified in the "Phase" custom property

if(c1,x1,c2,x2,...,cn,xn,xn+1) [c1,...,cn:A; v1,...,vn+1:A] conditional structure: return `x1` if `c1` is true, else `x2` if `c2` is true, else ..., and `xn+1` otherwise

indexOrigin() [ ] origin of array indexes, either 0 or 1, as set as model property

isNumber(x) [x:A] true if `x` is a number (i.e., is neither NaN nor Infinity), false otherwise

iter(x,e,z) [x:A; e:E; z:A] value obtained by repeatedly applying the expression `e` (possibly including functions of the form `fun($0,$1)` or `$0op$1`), whose identity element is `z`, to the elements of the vector `x` or to the elements of each vector of the last dimension of the higher dimension array `x`

readFromNet() [ ] value read from a Spacebrew network server

readFromXLS(x,y,r,c) [x:string; y,r,c:S] value read from the row `r` and column `c` of the sheet of index `y` of the xls workbook whose filename is `x`

readFromXLS(x,y,r1,c1,r2,c2) [x:string; y,r1,c1,r2,c2:S] vector or matrix read from the row `r1`, column `c1` and the row `r2`, column `c2` of the sheet of index `y` of the xls workbook whose filename is `x`

sysTime() [ ] number of milliseconds from the simulation start

writeToNet(x) [x:A] send `x` to a Spacebrew network server

Array functions

array(v,e) [v:V, e:E] array whose size is v and whose elements are specified by the expression e, that can contain the system variables $i0,...,$i5, each of them set to the value of the corresponding dimension

conc(x,y,n) [x,y:A, n:integer] x concatenated with y along the dimension n if specified or the last dimension otherwise

dec(x,y,n) [x:S, y:A; x:A, y:S, n:integer] array obtained by removing the first x elements from each vector along the dimension n if specified, or the last dimension otherwise, of the array y or the last y elements from each vector along the dimension n if specified, or the last dimension otherwise, of the array x

frequency(x,d,c) [x:S; d,c:V] vector containing the distribution of the data in d categorized by the values in c ; if x is specified it is supposed that d already contains the distribution, to which x is added

get(x,v0,v1,...) [x:A; v0,v1,...:V] array obtained from the array x by extracting its subarray of indexes v0 from the 0-th dimension, then its subarray of indexes v1 from the 1-st dimension, then... (equivalent to x[v0,v1,...])

getData(x,y) [x,y:A] vector of the elements of the array y identified by the indexes in the vector x if y is a vector or by the indexes in the matrix x if y is a higher order array

getIndex(s,x) [s:S, x:A] if x is a vector, vector of the indexes of the occurrences of s in x; if x is a higher order array, matrix of the indexes of the occurrences of s in x

@x [x:A] size of x

order(x) [x:A] order, i.e., number of dimensions, of the array x

remove(x,v) [x:A; v:S,V] array obtained by removing the v-th element(s) from the first dimension of the array x

resize(x,v) [x:A; v:V] array obtained by resizing the array x to the size v, and by trimming the exceeding trailing elements or padding with trailing zeros if required

set(x,v0,v1,...,y) [x:A; v0,v1,...:V; y:A] array obtained from the array x by substituting its subarray of indexes v0 from the 0-th dimension, of indexes v1 from the 1-st dimension, ... with the array y

shift(x,y,s) [x,y:A; s:S] array obtained by concatenating the array y to the array x at the left and removing the same number of elements from x at the right if s==0 or it is not specified, or in reverse order, if s==1

shuffle(x) [x:A] array obtained by randomly permutating the elements of the array x

size(x) [x:A] size of x

sort(x,s) [x:A; s:S] array obtained by sorting the elements of the array x, either in direct order, if s==0 or it is not specified, or in reverse order, if s==1

transpose(x) [x:A] array obtained by transposing the array x

STGraph - Functions listed by functional categories

STGraph - Monadic polymorphic functions / operators

Each of the following functions / operators has a single argument, which are arrays, and therefore in particular scalars (`0`-order arrays), vectors (`1`-order arrays), or matrices (`2`-order arrays), and behaves polymorphically.

Given `f(x)`:

- if `x` is an empty `n>0`-order array, the result is `x` itself
- if `x` is a scalar, the result is a scalar
- if `x` is a `n>0`-order array, the result is a `n>0`-order array `y` such that `y[i1,...in]=[f(x[i1]),...,f(x[i1])]`

acos(x)
asin(x)
atan(x)
cos(x)
deg2rad(x)
exp(x)
int(x)
isNumber(x)
log(x)
-x
!x
rad2deg(x)
randInt(x)
sign(x)
sin(x)
sqrt(x)
tan(x)

STGraph - Diadic polymorphic functions / operators

Each of the following functions / operators has two arguments, which are arrays, and therefore in particular scalars (`0`-order arrays), vectors (`1`-order arrays), or matrices (`2`-order arrays), and behaves polymorphically.

Given `f(x1,x2)`:

- if `x1` or `x2` is an empty `n>0`-order array, the result is `0.0`
- if `x1` and `x2` are scalars, the result is a scalar
- if `x1` is a scalar and `x2` is a `n>0`-order array, the result is a `n`-order array `y` such that `y[i1,...in]=f(x1,x2[i1,...in])` (or viceversa)
- if `x1` and `x2` are `n>0`-order arrays of the same dimensions, the result is a `n`-order array `y` such that `y[i1,...in]=f(x1[i1,...in],x2[i1,...in])`
- if `x1` is a `n>1`-order array and `x2` is a `n-1`-order array, and their first `n-1` dimensions are the same, the result is a `n`-order array `y` such that `y[i1,...in]=f(x1[i1,...in],x2[i1,...in-1])`

An exception is thrown in the other cases.

log(x,y)
max(x,y)
min(x,y)
mod(x,y)
x+y
x&&y
x-y
x==y
x>=y
x>y
x<=y
x<y
x%y
x!=y
x||y
x^y
x*y
x/y
round(x,y)
wrap(x,y)

STGraph - Boolean operators

Each of the following operators deals with with the true and false boolean values, which are defined in STGraph as follows:
- each value greater than zero (actually: greater or equal to `EPSILON`) is evaluated as true;
- each value less or equal than zero (actually: less to `EPSILON`) is evaluated as false.
The only primitive type in STGraph is double: hence, true and false are implemented as `1.0` and `0.0` respectively.

x&&y
x==y
x>=y
x>y
x<=y
x<y
x!=y
!x
x||y

STGraph - Polymorphic functions handling statistical distributions

Each of the following functions can be called for either generating a random number extracted from the given statistical distribution, or returning the `y` values of either the probability density function (PDF), or the

cumulative density function (CDF), or the inverse cumulative density function for the given `x` values. Provided that the parameters characterizing the distribution `f` are stored in the vector `v` (e.g., in the case of the gaussian distribution `v=[mean,stddev]`), then the following usages are allowed:
- `f()` returns a random number extracted from the distribution with default parameters
- `f(v)` returns a random number extracted from the distribution whose parameters are in the vector `v`
- `f(v,x,0)` returns the PDF values of the distribution whose parameters are in the vector `v` and for the arguments in the array `x`
- `f(v,x,1)` returns the CDF values of the distribution whose parameters are in the vector `v` and for the arguments in the array `x`
- `f(v,x,2)` returns the inverse CDF values of the distribution whose parameters are in the vector `v` and for the arguments in the array `x`, in the range `[0,1]` (or `(0,1)`).
The algorithmic relations between PDFs and CDFs are as follows:
- the CDF is approximated by `(+\pdf)*deltax`, where `pdf` is the vector of the PDF values and `deltax` is the step of the domain values;
- the PDF is approximated by `(-|-cdf)/deltax`, where `cdf` is the vector of the CDF values and `deltax` is as before.

beta(v,x,s)
binomial(v,x,s)
chiSquare(v,x,s)
exponential(v,x,s)
gamma(v,x,s)
gaussian(v,x,s)
poisson(v,x,s)
tDistribution(v,x,s)
uniform(v,x,s)

STGraph - Interpolation functions

Each of the following functions generates an interpolation value according to the same logic:
- two vectors `vx` and `vy` are specified, as control points defining the interpolating curve
- an array of `x` values is specified, whose corresponding `y` values on the curve are computed by the function.

bline(vx,vy,x)
line(vx,vy,x)
pline(vx,vy,x)
sigmoid(vx,vy,x)
spline(vx,vy,x)

STGraph - Functions

acos

Format: `acos(x)`

Constraints: `x` is an array

Description: arccosine of `x`

`x` is expressed in radians.

`acos` is a mathematical function

`acos` is a monadic polymorphic function

Exceptions: no

array

Format: `array(v,e)`

Constraints: `v` is a scalar or a vector; `e` is an expression

Description: array whose size is `v` and whose elements are specified by the expression `e`, that can contain the system variables `$i0`,...,`$i5`, each of them set to the value of the corresponding dimension

`array` is an array function

Exceptions: no

asin

Format: `asin(x)`

Constraints: `x` is an array

Description: arcsine of `x`

`x` is expressed in radians.

`asin` is a mathematical function

`asin` is a monadic polymorphic function

Exceptions: no

atan

Format: `atan(x)`

Constraints: `x` is an array

Description: arctangent of x
x is expressed in radians.

atan is a mathematical function
atan is a monadic polymorphic function

Exceptions: no

beta

Format: `beta(v,x,s)`
Constraints: v (optional) is a vector; x (optional) is an array; s (optional) is a scalar, either 0, or 1, or 2
Description: beta probability distribution of parameters v=[p1,p2], where p1=alpha (default value: 1), p2=beta (default value: 1): if x and s are not specified, random number extracted from the distribution; if s==0, pdf value at the point x; if s==1, cdf value at the point x

beta is a statistical function
beta is a statistical distribution function

Exceptions: s must be either 0 or 1.

binomial

Format: `binomial(v,x,s)`
Constraints: v (optional) is a vector; x (optional) is an array; s (optional) is a scalar, either 0 or 1
Description: binomial probability distribution of parameters v=[p1,p2], where p1=population cardinality (default value: 10), p2=probability (default value: 0.5): if x and s are not specified, random number extracted from the distribution; if s==0, pdf value at the point x; if s==1, cdf value at the point x

binomial is a statistical function
binomial is a statistical distribution function

Exceptions: s must be either 0 or 1.

bline

Format: `bline(vx,vy,x)`
Constraints: vx and vy are vectors; x is an array
Description: the y value corresponding to x over the segment from (vx[0],vy[0]) to (vx[1],vy[1]), and constant elsewhere

`bline` is [a mathematical function]
`bline` is [an interpolation function]

Exceptions: Both `vx` and `vy` must have two elements; `vx[0]` must be less than `vx[1]`.

chiSquare

Format: `chiSquare(v,x,s)`
Constraints: `v` (optional) is a vector; `x` (optional) is an array; `s` (optional) is a scalar, either `0`, or `1`, or `2`
Description: chi square probability distribution of parameters `v=[p1]`, where `p1`=degrees of freedom (default value: `5`): if `x` and `s` are not specified, random number extracted from the distribution; if `s==0`, pdf value at the point `x`; if `s==1`, cdf value at the point `x`; if `s==2`, inverse cdf value at the probability `x`

`chiSquare` is [a statistical function]
`chiSquare` is [a statistical distribution function]

Exceptions: `s` must be either `0`, `1`, or `2`; if `s==2` then all elements of `x` must be in the interval `[0,1]`.

conc

Format: `conc(x,y,n)`
Constraints: `x` and `y` are arrays, `n` (optional) is an integer
Description: `x` concatenated with `y` along the dimension `n` if specified or the last dimension otherwise
If `n` is omitted it is equivalent to the operator `x#y`.

`conc` is [an array function]

Exceptions: It must be possible to concatenate `x` with `y` in terms of their dimensions; `n` must be a correct dimension for `x`, currently `0` or `1`.

cos

Format: `cos(x)`
Constraints: `x` is an array
Description: cosine of `x`
`x` is expressed in radians.

`cos` is [a mathematical function]

`cos` is a monadic polymorphic function

Exceptions: no

dec

Format: `dec(x,y,n)`
Constraints: `x` is a scalar and `y` an array, or viceversa, `n` (optional) is an integer
Description: array obtained by removing the first `x` elements from each vector along the dimension `n` if specified, or the last dimension otherwise, of the array `y` or the last `y` elements from each vector along the dimension `n` if specified, or the last dimension otherwise, of the array `x`
If `n` is omitted it is equivalent to the operator `x##y`.

`dec` is an array function

Exceptions: `n` must be a correct dimension for `x`, currently `0` or `1`.

deg2rad

Format: `deg2rad(x)`
Constraints: `x` is an array
Description: value of `x` converted from degrees to radians

`deg2rad` is a mathematical function
`deg2rad` is a monadic polymorphic function

Exceptions: no

exp

Format: `exp(x)`
Constraints: `x` is an array
Description: exponential of `x`, i.e., `e` to the power of `x`

`exp` is a mathematical function
`exp` is a monadic polymorphic function

Exceptions: no

exponential

Format: `exponential(v,x,s)`

Constraints: `v` (optional) is a vector; `x` (optional) is an array; `s` (optional) is a scalar, either `0` or `1`

Description: exponential probability distribution of parameters `v=[p1]`, where `p1`=lambda (`p1>0`; default value: `1`): random number extracted from the distribution if `x` and `s` are not specified; value of the pdf, if `s==0` or the cdf, if `s==1`, at the point `x`

`exponential` is a statistical function
`exponential` is a statistical distribution function

Exceptions: All elements of `x` must be strictly positive; `s` must be either `0` or `1`

FFT

Format: `FFT(x,s)`

Constraints: `s` is a scalar, either `1` or `2`; `x` is a vector if `x==1`, and a matrix if `x==2`

Description: if `s==1`, fast Fourier transform of the vector `x`; if `s==2`, inverse fast Fourier transform of the matrix `x`
If FFT is computed, it takes a vector and returns a `size(x) x 2` matrix, to be interpreted as a vector of complex numbers. If inverse FFT is computed, it takes a `2` column matrix, to be interpreted as a vector of complex numbers, and returns a vector.

`FFT` is a mathematical function

Exceptions: If FFT is computed, the size of `x` must be a power of `2`. If inverse FFT is computed, the row number of `x` must be a power of `2` and its column number must be `2`.

frequency

Format: `frequency(x,d,c)`

Constraints: `x` (optional) is a scalar; `d` and `c` are vectors

Description: vector containing the distribution of the data in `d` categorized by the values in `c`; if `x` is specified it is supposed that `d` already contains the distribution, to which `x` is added
The elements of the vector `c` are the right bounds of the intervals defining the categories; the last category is automatically added to contain the values greater than the last element of `c`.

`frequency` is an array function

Exceptions: `y` and `z` must have the same size.

function

Format: `function(x)`

Constraints: `x` is an expression

Description: new function, named as the node in which it is used (the node name must start with an underscore) and defined by the expression `x`
The arguments of the defined function are referred to in the expression as `$a0`, `$a1`, ... (up to `$a3`).
For example, if the expression of the node `_x` is `function(($a0+$a1)/2)` then `_x(2,3)` produces the result `(2+3)/2=2.5`.
Furthermore, such expression can contain the system variable `$numArgs`, that is set to the number of arguments to the function.
Note that the (meta)function `function` allows recursive definitions. For example, the factorial product function can be defined in a node named `_fact` as follows: `function(if($a0==0,1,$a0*_fact($a0-1)))`.

`function` is [a control function]

Exceptions: no

gamma

Format: `gamma(v,x,s)`

Constraints: `v` (optional) is a vector; `x` (optional) is an array; `s` (optional) is a scalar, either `0`, or `1`, or `2`

Description: gamma probability distribution of parameters `v=[p1,p2]`, where `p1`=alpha (default value: `1`), `p2`=beta (default value: `1`): if `x` and `s` are not specified, random number extracted from the distribution; if `s==0`, pdf value at the point `x`; if `s==1`, cdf value at the point `x`; if `s==2`, inverse cdf value at the probability `x`

`gamma` is [a statistical function]
`gamma` is [a statistical distribution function]

Exceptions: `s` must be either `0`, `1`, or `2`; if `s==2` then all elements of `x` must be in the interval `[0,1]`.

gaussian

Format: `gaussian(v,x,s)`

Constraints: `v` (optional) is a vector; `x` (optional) is an array; `s` (optional) is a scalar, either `0` or `1`

Description: gaussian probability distribution of parameters `v=[p1,p2]`, where `p1`=mean (default value: `0`), `p2`=stddev (default value: `1`): if `x` and `s` are not specified, random number extracted from the distribution; if `s==0`, pdf value at the point `x`; if `s==1`, cdf value at the point `x`; if `s==2`, inverse cdf value at the probability x

gaussian is a statistical function
gaussian is a statistical distribution function

Exceptions: s must be either 0, 1, or 2; if s==2 then all elements of x must be in the interval [0,1].

get

Format: get(x,v0,v1,...)
Constraints: x is an array; v1, v2, ... are scalars or vectors
Description: array obtained from the array x by extracting its subarray of indexes v0 from the 0-th dimension, then its subarray of indexes v1 from the 1-st dimension, then... (equivalent to x[v0,v1,...])

get is an array function

Exceptions: no

getCProp

Format: getCProp(n,x)
Constraints: n (optional) is a node name; x is a string
Description: value of the custom property named x, either of the node n or of the current node if n is not specified
Only numerical values are dealt with. If specified, n must be written without quotes.

getCProp is a control function

Exceptions: If specified, n must be the name of a connected node. x must be the name of an existing custom property, whose value must be a number.

getData

Format: getData(x,y)
Constraints: x and y are arrays
Description: vector of the elements of the array y identified by the indexes in the vector x if y is a vector or by the indexes in the matrix x if y is a higher order array
If no elements are not found, it returns the scalar -1.

`getData` is [an array function]

Exceptions: no

getIndex

Format: `getIndex(s,x)`
Constraints: `s` is a scalar; `x` is an array
Description: if `x` is a vector, vector of the indexes of the occurrences of `s` in `x`; if `x` is a higher order array, matrix of the indexes of the occurrences of `s` in `x`
If `s` is not found, it returns the vector `[-1]`.

`getIndex` is [an array function]

Exceptions: no

getPhases

Format: `getPhases()`
Constraints:
Description: vector of phases as specified in the "Phase" custom property
The format for the custom property is, e.g., 1,5-8.

`getPhases` is [a control function]

Exceptions: The custom property "Phase" must be defined and set as specified.

if

Format: `if(c1,x1,c2,x2,...,cn,xn,xn+1)`
Constraints: all arguments are arrays, either of the same size or of “compatible size” (see the Notes)
Description: conditional structure: return `x1` if `c1` is true, else `x2` if `c2` is true, else ..., and `xn+1` otherwise
The function `if` operates as a chain of if ... then ... else if ... then ...
Its simplest form, `if(c,v1,v2)`, is equivalent to the structure if `c` then `v1` else `v2`.
It behaves "as polymorphically as possible": in particular, if `c`, `v1` and `v2` are arrays of the same size, the function produces an array of that size, such that `result[i1,...,in] == v1[i1,...,in]` if `c[i1,...,in]` is true and `result[i1,...,in] == v2[i1,...,in]` otherwise.
Given the general form, `if(c1,v1,c2,v2,...,cn,vn,vn+1)`, the conditions `ci` must have the same size. If they are scalars, the values `vj` are not constrained. Otherwise, the values `vj` must have the same size of the conditions `ci` or must be scalars.

`if` is a control function

Exceptions: The number of parameters must be odd.

indexOrigin

Format: `indexOrigin()`
Constraints:
Description: origin of array indexes, either 0 or 1, as set as model property

`indexOrigin` is a control function

Exceptions: no

int

Format: `int(x)`
Constraints: `x` is an array
Description: integer part of `x`

`int` is a mathematical function
`int` is a monadic polymorphic function

Exceptions: no

integral

Format: `integral(x)`
Constraints: `x` is an array
Description: in state transitions of state nodes, iterative sum of `x`, according to the chosen integration algorithm
In the case of the Euler algorithm, the value is `this+x*timeD`.

`integral` is a mathematical function

Exceptions: no

isNumber

Format: `isNumber(x)`

Constraints: `x` is an array

Description: true if `x` is a number (i.e., is neither NaN nor Infinity), false otherwise

`isNumber` is a control function

`isNumber` is a monadic polymorphic function

Exceptions: no

iter

Format: `iter(x,e,z)`

Constraints: `x` is an array; `e` is an expression; `z` is an array

Description: value obtained by repeatedly applying the expression `e` (possibly including functions of the form `fun($0,$1)` or `$0op$1`), whose identity element is `z`, to the elements of the vector `x` or to the elements of each vector of the last dimension of the higher dimension array `x`

`iter` is a control function

Exceptions: `x` must be non null.

line

Format: `line(vx,vy,x)`

Constraints: `vx` and `vy` are vectors; `x` is an array

Description: the `y` value corresponding to `x` over the straight line crossing the points `vx[0],vy[0]` and `vx[1],vy[1]`

`line` is a mathematical function

`line` is an interpolation function

Exceptions: Both `vx` and `vy` must have two elements; `vx[0]` must be less than `vx[1]`.

log-1

Format: `log(x)`

Constraints: `x` is an array

Description: natural logarithm of `x`

`log-1` is a mathematical function
`log-1` is a monadic polymorphic function

Exceptions: `x` must be strictly positive.

log-2

Format: `log(x,y)`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: logarithm of `x` to the base `y`

`log-2` is a mathematical function
`log-2` is a diadic polymorphic function

Exceptions: As specified here. Furthermore, `x` and `y` must be strictly positive.

max

Format: `max(x,y)`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: maximum between `x` and `y`

`max` is a mathematical function
`max` is a diadic polymorphic function

Exceptions: As specified here.

min

Format: `min(x,y)`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: minimum between `x` and `y`

`min` is a mathematical function
`min` is a diadic polymorphic function

Exceptions: As specified here.

mod

Format: `mod(x,y)`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: `x` modulus `y`
Equivalent to the operator `x%y`.

`mod` is a mathematical function
`mod` is a diadic polymorphic function

Exceptions: As specified here.

Addition

Format: `x+y`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: `x` plus `y`

`Addition` is an operator
`Addition` is a diadic polymorphic function

Exceptions: As specified here.

And

Format: `x&&y`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: logical conjunction: `x` and `y`

`And` is an operator
`And` is a diadic polymorphic function, a boolean operator

Exceptions: As specified here.

Conc

Format: `x#y`
Constraints: `x` and `y` are arrays
Description: `x` concatenated with `y`
Equivalent to the function `conc(x,y)`. It must be possible concatenate `x` with `y` in terms of their sizes.

`Conc` is an operator

Exceptions: no

Dec

Format: `x##y`
Constraints: `x` is a scalar and `y` an array, or viceversa
Description: array obtained by removing the first `x` elements from each vector in the last dimension of the array `y` or the last `y` elements from each vector in the last dimension of the array `x`
Equivalent to the function `dec(x,y)`.

`Dec` is an operator

Exceptions: no

DefExpr

Format: `{x}`
Constraints: `x` is an expression
Description: subexpression that is assigned the variable `$wn`, where `n` ranges from 0 to 3 according to the order of evaluation

`DefExpr` is an operator

Exceptions: no

Difference

Format: `x-y`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: `x` minus `y`

`Difference` is an operator

`Difference` is a diadic polymorphic function

Exceptions: As specified here.

Eq

Format: `x==y`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: logical comparison: is `x` equal to `y`?

`Eq` is an operator
`Eq` is a diadic polymorphic function, a boolean operator

Exceptions: As specified here.

GE

Format: `x>=y`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: logical comparison: is `x` greater than or equal to `y`?

`GE` is an operator
`GE` is a diadic polymorphic function, a boolean operator

Exceptions: As specified here.

GT

Format: `x>y`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: logical comparison: is `x` greater than `y`?

`GT` is an operator
`GT` is a diadic polymorphic function, a boolean operator

Exceptions: As specified here.

LE

Format: `x<=y`

Constraints: `x` and `y` are arrays (the constraints are specified here)

Description: logical comparison: is `x` less than or equal to `y`?

`LE` is an operator

`LE` is a diadic polymorphic function, a boolean operator

Exceptions: As specified here.

GenVect1

Format: `[x:y]`

Constraints: `x` and `y` are scalars

Description: vector of scalars from `x` to `y`

`GenVect1` is an operator

Exceptions: no

GenVect2

Format: `[x:y:z]`

Constraints: `x`, `y` and `z` are scalars

Description: vector of scalars from `x` to `y` separated by `z`

`GenVect2` is an operator

Exceptions: no

LT

Format: `x<y`

Constraints: `x` and `y` are arrays (the constraints are specified here)

Description: logical comparison: is `x` less than `y`?

`LT` is an operator

`LT` is a diadic polymorphic function, a boolean operator

Exceptions: As specified here.

Minus

Format: `-x`

Constraints: `x` is an array

Description: minus `x`

`Minus` is an operator

`Minus` is a monadic polymorphic function

Exceptions: no

Mod

Format: `x%y`

Constraints: `x` and `y` are arrays (the constraints are specified here)

Description: `x` modulus `y`

Equivalent to the function `mod(x,y)`.

`Mod` is an operator

`Mod` is a diadic polymorphic function

Exceptions: As specified here.

NE

Format: `x!=y`

Constraints: `x` and `y` are arrays (the constraints are specified here)

Description: logical comparison: is `x` different from `y`?

`NE` is an operator

`NE` is a diadic polymorphic function, a boolean operator

Exceptions: As specified here.

Not

Format: `!x`
Constraints: `x` is an array
Description: logical negation: not `x`

`Not` is an operator
`Not` is a monadic polymorphic function, a boolean operator

Exceptions: no

Or

Format: `x||y`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: logical disjunction: `x` or `y`

`Or` is an operator
`Or` is a diadic polymorphic function, a boolean operator

Exceptions: As specified here.

PairScan-1

Format: `f|x`
Constraints: `f` is a function or an operator; `x` is an array
Description: if `x` is a vector, vector whose first element is obtained by applying the dyadic function / operator `f` to the first two elements of `x`, the second element is obtained by applying `f` to the the second and the third element, and so on; if `x` is a higher order array, array obtained in the same way by applying `f` in parallel to the elements of each vector of the last dimension
If `f` is a function, then `x` must be delimited by parentheses (e.g., `max|(v)`).

`PairScan-1` is an operator

Exceptions: `x` must be a non-null array.

PairScan-2

Format: `f|[n]x`
Constraints: `f` is a function or an operator; `x` is an array; `n` is an integer

Description: as f|x, where `f` is applied to the dimension `n` of `x`
If `f` is a function, then `x` must be delimited by parentheses (e.g., `max|[0](v)`).

`PairScan-2` is an operator

Exceptions: `x` must be a non-null array; `n` must be a correct dimension for `x`, currently between `0` and `2`.

Power

Format: `x^y`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: `x` to the power of `y`

`Power` is an operator
`Power` is a diadic polymorphic function

Exceptions: As specified here.

Product

Format: `x*y`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: `x` times `y`

`Product` is an operator
`Product` is a diadic polymorphic function

Exceptions: As specified here.

Quotient

Format: `x/y`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: `x` divided by `y`

`Quotient` is an operator
`Quotient` is a diadic polymorphic function

Exceptions: As specified here.

Reduction-1

Format: `f/x`
Constraints: `f` is a function or an operator; `x` is an array
Description: if `x` is a vector, scalar obtained by applying the dyadic function / operator `f` to the first two elements of `x`, then to the result and the third element, and so on; if `x` is a [n]order array, [n-1]order array obtained in the same way by applying `f` in parallel to the elements of each vector of the last dimension
If `f` is a function, then `x` must be delimited by parentheses (e.g., `max/(v)`).

`Reduction-1` is an operator

Exceptions: `x` must be a non-null array.

Reduction-2

Format: `f/[n]x`
Constraints: `f` is a function or an operator; `x` is an array; `n` is an integer
Description: as f/x, where `f` is applied to the dimension `n` of `x`
If `f` is a function, then `x` must be delimited by parentheses (e.g., `max/[0](v)`).

`Reduction-2` is an operator

Exceptions: `x` must be a non-null array; `n` must be a correct dimension for `x`, currently between `0` and `2`.

Scan-1

Format: `f\x`
Constraints: `f` is a function or an operator; `x` is an array
Description: if `x` is a vector, vector whose first element is obtained by applying the dyadic function / operator `f` to the first two elements of `x`, the second element is obtained by applying `f` to the result and the third element, and so on; if `x` is a higher order array, array obtained in the same way by applying `f` in parallel to the elements of each vector of the last dimension
If `f` is a function, then `x` must be delimited by parentheses (e.g., `max\(v)`).

`Scan-1` is an operator

Exceptions: `x` must be a non-null array.

Scan-2

Format: `f\[n]x`
Constraints: `f` is a function or an operator; `x` is an array; `n` is an integer
Description: as f\x, where `f` is applied to the dimension `n` of `x`
If `f` is a function, then `x` must be delimited by parentheses (e.g., `max\[0](v)`).

`Scan-2` is an operator

Exceptions: `x` must be a non-null array; `n` must be a correct dimension for `x`, currently between `0` and `2`.

Size

Format: `@x`
Constraints: `x` is an array
Description: size of `x`
Equivalent to the function `size(x)`. The value is zero if `x` is a scalar, and a vector otherwise.

`Size` is an array function

Exceptions: no

order

Format: `order(x)`
Constraints: `x` is an array
Description: order, i.e., number of dimensions, of the array `x`

`order` is an array function

Exceptions: no

pline

Format: `pline(vx,vy,x)`
Constraints: `vx` and `vy` are vectors; `x` is an array
Description: the `y` value corresponding to `x` over the polyline whose vertices are in the vectors `vx` and `vy`

`pline` is a mathematical function
`pline` is an interpolation function

Exceptions: `vx` and `vy` must have the same number of elements, at least two; for each `i`, `vx[i]` must be less than `vx[i+1]`.

poisson

Format: `poisson(v,x,s)`
Constraints: `v` (optional) is a vector; `x` (optional) is an array; `s` (optional) is a scalar, either `0` or `1`
Description: Poisson probability distribution of parameters `v=[p1]`, where `p1`=mean (`p1>0`; default value: `5`): random number extracted from the distribution if `x` and `s` are not specified; value of the pdf, if `s==0` or the cdf, if `s==1`, at the point `x`

`poisson` is a statistical function
`poisson` is a statistical distribution function

Exceptions: All elements of `x` must be strictly positive; `s` must be either `0` or `1`.

rad2deg

Format: `rad2deg(x)`
Constraints: `x` is an array
Description: value of `x` converted from radians to degrees

`rad2deg` is a mathematical function
`rad2deg` is a monadic polymorphic function

Exceptions: no

rand

Format: `rand(x,y)`
Constraints: `x` (optional) is a scalar; `y` (optional) is a scalar
Description: random number extracted from a uniform distribution between `0` and `1`, or between `0` and `x` if only `x` is specified, or between `x` and `y` if both are specified

`rand` is a statistical function

Exceptions: If `x` is specified, it must be strictly positive; if both `x` and `y` are specified, the former must be less than the latter.

randInt

Format: `randInt(x)`
Constraints: `x` is an array
Description: integer random number extracted from a uniform distribution between `0` and `int(x-1)`

`randInt` is a statistical function
`randInt` is a monadic polymorphic function

Exceptions: no

readFromNet

Format: `readFromNet()`
Constraints:
Description: value read from a Spacebrew network server

`readFromNet` is a control function

Exceptions: The Spacebrew server must be reachable.

readFromXLS-1

Format: `readFromXLS(x,y,r,c)`
Constraints: `x` is a string; `y`, `r`, and `c` are scalars
Description: value read from the row `r` and column `c` of the sheet of index `y` of the xls workbook whose filename is `x`
The xls workbook must be in the same directory of the model, and the filename `x` must be specified with no pathname.

`readFromXLS-1` is a control function

Exceptions: no

readFromXLS-2

Format: `readFromXLS(x,y,r1,c1,r2,c2)`
Constraints: `x` is a string; `y`, `r1`, `c1`, `r2`, and `c2` are scalars
Description: vector or matrix read from the row `r1`, column `c1` and the row `r2`, column `c2` of the sheet of index `y` of the xls workbook whose filename is `x`
The xls workbook must be in the same directory of the model, and the filename `x` must be specified with no pathname.

`readFromXLS-2` is a control function

Exceptions: no

remove

Format: `remove(x,v)`
Constraints: `x` is an array; `v` is either a scalar or a vector
Description: array obtained by removing the `v`-th element(s) from the first dimension of the array `x`

`remove` is an array function

Exceptions: no

resize

Format: `resize(x,v)`
Constraints: `x` is an array; `v` is a vector
Description: array obtained by resizing the array `x` to the size `v`, and by trimming the exceeding trailing elements or padding with trailing zeros if required

`resize` is an array function

Exceptions: no

round

Format: `round(x,y)`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: `x` rounded to `y` decimals

`round` is a mathematical function
`round` is a diadic polymorphic function

Exceptions: As specified here

set

Format: `set(x,v0,v1,...,y)`
Constraints: `x` is an array; `v1`, `v2`, ... are scalars or vectors; `y` is an array
Description: array obtained from the array `x` by substituting its subarray of indexes `v0` from the 0-th dimension, of indexes `v1` from the 1-st dimension, ... with the array `y`

`set` is an array function

Exceptions: no

shift

Format: `shift(x,y,s)`
Constraints: `x` and `y` are arrays; `s` (optional) is a scalar
Description: array obtained by concatenating the array `y` to the array `x` at the left and removing the same number of elements from `x` at the right if `s==0` or it is not specified, or in reverse order, if `s==1`

`shift` is an array function

Exceptions: It must be possible concatenate `x` with `y` in terms of their sizes; if specified, `s` must be either `0` or `1`.

shuffle

Format: `shuffle(x)`
Constraints: `x` is an array
Description: array obtained by randomly permutating the elements of the array `x`

`shuffle` is an array function

Exceptions: no

sigmoid

Format: `sigmoid(vx,vy,x)`

Constraints: `vx` and `vy` are vectors; `x` is an array

Description: the `y` value corresponding to `x` over the sigmoid controlled by the points `vx[0],vy[0]` and `vx[1],vy[1]`

`sigmoid` is a mathematical function
`sigmoid` is an interpolation function

Exceptions: Both `vx` and `vy` must have two elements; `vx[0]` must be less than `vx[1]`.

sign

Format: `sign(x)`

Constraints: `x` is an array

Description: sign of `x`, i.e., `1` if `x > 0`, `-1` if `x < 0`, `0` if `x == 0`

`sign` is a mathematical function
`sign` is a monadic polymorphic function

Exceptions: no

sin

Format: `sin(x)`

Constraints: `x` is an array

Description: sine of `x`
`x` is expressed in radians.

`sin` is a mathematical function
`sin` is a monadic polymorphic function

Exceptions: no

size

Format: `size(x)`

Constraints: `x` is an array

Description: size of `x`
Equivalent to the operator `@x`. The value is zero if `x` is a scalar, and a vector otherwise.

`size` is an array function

Exceptions: no

sort

Format: `sort(x,s)`
Constraints: `x` is an array; `s` (optional) is a scalar
Description: array obtained by sorting the elements of the array `x`, either in direct order, if `s==0` or it is not specified, or in reverse order, if `s==1`

`sort` is an array function

Exceptions: If specified, `s` must be either `0` or `1`.

spline

Format: `spline(vx,vy,x)`
Constraints: `vx` and `vy` are vectors; `x` is an array
Description: the `y` value corresponding to `x` over the spline whose node points are in the vectors `vx` and `vy`

`spline` is a mathematical function
`spline` is an interpolation function

Exceptions: `vx` and `vy` must have the same number of elements, at least two; for each `i`, `vx[i]` must be less than `vx[i+1]`.

sqrt

Format: `sqrt(x)`
Constraints: `x` is an array
Description: square root of `x`

`sqrt` is a mathematical function
`sqrt` is a monadic polymorphic function

Exceptions: `x` must be positive.

sysTime

Format: `sysTime()`
Constraints:
Description: number of milliseconds from the simulation start

`sysTime` is [a control function]

Exceptions: no

tan

Format: `tan(x)`
Constraints: `x` is an array
Description: tangent of `x`
`x` is expressed in radians.

`tan` is [a mathematical function]
`tan` is [a monadic polymorphic function]

Exceptions: no

tDistribution

Format: `tDistribution(v,x,s)`
Constraints: `v` (optional) is a vector; `x` (optional) is an array; `s` (optional) is a scalar, either `0` or `1`
Description: t-distribution probability distribution of parameters `v=[p1]`, where `p1`=degrees of freedom (`p1>0`; default value: `5`): random number extracted from the distribution if `x` and `s` are not specified; value of the pdf, if `s==0` or the cdf, if `s==1`, at the point `x`

`tDistribution` is [a statistical function]
`tDistribution` is [a statistical distribution function]

Exceptions: All elements of `x` must be strictly positive; `s` must be either `0` or `1`.

transpose

Format: `transpose(x)`
Constraints: `x` is an array
Description: array obtained by transposing the array `x`
This function implements a generalized concept of transposition. If the argument is a scalar, then it is returned as it is. Otherwise the i-th dimension of the argument is cycled to the (i+1)-th dimension of the value.

`transpose` is an array function

Exceptions: no

uniform

Format: `uniform(v,x,s)`
Constraints: `v` (optional) is a vector; `x` (optional) is an array; `s` (optional) is a scalar, either `0`, or `1`, or `2`
Description: uniform (rectangular) probability distribution of parameters `v=[p1,p2]`, where `p1`=mean (default value: `0`), `p2`=stddev (default value: `1`): if `x` and `s` are not specified, random number extracted from the distribution; if `s==0`, pdf value at the point `x`; if `s==1`, cdf value at the point `x`; if `s==2`, inverse cdf value at the probability `x`

`uniform` is a statistical function
`uniform` is a statistical distribution function

Exceptions: `s` must be either `0`, `1`, or `2`; if `s==2` then all elements of `x` must be in the interval `[0,1]`.

wrap

Format: `wrap(x,y)`
Constraints: `x` and `y` are arrays (the constraints are specified here)
Description: `x` modulus `y` also dealing with negative values
While `mod(-2,5)==-2`, `wrap(-2,5)==3`.

`wrap` is a mathematical function
`wrap` is a diadic polymorphic function

Exceptions: As specified here

writeToNet

Format: `writeToNet(x)`
Constraints: `x` is an array
Description: send `x` to a Spacebrew network server

`writeToNet` is [a control function]

Exceptions: The Spacebrew server must be reachable.