Introduction to AUXLAB

version 1.49

AUXLAB is an integrated environment for sound generation, processing, visualization and audio playback, based on a programming language AUX (AUdio syntaX). AUXLAB allows audio manipulations, plotting of audio and non-audio data, playback of audio data, implementation of algorithms, creation of user interfaces and interfacing with programs written in other programming languages such as C/C++ and MATLAB.

Although AUXLAB is primarily intended for audio processing, it offers versatile functionalities allowing the users to compute and handle non-audio data.

While the syntax of AUXLAB resembles that of MATLAB, there are many unique syntax features of AUXLAB and differences in syntax conventions.

This is a work in progress, more so for documentation. The following is the list of features of the software that are not yet included in this documentation.

- User defined functions, general (how-to info) info and the Debugger feature
- Custom user interface module--you can design your own window components (dialog box, buttons, etc), so you can create your own program to be used for an experimental procedure with functionalities of AUX playback and graphics
- Complete descriptions on properties of graphics objects---Smilar to MATLAB, but with numerous differences to note
- "# Reserves"--special keywords assigned to frequently performed tasks or operations

BJ Kwon
bjkwon@gmail.com

http://auxlab.org
Last updated 10/2/2018

License & Credit

Languages used: C++11 with yacc/lex. All codes written with Win32 API

Internal libraries:

- sigproc: syntax tokenizing, parsing; signal generation and processing
- graffy: visualization and screen processing
- xcom: console handing, variable display, history window, managing debugger and coordinating with sigproc
- wavplay: audio playback
- auxp: private user-defined functions
- auxcon: module for the custom user interface environment

Source codes will be available in GitHub soon. If you can't wait, let me know.

External libraries (open source) utilized:

- FFTW 3.3.4
- libsndfile 1.0.26, libsamplerate 0.1.8; Erik de Castro Lopo
- ELLF (2014-10-03 release); Stephen L Moshier
- Win32++ 7.3; David Nash
- Bison 2.4.1
- Flex 2.5.4a

Developer: BJ Kwon
bjkwon@gmail.com

http://auxlab.org
Last updated 9/25/2018

System Requirements

- Windows 7, 8 and 10
- Minimum RAM: 128 MB
- Microsoft Visual C++ Redistributable for Visual Studio 2017

Data types in AUX

The following data types are used in AUX/AUXLAB:

NUL	null/empty data
SCAL	scalar
TEXT	text string
VCT	vector; array
AUD	audio
CEL	cell array
CLAS	class; structure
TSEQ	Time Sequence
HAUD	Handle to audio playback
HGO	Handle to graphic object

- Values can be either real or complex.
- Matrix is treated as a "grouped" VCT according to the row.
- The difference between VCT and AUD is that the latter has the information of 1) the sample rate, and 2) the time marker. In addition, one AUD object can have many chunks of audio in different times.
- Another difference: for some functions that do not allow negative values but could be useful in sound processing or computations are treated as an even-function for AUD (for example, the sqrt function), whereas for VCT, it is either an error or produce an imaginary value (i.e., sqrt(-1))

TSEQ: Time sequence

Introduction

A data type TSEQ is an array container consisting of a time point and the data corresponding to the time point.

Defining a TSEQ object

A TSEQ object has the following form:

```
[x][y]
```

where x is the time point vector in milliseconds and y is the data vector. Here x and y must have the same length. In this form, at each time point the data is a scalar. In general, the data do not need to be a scalar; but can be in any form. To define such time sequence,

```
[t1 | y1; t2 | y2; ...]
```

where tn and yn are time marker and the corresponding value array in any length. (note: this is not implemented yet as of AUXLAB 1.47).

Relative TSEQ

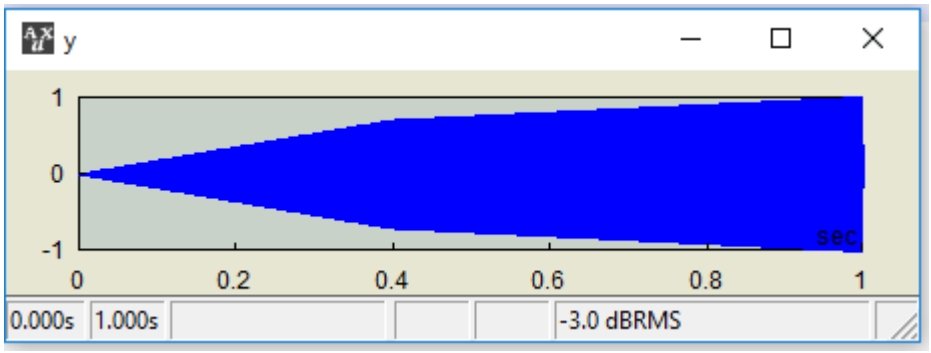
Sometimes it is very useful to have time values relative to another audio signal. In such cases, define a relative time sequence as follows:

```
[x;][y]
```

Example 1

The amplitude of a tone is scaled with a TSEQ, 0 at t=0, .7 at t=250ms, .3 at t=500ms, and 1 at t=1000ms. The multiplication operation with a TSEQ involves linear interpolation between specified time points.

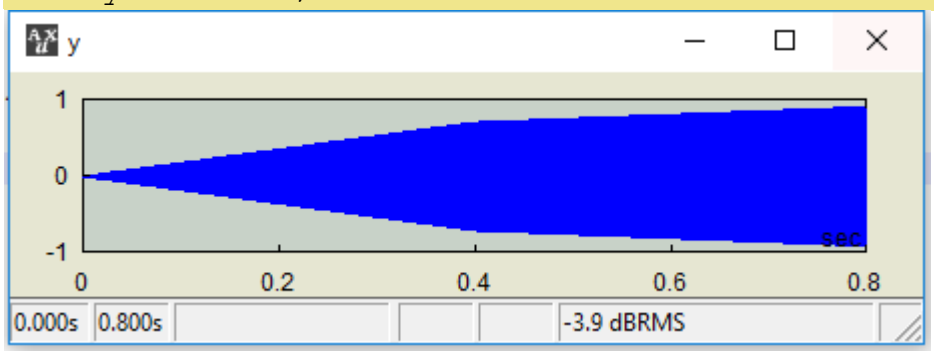
```
AUX> x = tone(500,1000);
AUX> ts = [0 250 500 1000][0 .7 .3 1];
AUX> y = ts * x;
```



Example 2

The same TSEQ as above but the audio signal with a differation duration. Is this what you want?

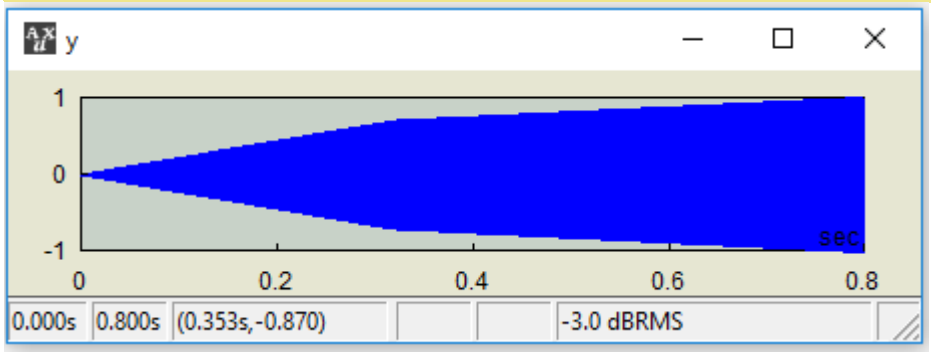
```
AUX> x = tone(500,800);
AUX> ts = [0 250 500 1000][0 .7 .3 1];
AUX> y = ts * x;
```



Example 3

If you wanted to scale the audio with the same relative time course as Example 1, then go with a relative TSEQ.

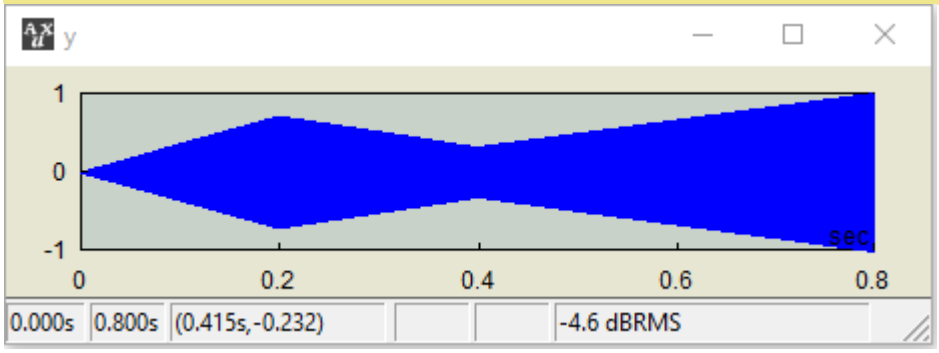
```
AUX> x = tone(500,800);
AUX> ts = [0 .25 .5 1;][0 .7 .3 1];
AUX> y = ts * x;
```



Example 4

To adjust the amplitude of the audio signal with a desired time course in terms of dB, use the @ operator:

```
AUX> x = tone(500,800);
AUX> ts = [0 .25 .5 1;][-100 -3 -10 0];
AUX> y = x @ ts;
```



Operators in AUX

In addition to arithmetic operators (+ - * / ^ %) and logical operators (&& || !), that you can find in most programming or scripting languages, AUX offers unique operators designed for audio signals, such as @ (amplitude scaling), >> (time-shift), ~ (time-compression/spectrum-expansion), -> (spectrum-shift), <> (change of duration), and # (change of pitch).

+ -

```
z = x + y
z = x - y
```

Arithmetic Plus or Minus

Commutative

Yes for + No for -

Data Types

Allowed	**Not allowed**
SCAL, VCT, AUD, TEXT, TSEQ, NUL	CLASS, CELL

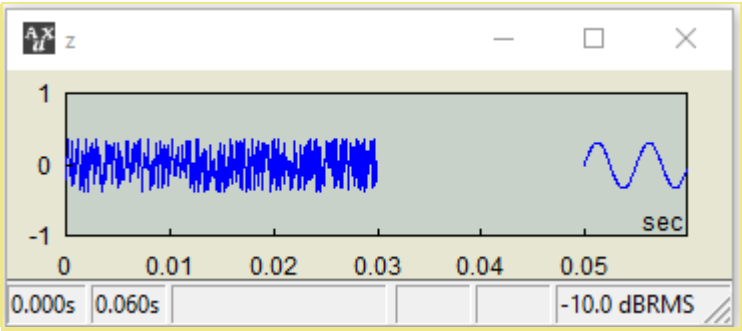
x	**y**	**z**
NUL	anything	anything
SCAL	anything	anything
VCT	VCT	VCT (see below)
VCT	AUD	AUD (see below)
TSEQ	array	not allowed

Notes

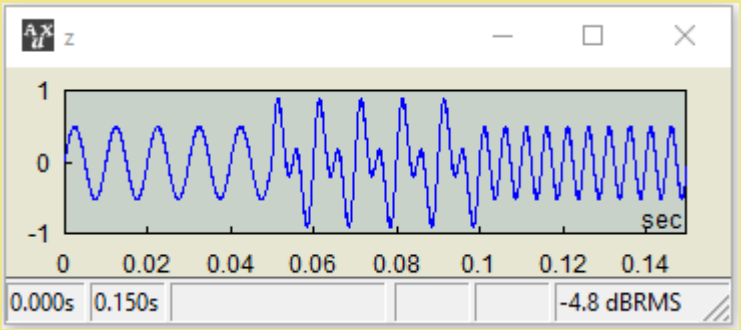
- In the case of SCAL + VCT or SCAL + AUD, the scalar is applied (added) to the entire array.
- In the case of VCT + VCT, if the lengths of operands are different, the operation takes place until the two values are available.
- In the case of VCT + AUD, if the lengths of operands are different, the operation takes place until the two values are available.
- In the case of AUD + AUD, the operation is time-based: i.e., if the signal is available at the particular time, the operation takes place.
- In the case of SCAL + TSEQ, the operation applies to the value of the TSEQ: i.e., the output TSEQ has the value added to SCAL at each time point.
- In the case of TSEQ + TSEQ, both must have the same number of time points (individual time points don't need to be the same).
- When both operands are grouped (i.e., matrix), both must have the equal number of groups(i.e., rows)

Examples

```
AUX> noise(30)@-10 + tone(200,10)@-10 >>50
z =
audio (0.0ms~30.0ms) (50.0ms~60.0ms)
```




```
AUX> tone(100,100)@-6 + tone(200,100)@-6 >>50
z =
audio (0.0ms~150.0ms)
```



* /

```
z = x * y
z = x / y
```

Arithmetic Multiplication or Division

Commutative

Yes for * No for /

Data Types

Allowed	**Not allowed**
SCAL, VCT, AUD, TEXT, TSEQ, NUL	CLASS, CELL

x	**y**	**z**
NUL	anything	anything
SCAL	anything	anything
VCT	VCT	VCT (see below)
VCT	AUD	AUD (see below)
TSEQ	AUD	AUD (special meaning; see below)
TSEQ	VCT	TSEQ

Notes

- In essence, this is "dot-multiplication," in the MATLAB language.
- In the case of SCAL * VCT or SCAL * AUD, the scalar is applied (multiplied) to the entire array.
- In the case of VCT * VCT, if the lengths of operands are different, the operation takes place until the two values are available.
- In the case of VCT * AUD, if the lengths of operands are different, the operation takes place until the two values are available.
- In the case of AUD * AUD, the operation is time-based: i.e., if the signal is available at the particular time, the operation takes place.
- In the case of SCAL * TSEQ, the operation applies to the value of the TSEQ: i.e., the output TSEQ has the value multiplied by SCAL at each time point.
- In the case of TSEQ * TSEQ, both must have the same number of time points (individual time points don't need to be the same).
- TSEQ * AUD multiplies each value of the audio signal with a linear interpolated version of TSEQ (see example below).
- When both operands are grouped (i.e., matrix), both must have the equal number of groups(i.e., rows)
- As of 9/20/2018, AUXLAB does not support matrix multiplication (in the mathematical sense). I will add the feature in future releases (it is not difficult to do it, anyway; we need to choose the operator symbol, though), if there are enough requests from users.

++

x ++ y

Append (Concatenate) `y` at the end of `x`

Commutative

No

Data Types

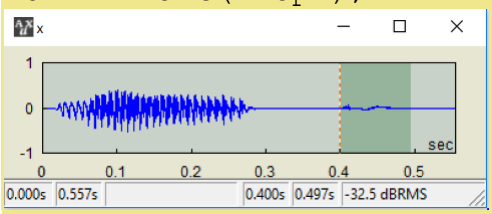
x	y
AUD	AUD

Notes

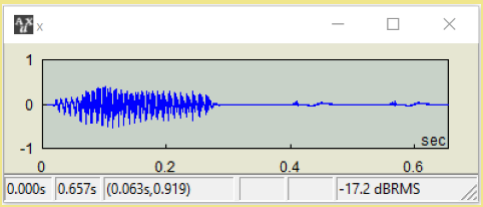
Examples

Open a .wav file "mop" and repeat the /p/ portion at the end.

```
AUX> x=wave("mop");
```




```
AUX> x=x++x(400~500);
```



:

a = x : y

Make an array a beginning from x to y with step of 1 or -1

a = x : y : z

Make an array a beginning from x to z with step of y

Commutative

No

Data Types

x	y	z
SCAL	SCAL	SCAL

Notes

- In x:y, the step size is either 1, if x < y, or -1, if x > y. Note that this is different from the MATLAB convention.

Examples

```
AUX> 1:5
ans = 1 2 3 4 5

     AUX> 5:1
     ans = 5 4 3 2 1

     AUX> 1:.5:3
     ans = 1 1.5 2 2.5 3
```

: (indexing)

```
a = x(:)
```

"Serialize" the matrix `x` and turn to a vector.

Commutative

N/A

Data Types

x	z
VCT	VCT

Notes

- If `x:` is a vector, it won't have any effect.

Examples

```
AUX> x=(1:6).matrix(2)
ans =
1 2 3
4 5 6
AUX> x(:)
ans = 1 2 3 4 5
```

~ (time-indexing)

x(t1 ~ t2)

Portion of audio `x` from `t1` ms to `t2` ms

Commutative

N/A

Data Types

x	t1	t2
AUD	SCAL	SCAL

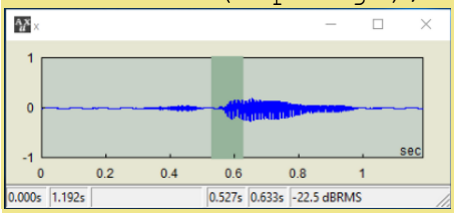
Notes

- If `t1 > t2`, the extracted signal is time-reversed.

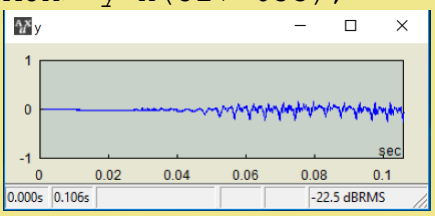
Examples

Open a .wav file "spring" and extract from 527 ms to 633 ms.

```
AUX> x=wave("spring");
```

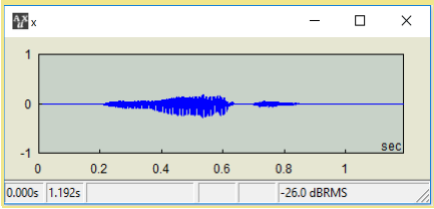



```
AUX> y=x(527~633);
```



To time-reverse the sound,

```
AUX> x=x(end~0);
```



• •

```
x = any expression with ..
x(id) = any expression with ..
x(id1:id2) = any expression with ..
x(id1:id2,jd1:jd2) = any expression with ..
x(t1~t2) = any expression with ..
```

Replicator; Replicate what's in the left hand side to the right hand side

Commutative
N/A
Data Types

This applies to multiple cases of expressions, not necessarily tied to one data type.

Notes

- Whatever is on the left hand side is being replicated to the right hand side with **..** . Therefore,
  `very_long_variable_name = any_function(..)`
  is equivalent to:
  `very_long_variable_name = any_function(very_long_variable_name)`
- The dot (.) notation also applies; therefore, the following is also an equivalent statement:
  `very_long_variable_name = ...any_function`
  as long as `any_function` supports the object function style.
- Conceptually, you might understand this as a sort of an advanced compound assignment operator:
  `x = x + 50` can be written as either `x += 50` or `x = .. + 50`
  On the other hand, `x = sqrt(x)` can't be written with a regular compound assignment operator, but `..` works perfectly: `x = sqrt(..)`

- This feature will save you from tedious typings or copy-pastings across left and right hand sides.

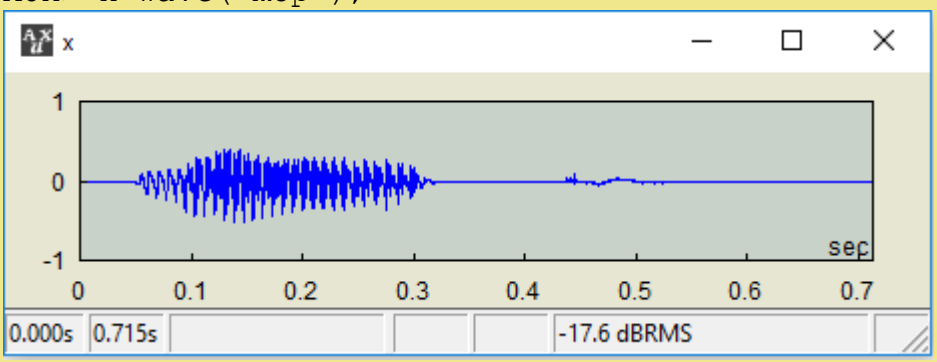
Examples

- You have a vector with a long variable name, if you append a value to it. You may write it in the vector form or use `++`
  `long_var_name = [long_var_name more_values]`
  `long_var_name = long_var_name ++ more_values` or
  `long_var_name ++= more_values`

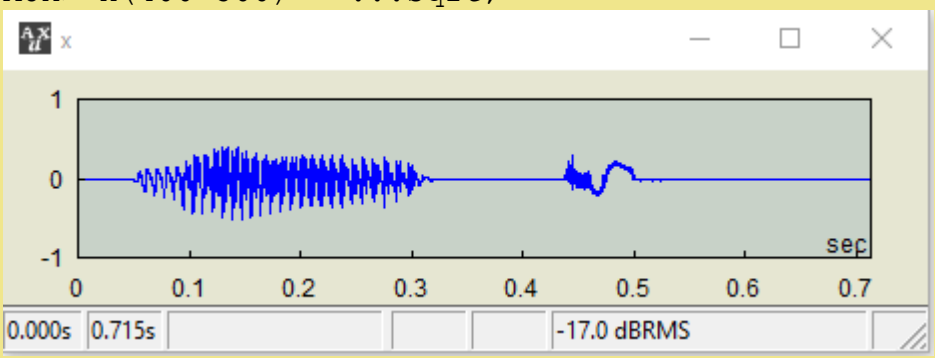
  But if values are inserted before it, you can't use ++=. Then, instead of typing long_var_name again on the RHS, just

`long_var_name = [values ..]` or
`long_var_name = values ++ ..` will do it.

- You want to modify a part of a signal: for example,
  `AUX> x=wave("mop");`



To distort the signal from 400 to 500 ms by applying sqrt, do this:
`AUX> x(400~500) = ...sqrt;`



- This is a more efficient way of coding. Internally, the expression `x(400~500) = x(400~500).sqrt` evaluates `x(400~500)` twice for the LHS and RHS, but the use of `..` bypasses one.

'

```
a = x'
```

Transpose; Swap the row and column of a matrix `x`.

Commutative

N/A

Data Types

x	z
VCT	VCT

Notes

Examples

```
AUX> x=(1:6).matrix(2)
ans =
1 2 3
4 5 6
AUX> x'
ans =
1 4
2 5
3 6
```

%

```
z = x % y
```

Remainder after division (modulo operation)

Commutative

No

Data Types

x	y	z
AUD, VCT, SCAL	SCAL	AUD, VCT, SCAL

Notes

- This is equivalent to `z=mod(x,y)` or `z=x.mod(y)`
- See mod for examples.

^

```
z = x ^ y
```

z is x raised to the power y

Commutative

No

Data Types

x	y	z
AUD, VCT, SCAL	SCAL	AUD, VCT, SCAL

Notes

-
- See ___ for examples.

@

z = x @ y

"At" operator; Amplitude scaling in terms of dB

Commutative
No
Data Types

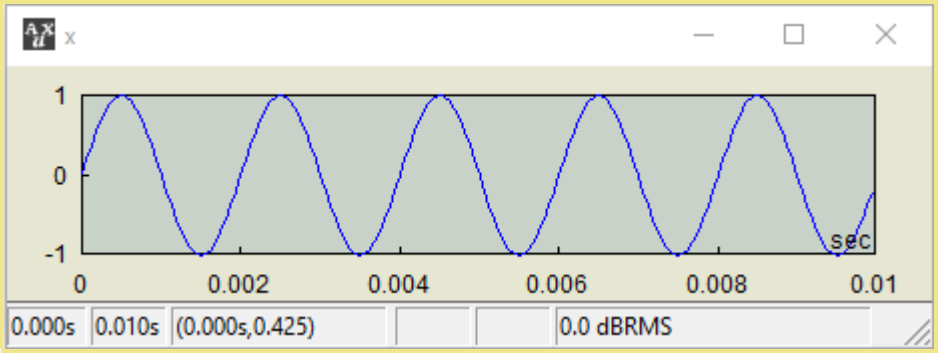
x	y	z
AUD	SCAL	AUD
AUD	TSEQ	AUD

Notes

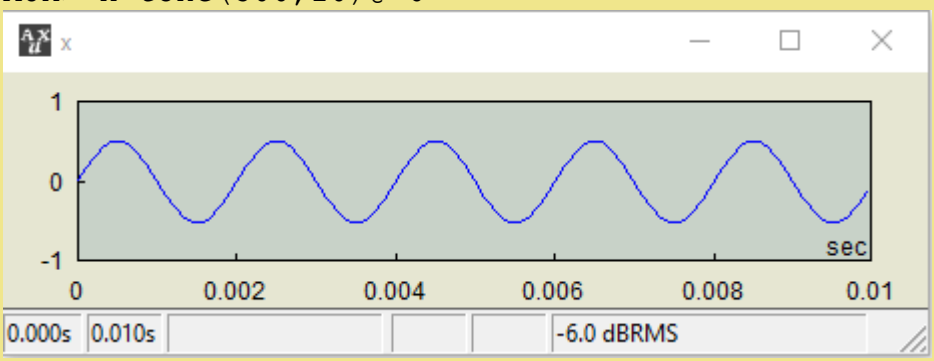
- The rms level of `z` is to be `y` dB, if `y` is a scalar.
- 0 dB is defined as the RMS level of a full-scale sinusoid. This means that the RMS of a full-scale square wave is to be 3 dB.

Examples

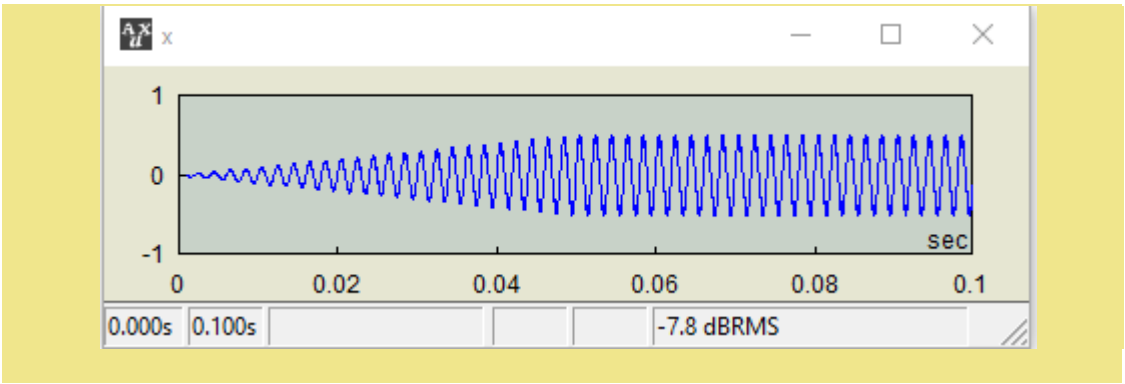
```
AUX> x=tone(500,10)
```




```
AUX> x=tone(500,10)@-6
```




```
AUX> x=tone(500,100)@[0 .5 1;][-100 -6 -6]
```


x
1
0
-1
sec
0
0.02
0.04
0.06
0.08
0.1
0.000s
0.100s
-7.8 dBRMS

>>

`z = x >> y`

Time-shift:Shift the audio signal `x` by `y` milliseconds

Commutative
No
Data Types

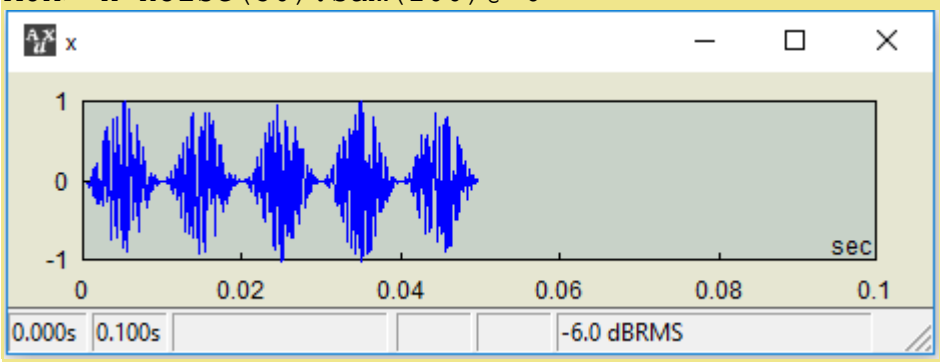
x	y	z
AUD	SCAL	AUD

Notes

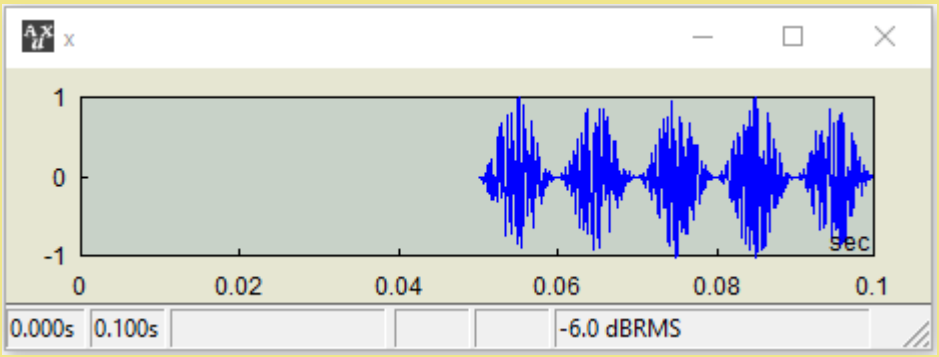
- If `x` is present in the interval `(t1, t2)`, `z` will be present in the interval `(t1+y, t2+y)`

Examples

```
AUX> x=noise(50).sam(100)@-6
```




```
AUX> x=noise(50).sam(100)@-6 >> 50
```



[;]

```
z = [x ; y]
```

Make a stereo signal `z` from `x` and `y`

Commutative
No
Data Types

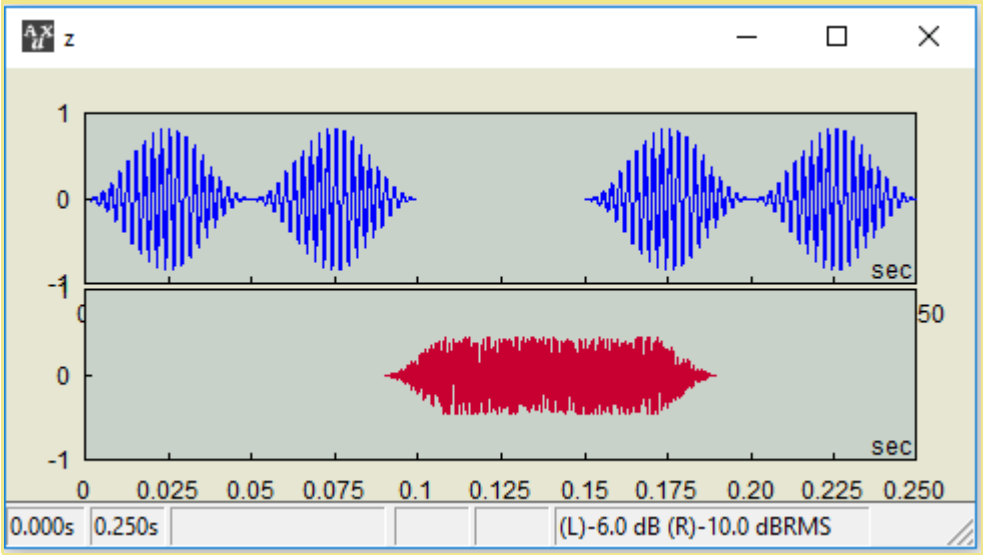
x	y	z
AUD	AUD	AUD

Notes

- `x` and `y` do not need to be the same length or even in the same time period.

Examples

```
AUX> x=tone(400,100).sam(20)@-6;
AUX> x = x+x>>150;
AUX> y=noise(100).ramp(50)@-10>>90;
AUX> z=[x;y]
z =
audio(L) (0.0ms~100ms) (150.0ms~250.0ms)
audio(R) (90.0ms~190ms)
```



~

```
z = x ~ y
```

Time-Frequency Scaling Operation; i.e., Time-Compress (or Frequency-Expand at the same time) an audio signal `x` by a factor of `y`

Commutative

No

Data Types

x	y	z
AUD	SCAL	AUD

Notes

- This simulates a playback of audio samples with a different rate of its original sample rate.

Examples

->

z = x -> y

Frequency-Shift Operator; Shift the spectrum of an audio signal `x` by `y` Hz

Commutative
No
Data Types

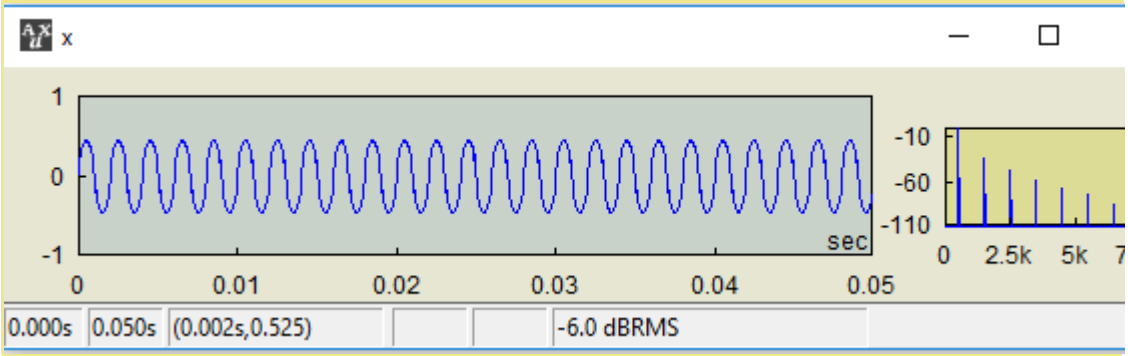
x	y	z
AUD	SCAL	AUD

Notes

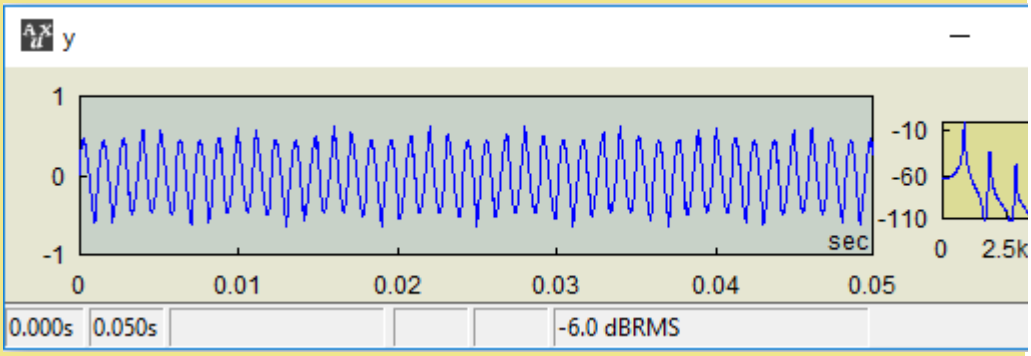
- This is equivalent to `z=movespec(x,y)` or `z=x.movespec(y)`
- This does not change the pitch of a harmonic tone `x` except for a sinesoid. Frequencies of the harmonics are shifted equally by `y` Hz, so it won't sound like a natural harmonic tone.

Examples

```
AUX> x=tone(500,50).sqrt@-6
```




```
AUX> y=x->330
```



Compound Assignment Operators in AUX

AUX supports compound assignment operators, also known as augmented operators. The following examples are self-explanatory, as their usage is the same in other programming languages such as C and Java:

```
+= -= *= /=
```

The following examples are unique in AUX:

```
++= @= @@= >>= ~= ->= <>= #=
```

+= -= *= /=

```
x += y
x -= y
x *= y
x /= y
```

Compound operators for arithmetic operations, + - * and /

Equivalent expressions

x = x + y
x = x - y
x = x * y
x = x / y

Commutative

No

Data Types

Allowed	**Not allowed**
SCAL, VCT, AUD, TEXT, TSEQ, NUL	CLASS, CELL

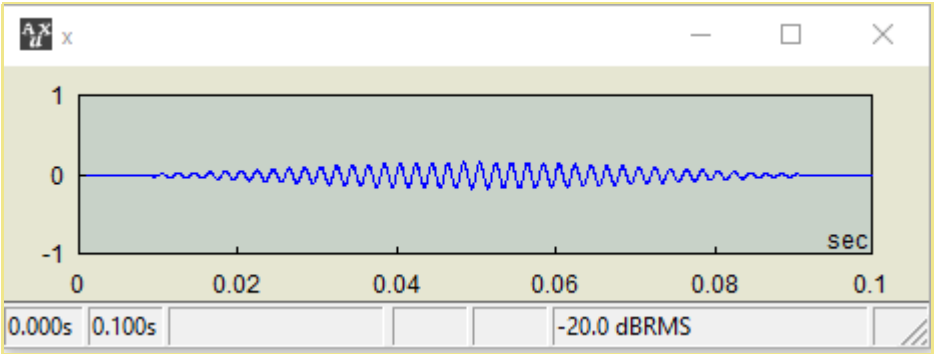
x	**y**	**x becomes**
NUL	T*	T
SCAL	T	T
VCT	VCT	VCT
AUD	VCT	AUD
VCT	AUD	AUD
TSEQ	VCT	TSEQ
TSEQ	AUD	not allowed
AUD	TSEQ	AUD

* T *means an allowed type.*

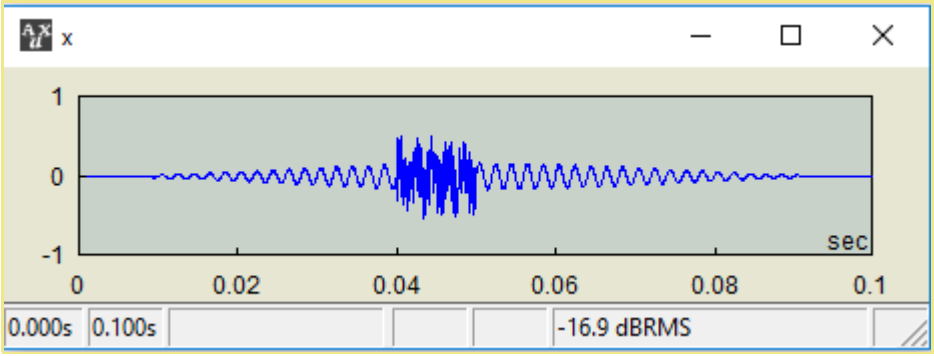
Notes

Examples

```
AUX> x=rand(5)
x =
0.5853 0.2218 0.6290 0.1231 0.2736
AUX> x*=10
x =
5.8531 2.2182 6.2896 1.2313 2.7363
AUX> x=tone(500,100).hann@-20
x =
audio (0.0ms~100.0ms)
```




```
AUX> y=noise(10)@-10>>40
y =
audio (40.0ms~50.0ms)
AUX> x += y
x =
audio (0.0ms~100.0ms)
```



++=

```
x ++= y
```

Compound operator for the append operation ++

Equivalent expression

x = x ++ y

Commutative

No

Data Types

x	**y**
AUD	AUD

Notes

Examples

@=

x @= y

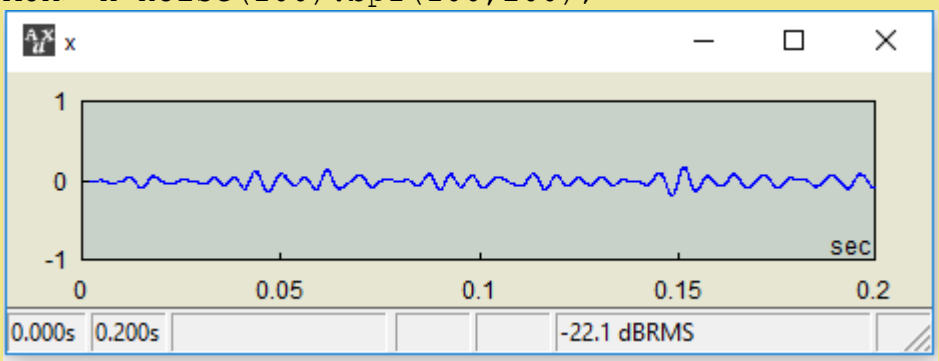
Compound operator for the level operator @

Equivalent expression
x = x @ y
Commutative
No
Data Types

x	y
AUD	SCAL

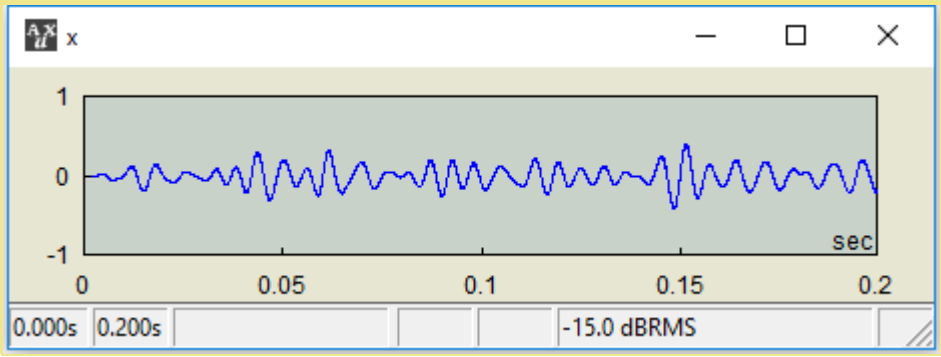
Notes
This is used to set the level at the target level y (with an absolute level)
Examples
Bandpass filter the white noise x

```
AUX> x=noise(100).bpf(100,200);
```



The RMS level of x, -22.1 dB, as seen in the plot, is simply a byproduct of filtering. To set it specifically at -15 dB, do the following:

```
AUX> x @= -15
```



@@=

x @= y

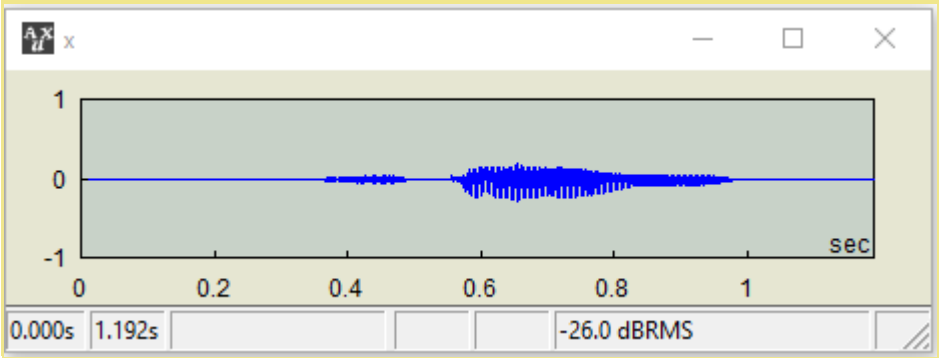
Compound operator for the level operator @@
Also known as relative level operator

Equivalent expression
`x = x @ x @ y`
Commutative
No
Data Types

x	y
AUD	SCAL

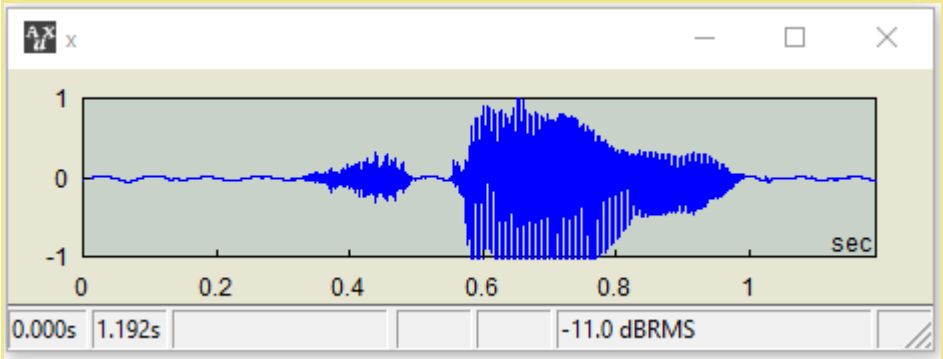
Notes
`x @ x @ y` literally means the signal `x` with the level at `y` dB re. `x`
In other words, the signal `x` with the level adjusted by `y` dB from its own rms.
Therefore, this is also known as an "incremental" level operator.
Examples
Open a .wav file. The RMS level of this audio happens to be -26 dB.

```
AUX> x=wave("spring")
```



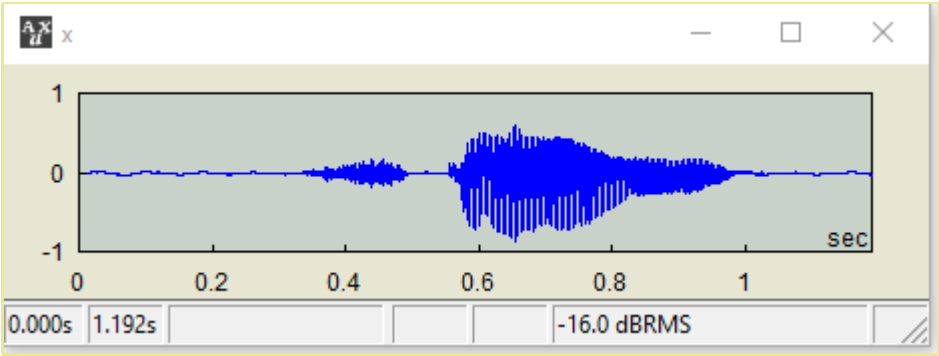
I want to increase its RMS level by 15 dB.

```
AUX> x @@= 15
```



Oops, we have a peak-clipping situation. A 15-dB increase was too much. Let's go down by 5 dB.

```
AUX> x @@= -5
```



This looks good!

>>=

x >>= y

Compound operator for the time-shift operator >>

Equivalent expression

`x = x >> y`

Commutative

No

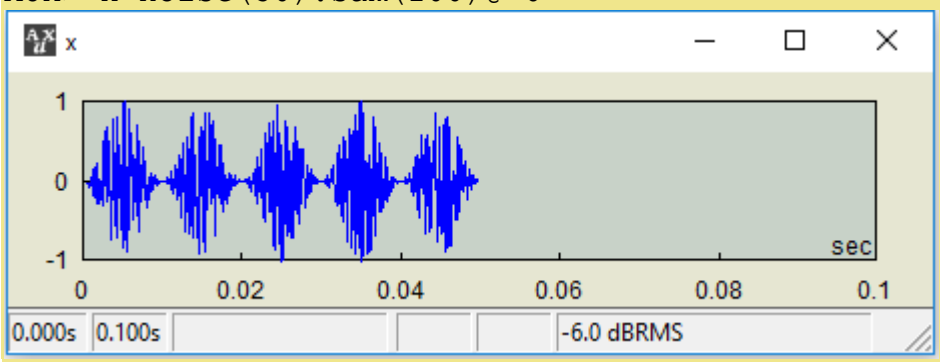
Data Types

x	y
AUD	SCAL

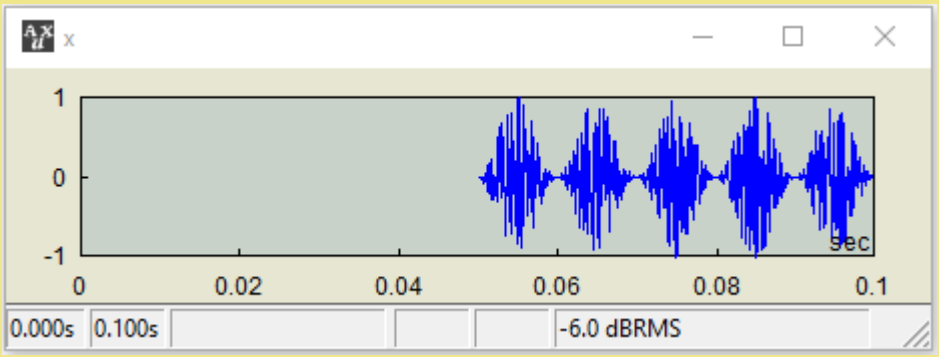
Notes

Examples

```
AUX> x=noise(50).sam(100)@-6
```




```
AUX> x >>= 50
```



~=

x ~= y

Compound operator for the time-frequency scale operator ~

Equivalent expression

`x = x ~ y`

Commutative

No

Data Types

x	y
AUD	SCAL

Notes

Examples

->=

x ->= y

Compound operator for the frequency-shift operator ->

Equivalent expression

`x = x -> y`

Commutative

No

Data Types

x	**y**
AUD	SCAL

Notes

Examples

Functions that generate a signal or vector

Here are the functions that generate a signal or vector with a given list of parameters. Notable example include tone (to create a pure tone), noise (to create white noise), wave (to read .wav file), and rand (to create an array of random numbers).

cell

```
y = cell ( n )
y = n.cell ( )
```

Create a cell array with specified blank elements

arg	type	Description	Unit or Value scale
`n`	SCL	Number of blank elements	

Outputs

- `y` is a cell array with blank elements.

Notes

Examples

See Also

dc

```
y = dc ( duration )
```

Generate a dc signal (constant amplitude at the full scale)

arg	type	Description	Unit or Value scale
`duration`	SCAL	duration of the signal to generate	

Outputs

- `y` is an audio signal (not really) of all values of one with the duration `duration` milliseconds.

Notes

- Although this is not technically "sound", this is treated as an audio signal.

Examples

See Also

ones | silence | zeros

fm

```
y = fm ( freq1,freq2,mod_rate,duration,phase[=0] )
```

Generate a frequency-modulated tone signal

arg	type	Description	Unit or Value scale
`freq1`	SCAL	FM swing from	
`freq2`	SCAL	FM swing to	
`mod_rate`	SCAL	how many times the frequency will swing	
`duration`	SCAL	duration of the signal to generate	
`phase`	SCAL	The initial modulation phase between 0 and 1	

Outputs

- `y` is a frequency-modulated tone.

Notes

Examples

See Also

gnoise

```
y = gnoise ( duration )
```

Generate a white noise signal with a Gaussian distribution

arg	type	Description	Unit or Value scale
`duration`	SCAL	duration of the signal to generate	

Outputs

- `y` is Gaussian white noise (random numbers from a Gaussian distribution).

Notes

- Unlike noise, the amplitude of gnoise is not bounded between -1 and 1. → Scale down the output appropriately to avoid clipping at ±1.

Examples

See Also

noise

input

```
y = input ( str )
y = str.input ( )
```

Prompt a user response.

arg	type	Description	Unit or Value scale
`str`	STR	message to display	

Outputs

- `y` is the string input from the user.

Notes

- This does not return until the user presses the Return/Enter key.

Examples

See Also

inputdlg

irand

```
y = irand ( n )
y = n.irand ( )
```

Create an integer random number, uniformly distributed in the interval [1, **n**]

arg	type	Description	Unit or Value scale
n	SCAL	the upper limit of the range	

Outputs

- **y** is an integer random number between 1 and **n**

Notes
Examples
See Also

rand | randperm

noise

```
y = noise ( duration )
```

Generate a white noise signal

arg	type	Description	Unit or Value scale
`duration`	SCAL	duration of the signal to generate	

Outputs

- `y` is uniform white noise (random numbers from a uniform distribution) with amplitude bound between -1 and 1.

Notes

- The random signal is generated from a uniform distribution.

Examples

See Also

gnoise

ones

```
y = ones ( n )
y = n.ones ( )
```

Create a non-audio array of all ones

arg	type	Description	Unit or Value scale
`n`	SCAL	Size of array to create	

Outputs

- `y` is an array of value one with size `n`

Notes

Examples

See Also

zeros

rand

```
y = rand ( sz )
y = sz.rand ( )
```

Create an array of random numbers, uniformly distributed between 0 and 1, with the size of **sz**

arg	type	Description	Unit or Value scale
sz	SCAL	the size of the array to create	

Outputs

- **y** is an array of random numbers

Notes

Examples

See Also

irand | randperm

randperm

```
y = randperm ( n )
y = n.randperm ( )
```

Random permutation of the integers from 1 to `n`

arg	type	Description	Unit or Value scale
`n`	SCAL	the upper limit of the range, or the size of the array to create	

Outputs

- `y` is an array of integers between 1 and `n` in a random order

Notes

Examples

See Also

irand | rand

silence

```
y = silence ( duration )
```

Generate a silence signal

arg	type	Description	Unit or Value scale
`duration`	SCAL	duration of the signal to generate	

Outputs

- `y` is AUDIO with zero values.

Notes

- zeros generates a non-audio vector; silence generates an audio signal

Examples

See Also

dc | zeros

sprintf

```
y = sprintf ( format,... )
```

Generate a formatted text

arg	type	Description	Unit or Value scale
`format`	TXT	String that contains the text to be writeen to the file, following the same convention as in C language.	
`...`	...	n/a	

Outputs

- `y` is TXT containing the formatted text.

Notes

Examples

See Also

fprintf

str2num

```
y = str2num ( str )
y = str.str2num ( )
```

Read a string and convert it to a numerical array.

arg	type	Description	Unit or Value scale
`str`	STR	String (it must not have a non-numerical character)	

Outputs

- `y` is an array from the string `str`.

Notes

- If there's a non-numerical character, an empty array will be returned.

Examples

See Also

eval | sprintf

tone

```
y = tone ( freq,duration,phase[=0] )
```

Generate a pure tone

arg	type	Description	Unit or Value scale
`freq`	SCAL or VCT	frequency	
`duration`	SCAL	duration	
`phase`	SCAL	initial phase (0 to 1)	

Outputs

- `y` is a pure tone.

Notes

- If `freq` is a two-element vector, a tone glide is generated with beginning and ending frequencies, as specified.

Examples

- tone(100, 50, .25) // 100-Hz, 50-ms tone with a starting phase of 90°

See Also

wave

```
y = wave ( filename )
```

Open a .wav file

arg	type	Description	Unit or Value scale
`filename`	STR	the name of the file to open.	

Outputs

- `y` is an audio signal retried from the file `filename`.

Notes

- If the extension is not specified, .wav is used.
- If path is not included, the current folder (the same directory as AUXLAB)

Examples

- wave ("c:\soundData\specialnoise")

See Also

wavwrite

zeros

```
y = zeros ( n )
y = n.zeros ( )
```

Create a non-audio array of all zeros

arg	type	Description	Unit or Value scale
`n`	SCAL	Size of array to create	

Outputs

- `y` is a non-audio array with `n` zeros

Notes

- zeros generates a non-audio vector; silence generates an audio signal

Examples

See Also

ones

Functions that produce an signal or vector based on an existing signal or vector

These are the functions that produce a modified version of an existing signal or vector with a given list of parameters. They include functions for filtering (filt, filtfilt, lpf, hpf, bpf, bsf), windowing (ramp, blackman, hann, hamming), or altering features of audio signals, such as time-scaling tscale, frequency-scaling fscale or spectrum-shifting movespec.

audio

```
y = audio ( x )
y = x.audio ( )
```

Convert a non-audio vector to an audio signal

arg	type	Description	Unit or Value scale
`x`	VCT	non-audio vector	

Outputs

- `y` is a converted audio signal of `x`.

Notes

- This is used when hacking aux functions requiring audio signal arguments.
- In AUX, all audio signals should be bound between -1 and 1. If the maximum of `x` exceeds the boundary, clipping will occur.

Examples

See Also

vector

blackman

```
y = blackman ( x,alpha )
y = x.blackman ( alpha )
```

Apply a blackman window

arg	type	Description	Unit or Value scale
`x`	AUDIO or VCT	Signal to apply the windowing to	
`alpha`	SCAL	α	

Outputs

- `y` is `x` with a blackman window applied.

Notes

Examples

- blackman(ones(128),.4) or ones(128).blackman(.16): 128-pt blackman window with α=.16
- blackman(dc(100)) or dc(100).blackman(.16): 100-ms blackman window α=.16
- blackman(tone(500,100)) or tone(500,100).blackman(.16): blackman window α=.16 applied to a 500-Hz, 100-ms tone

See Also

blackman | hamming

bpf

```
y = bpf (
x,fcut1,fcut2,order[=8],type[=1],dBpass[=.5],dBstop[=-40]
)
y = x.bpf (
fcut1,fcut2,order[=8],type[=1],dBpass[=.5],dBstop[=-40] )
```

IIR, band-pass filtering; Apply a band-pass filter to the audio signal

arg	type	Description	Unit or Value scale
x	AUDIO	audio signal	
fcut1	SCAL	cut-off frequency1	
fcut2	SCAL	cut-off frequency2	
order	SCAL	order of the IIR filter	
type	SCAL	IIR filter type (1: Butterworth, 2: Chebyshev, 3: Elliptic)	
dBpass	SCAL	Passband ripple allowed	
dBstop	SCAL	Stopband attenuation	

Outputs

- **y** is a bandpass-filterd version of **x**.

Notes

- The output is not normalized; i.e., the rms of **x** is adjusted according to the filter gain.
- **fcut1** and **fcut2** should be less than the Nyquist frequency.
- IIR filter coefficients are designed by the specification **requested** in the argument list, **but not guranteed**. The user is responsible for making sure the output follows the spec.

Examples

- bpf(x, 2000, 4000)

See Also

hpf | bpf | bsf

Algorithm

ELLF Digital Filter Calculator

bsf

```
y = bsf (
x,fcut1,fcut2,order[=8],type[=1],dBpass[=.5],dBstop[=-40]
)
y = x.bsf (
fcut1,fcut2,order[=8],type[=1],dBpass[=.5],dBstop[=-40] )
```

IIR, band-stop filtering; Apply a band-stop filter to the audio signal

arg	type	Description	Unit or Value scale
x	AUDIO	audio signal	
fcut1	SCAL	cut-off frequency1	
fcut2	SCAL	cut-off frequency2	
order	SCAL	order of the IIR filter	
type	SCAL	IIR filter type (1: Butterworth, 2: Chebyshev, 3: Elliptic)	
dBpass	SCAL	Passband ripple allowed	
dBstop	SCAL	Stopband attenuation	

Outputs

- **y** is a bandstop-filterd version of **x**.

Notes

- The output is not normalized; i.e., the rms of **x** is adjusted according to the filter gain.
- **fcut1** and **fcut2** should be less than the Nyquist frequency.
- IIR filter coefficients are designed by the specification **requested** in the argument list, **but not guranteed**. The user is responsible for making sure the output follows the spec.

Examples

- bsf(x, 2000, 4000)

See Also

lpf | hpf | bpf

Algorithm

ELLF Digital Filter Calculator

filt

```
y = filt ( x,num,den )
y = x.filt ( num,den )
```

1-D digital filter

arg	type	Description	Unit or Value scale
x	AUD or VECT	Input data	
num	VECT	Numerator coefficients of rational transfer function	
den	VECT	Denominator coefficients of rational transfer function	

Outputs

- **y** is filtered data.

Notes

- For a grouped or "2-D" array, filtering takes place for each group.

Examples

See Also

filtfilt

filtfilt

```
y = filtfilt ( x,num,den )
y = x.filtfilt ( num,den )
```

Zero-phase digital filtering

arg	type	Description	Unit or Value scale
`x`	AUD or VECT	Input data	
`num`	VECT	Numerator coefficients of rational transfer function	
`den`	VECT	Denominator coefficients of rational transfer function	

Outputs

- `y` is filtered data.

Notes

- For a grouped or "2-D" array, filtering takes place for each group.

Examples

See Also

filt

fscale

```
y = fscale ( x,freq )
y = x.fscale ( freq )
y = x # freq
```

Increase/decrease the pitch of the signal without changing the duration

arg	type	Description	Unit or Value scale
`x`	AUD	audio signal	
`freq`	VCT	The desired pitch change in semitones	

Outputs

- `y` is audio with the same duration as `x`

Notes

Examples

See Also

tscale | movespec

Algorithm

First the signal time is rescaled with the phase vocoder(Flanagan & Golden, "Phase Vocoder" 1966); then the signal is resampled to equalize the duration.

hamming

```
y = hamming ( x )
y = x.hamming ( )
```

Apply a hamming window

arg	type	Description	Unit or Value scale
x	AUDIO or VCT	Signal to apply the windowing to	

Outputs

- y is x with a hamming window applied.

Notes

Examples

- hamming(ones(128)) or ones(128).hann: 128-pt hamming window
- hamming(dc(100)) or dc(100).hamming: 100-ms hamming window
- hamming(tone(500,100)) or tone(500,100).hamming: hamming window applied to a 500-Hz, 100-ms tone

See Also

hann | blackman

hann

```
y = hann ( x )
y = x.hann ( )
```

Apply a hann (hanning) window

arg	type	Description	Unit or Value scale
x	AUDIO or VCT	Signal to apply the windowing to	

Outputs

- y is x with a hanning window applied.

Notes

Examples

- hann(ones(128)) or ones(128).hann: 128-pt hann window
- hann(dc(100)) or dc(100).hann: 100-ms hann window
- hann(tone(500,100)) or tone(500,100).hann: hann window applied to a 500-Hz, 100-ms tone

See Also

hamming | blackman

hpf

```
y = hpf ( x,fcut,order[=8],type[=1],dBpass[=.5],dBstop[=-40] )
y = x.hpf ( fcut,order[=8],type[=1],dBpass[=.5],dBstop[=-40] )
```

IIR, high-pass filtering; Apply a high-pass filter to the audio signal

arg	type	Description	Unit or Value scale
x	AUDIO	audio signal	
fcut	SCAL	cut-off frequency	
order	SCAL	order of the IIR filter	
type	SCAL	IIR filter type (1: Butterworth, 2: Chebyshev, 3: Elliptic)	
dBpass	SCAL	Passband ripple allowed	
dBstop	SCAL	Stopband attenuation	

Outputs

- **y** is a highpass-filterd version of **x**.

Notes

- The output is not normalized; i.e., the rms of **x** is adjusted according to the filter gain.
- **fcut** should be less than the Nyquist frequency.
- IIR filter coefficients are designed by the specification **requested** in the argument list, **but not guranteed**. The user is responsible for making sure the output follows the spec.

Examples

- hpf(x, 2000)

See Also

lpf | bpf | bsf

Algorithm

ELLF Digital Filter Calculator

interp

```
y = interp ( x,factor )
y = x.interp ( factor )
y = x ~ factor
```

Interpolate the array. This is equivalent to spectrum compression / time expansion (or spectrum expansion / time compression) for an audio signal.

arg	type	Description	Unit or Value scale
`x`	AUD or VCT	Audio signal	
`factor`	SCL	The factor to compress or expand the signal in time and frequency	

Outputs

- `y` is AUD or VCT

Notes

- This changes the number of samples in an audio signal. Internally the signal is treated as if it was resampled. Note that this is not actual resampling unless the sample rate in AUXLAB was adjusted.

Examples

See Also

Algorithm

The library from Secret Rabbit Code (aka libsamplerate) is used.

lpf

```
y = lpf ( x,fcut,order[=8],type[=1],dBpass[=.5],dBstop[=-40] )
y = x.lpf ( fcut,order[=8],type[=1],dBpass[=.5],dBstop[=-40] )
```

IIR, low-pass filtering; Apply a low-pass filter to the audio signal

arg	type	Description	Unit or Value scale
x	AUDIO	audio signal	
fcut	SCAL	cut-off frequency	
order	SCAL	order of the IIR filter	
type	SCAL	IIR filter type (1: Butterworth, 2: Chebyshev, 3: Elliptic)	
dBpass	SCAL	Passband ripple allowed	
dBstop	SCAL	Stopband attenuation	

Outputs

- **y** is a lowpass-filterd version of **x**.

Notes

- The output is not normalized; i.e., the rms of **x** is adjusted according to the filter gain.
- **fcut** should be less than the Nyquist frequency.
- IIR filter coefficients are designed by the specification **requested** in the argument list, **but not guranteed**. The user is responsible for making sure the output follows the spec.

Examples

- lpf(x, 2000)
- noise(500).lpf(500)

See Also

hpf | bpf | bsf

Algorithm

ELLF Digital Filter Calculator

matrix

```
y = matrix ( x,m )
y = x.matrix ( m )
```

Turns an array into a matrix

arg	type	Description	Unit or Value scale
`x`	VCT or AUD	input array	
`m`	SCAL	Number of rows (also known as the number of "groups")	

Outputs

- `y` is a matrix with `m` rows.

Notes

- The length of `x` must be a multiple of `m`. Number of columns is (length of `x`) / `m`

Examples

- (1:12).matrix(3) →
- 1 2 3 4
  5 6 7 8
  9 10 11 12

See Also

movespec

```
y = movespec ( x,freq )
y = x.movespec ( freq )
y = x -> freq
```

Shift the spectrum in the frequency domain

arg	type	Description	Unit or Value scale
`x`	AUD	audio signal	
`freq`	VCT	how much the signal is shifted by frequency	

Outputs

- `y` is audio with the same duration as `x`

Notes

- This function shifts the frequencies all harmonic components; i.e., this is not the same as shifting the pitch.

Examples

See Also

fscale

ramp

```
y = ramp ( x,dur_ramp )
y = x.ramp ( dur_ramp )
```

Smooth out beginning and ending portions of the audio signal; Apply the cosine square envelope

arg	type	Description	Unit or Value scale
`x`	AUDIO	audio signal	
`dur_ramp`	SCAL	ramping duration (i.e., duration to smooth out)	

Outputs

- `y` is an audio signal with fade-in and fade-out.

Notes

- Ramping is applied only to the portions specified by `dur_ramp` . Windowing functions, such as hann hamming or blackmand, apply the window for the whole duration.

Examples

- tone(440,100).ramp(40)

See Also

hann | hamming | blackman

sam

```
y = sam ( x,rate,depth[=1],phase[=0] )
y = x.sam ( rate,depth[=1],phase[=0] )
```

Apply Sinusoidal-Amplitude-Modulation to an audio signal.

arg	type	Description	Unit or Value scale
x	AUD	audio signal	
rate	SCAL	modulation frequency	
depth	SCAL	degree of modulation, 0 (no modulation) to 1 (full modulation)	
phase	SCAL	initial AM phase (0 to 1)	

Outputs

- **y** is AUDIO with sinusoidally amplitude modulation of **x**

Notes

Examples

See Also

sort

```
y = sort ( x,order[=1] )
y = x.sort ( order[=1] )
```

Sort the input array

arg	type	Description	Unit or Value scale
`x`	VCT	array to sort	
`order`	SCL	Direction: positive value for ascending (default); negative value for descending	

Outputs

- `y` is the sorted array of `x`. If `x` is a matrix, each row is sorted.

Notes

- Row-wise operation

Examples

- Given y = [3 1 8 -2],
  y.sort → [-2 1 3 8].

  Given y =
  [3 1 8 -2
  3 5 2 2],
  y.sort →
  [-2 1 3 8
  2 2 3 5]
  y.sort(-1) →
  [8 3 1 -2
  5 3 2 2].

See Also

tscale

```
y = tscale ( x,freq )
y = x.tscale ( freq )
y = x <> freq
```

Increase/decrease the duration of the signal without affecting the spectrum

arg	type	Description	Unit or Value scale
`x`	AUD	audio signal	
`freq`	VCT	The desired change of duration in ratio (> 1 to make longer or < 1 for shorter)	

Outputs

- `y` is audio with the same frequency as `x`

Notes

Examples

See Also

fscale | movespec

Algorithm

The algorithm is based on Dan Ellis's matlab code (http://www.ee.columbia.edu/ln/rosa/matlab/pvoc/), which was originally based on (Flanagan & Golden, "Phase Vocoder" 1966). This function in AUXLAB has minor modifications from Dan Ellis's code 1) adjustment of shot-term RMS to reduce unwanted fluctuations after processing and to adjust the duration properly.

vector

```
y = vector ( x )
y = x.vector ( )
```

Convert an audio signal to a non-audio vector

arg	type	Description	Unit or Value scale
x	AUDIO		

Outputs

- y is

Notes

- This is used when hacking some aux functions requiring a non-audio vector argument

Examples

See Also

audio

Functions that perform computation with the given signal or vectoror display properties of the signal.

These are the functions that perform computation with an existing signal or vector with a given list of parameters, or display the properties associated with the signal. Some are relevant to only audio signals, while others are applicable both audio and non-audio. They include functions for the rms-level (rms), duration (dur), the timing information (begint, endt), statistics (mean, std), length, size).

begint

```
y = begint ( x )
y = x.begint ( )
```

Get the begin time of an audio (the same as tmark)

arg	type	Description	Unit or Value scale
`x`	AUD	audio input	

Outputs

- `y` is a T-sequence showing the begin time of an audio signal `x`. `y` is a constant if there's a single signal chunk and tmark is 0.

Notes

- Operation executed by chunk

Examples

- tone(40,100).begint → 0 (constant)
- (tone(40,100)>>1000).begint → 0: 0 (time sequence)

See Also

dur | endt

cumsum

```
y = cumsum ( x )
y = x.cumsum ( )
```

Cumulative sum

arg	type	Description	Unit or Value scale
x	SCAL	the array	

Outputs

- y is the cumulative sum of x.

Notes

- If x is a matrix, y is a matrix of row-wise operations

Examples

- [1 2 3 4 5].cumsum → [1 3 6 10 15]
- (1:12).matrix(3).cumsum →
- ans =
  1 3 6 10
  5 11 18 26
  9 19 30 42

See Also

diff | reshape

diff

```
y = diff ( x,n[=1] )
y = x.diff ( n[=1] )
```

Calculates the n-th difference between adjacent elements of **x**

arg	type	Description	Unit or Value scale
x	VCT or AUDIO	the array or signal	
n	SCAL	skip count	

Outputs

- **y** is the array of n-th order difference.

Notes

- If **x** is a matrix, **y** is a matrix of row-wise operations

Examples

- a=1:5; a.diff → [1 1 1 1]
- a.diff(2) → [2 2 2]
- s=(1:12).matrix(3); s(2,:) *= 10; ss.diff →
  ans =
  1 1 1
  10 10 10
  1 1 1

See Also

cumsum

dur

```
y = dur ( x )
y = x.dur ( )
```

Get the duration of an audio

arg	type	Description	Unit or Value scale
`x`	AUD	audio input	

Outputs

- `y` is a T-sequence showing the duration of audio `x`. `y` is a constant if there's a single signal chunk and tmark is 0.

Notes

- Operation executed by chunk

Examples

- tone(40,100).dur → 100 (constant)
- (tone(40,100)>>1000).dur → 1000: 100 (time sequence)

See Also

begint | endt

endt

```
y = endt ( x )
y = x.endt ( )
```

Get the end time of an audio

arg	type	Description	Unit or Value scale
x	AUD	audio input	

Outputs

- y is a T-sequence showing the end time of an audio signal x. y is a constant if there's a single signal chunk and tmark is 0.

Notes

- Operation executed by chunk

Examples

- tone(40,100).endt → 100 (constant)
- (tone(40,100)>>1000).endt → 1000: 1100 (time sequence)

See Also

begint | dur

envelope

```
y = envelope ( x )
y = x.envelope ( )
```

The Hilbert envelope (the magnitude of the analytic signal)

arg	type	Description	Unit or Value scale
`x`	AUDIO	the audio signal	

Outputs

- `y` is the Hilbert envelope of `x`

Notes

Examples

See Also

hilbert

fft

```
y = fft ( x,n[=(size_of_x)] )
y = x.fft ( n[=(size_of_x)] )
```

Computes the FFT of the array

arg	type	Description	Unit or Value scale
`x`	VCT or AUDIO	the array	
`n`	SCAL	FFT size	

Outputs

- `y` is complex array of size `n`

Notes

Examples

See Also

ifft

Algorithm

FFTW

hilbert

```
y = hilbert ( x )
y = x.hilbert ( )
```

Computes the hilbert transform of the array--90 ° shift version; the imaginary part of the analytic signal

arg	type	Description	Unit or Value scale
`x`	AUDIO	the audio signal	

Outputs

- `y` is the 90 °-shift version of `x`.

Notes

Examples

See Also

envelope

ifft

```
y = ifft ( cx )
y = cx.ifft ( )
```

Computes the inverse FFT

arg	type	Description	Unit or Value scale
`cx`	complex VCT	complex array	

Outputs

- `y` is the inverse FFT of `cx`

Notes

Examples

See Also

fft

Algorithm

FFTW

left

```
y = left ( x )
y = x.left ( )
```

Extract the left channel from a stereo audio

arg	type	Description	Unit or Value scale
`x`	AUDIO	audio signal	

Outputs

- `y` is the left channel of `x`.

Notes

- If `x` is not stereo, the function returns `x`

Examples

- [.2*tone(400,100); .1*noise(300)].left // the 400-Hz tone channel is extracted

See Also

length

```
y = length ( x )
y = x.length ( )
```

Get the length of the array; for a 2-D array, it returns the entire length, i.e., the length of the "serialized" version of the array.

arg	type	Description	Unit or Value scale
x	STR AUD VCT SCL CEL	input array	

Outputs

- For a non-audio variable or a contiguous audio signal, **y** is a constant scalar showing the length of the array. If **x** has null portions, **y** is a time sequence showing the length of each segment.

Notes

- If **x** is an audio signal, it must not be a stereo signal.

Examples

See Also

max

```
[y, id] = max ( x1,x2,... )
[y, id] = x1.max ( x2,... )
```

Maximum element of an array if there is a single array argument; the maximum value from all values in the arguments

arg	type	Description	Unit or Value scale
`x1`	SCAL or VCT		
`x2`	SCAL or VCT		
`...`	...	...	

Outputs

- `y` is the maximum element.
- `id` is the index of the maximum element found first.

Notes

- Row-wise operation

Examples

- a=[8 2 4 6]; max(a) or a.max → 8
- b=[3 9 -1]; max(a,b) or a.max(b) → 9
- Multiple output [a,b]=(1:10:120).matrix(3).max() returns
- a=
  31
  71
  111

  b=
  4
  4
  4

See Also

min

min

```
[y, id] = min ( x1,x2,... )
[y, id] = x1.min ( x2,... )
```

Minimum element of an array if there is a single array argument; the minimum value from all values in the arguments

arg	type	Description	Unit or Value scale
`x1`	SCAL or VCT		
`x2`	SCAL or VCT		
`...`	...	...	

Outputs

- `y` is the minimum element.
- `id` is the index of the minimum element found first.

Notes

Examples

- a=[8 2 4 6]; min(a) or a.min → 2
- b=[3 0 -1]; min(a,b) or a.min(b) → -1
- Multiple output [a,b]=(1:10:120).matrix(3).max() returns
- a=
  1
  41
  81

  b=
  1
  1
  1

See Also

max

right

```
y = right ( x )
y = x.right ( )
```

Extract the right channel from a stereo audio

arg	type	Description	Unit or Value scale
`x`	AUDIO	audio signal	

Outputs

- `y` is the right channel of `x`.

Notes

- If `x` is not stereo, the function returns `x`

Examples

- [.2*tone(400,100); .1*noise(300)].right // the noise signal is extracted

See Also

rms

```
y = rms ( x )
y = x.rms ( )
```

Calculates the rms energy in dB

arg	type	Description	Unit or Value scale
x	AUDIO	the signal	

Outputs

- y is SCAL

Notes

- In AUX, by definition, the rms of a sinusoid with the full magnitude is 0 dB.

Examples

- The rms of a sinusoid with the magnitude half of the full scale is -6 dB.

See Also

size

```
y = size ( x )
y = x.size ( )
```

Get the dimension of the matrix.

arg	type	Description	Unit or Value scale
`x`	STR AUD VCT SCL CEL	input array	

Outputs

- `y` is a two-element vector showing the count of row and column, respectively.

Notes

- For an array, first element of the output is one.

Examples

See Also

length

std

```
y = std ( x,w[=0] )
y = x.std ( w[=0] )
```

Calculates the standard deviation

arg	type	Description	Unit or Value scale
`x`	VCT or AUDIO	the array	
`w`	SCAL	weight, 0: normalized by the (size-1) of #p1, 1: normalized by the size of #p1	

Outputs

- `y` is SCAL or grouped SCAL

Notes
Examples
See Also

sum

sum

```
y = sum ( x )
y = x.sum ( )
```

Calculates the sum of the array elements

arg	type	Description	Unit or Value scale
x	VCT or AUDIO	the array	

Outputs

- y is

Notes

Examples

See Also

Mathematical Functions

These are the collections of mathematical functions.

abs

```
y = abs ( x )
y = x.abs ( )
```

Absolute value or complex magnitude

arg	type	Description	Unit or Value scale
x	SCAL or VCT or AUDIO	n/a	

Outputs

- y is the absolute value for a real number, or the magnitude for a complex number

Notes

Examples

See Also

acos

```
y = acos ( x )
y = x.acos ( )
```

Inverse cosine in radians

arg	type	Description	Unit or Value scale
`x`	SCAL, VCT or AUDIO	n/a	

Outputs

- `y` is the Inverse Cosine, $\cos^{-1}$, of `x`

Notes

Examples

See Also

sin | cos | tan | asin | atan

angle

```
y = angle ( Z )
y = Z.angle ( )
```

Returns the phase angles

arg	type	Description	Unit or Value scale
`Z`	SCAL or VCT or AUDIO	complex value(s)	

Outputs

- `y` is the phase angles of `Z` in radian.

Notes
Examples
See Also

abs | atan

asin

```
y = asin ( x )
y = x.asin ( )
```

Inverse sine in radians

arg	type	Description	Unit or Value scale
x	SCAL, VCT or AUDIO	n/a	

Outputs

- **y** is the Inverse Sine, $\sin^{-1}$, of **x**

Notes

Examples

See Also

sin | cos | tan | acos | atan

atan

```
y = atan ( x )
y = x.atan ( )
```

Inverse tangent in radians

arg	type	Description	Unit or Value scale
x	SCAL, VCT or AUDIO	n/a	

Outputs

- **y** is the Inverse Tangent, $\tan^{-1}$, of **x**

Notes

Examples

See Also

sin | cos | tan | asin | acos

ceil

```
y = ceil ( x )
y = x.ceil ( )
```

Round toward positive infinity

arg	type	Description	Unit or Value scale
x	SCAL, VCT or AUDIO	n/a	

Outputs

- **y** is an "always-round-up" version of **x.**

Notes

Examples

- ceil(-2.3) returns -2.
- ceil(2.3) return 3.

See Also

fix | floor | round

conj

```
y = conj ( cx )
y = cx.conj ( )
```

Complex conjugate

arg	type	Description	Unit or Value scale
`cx`	SCAL or VCT or AUDIO	complex value(s)	

Outputs

- `y` is complex conjugate of `cx`.

Notes

Examples

- conj(3+2*sqrt(-1)) → 3-2*i
- (3+2*sqrt(-1)).conj → 3-2*i

See Also

imag | real

cos

```
y = cos ( x )
y = x.cos ( )
```

Cosine of argument in radians

arg	type	Description	Unit or Value scale
`x`	SCAL, VCT or AUDIO	n/a	

Outputs

- `y` is Cosine of `x`.

Notes
Examples
See Also

sin | tan | acos | asin | atan

exp

```
y = exp ( x )
y = x.exp ( )
```

Exponential

arg	type	Description	Unit or Value scale
x	SCAL, VCT or AUDIO	n/a	

Outputs

- **y** is the exponential e_x

Notes

Examples

- exp(1) → 2.71828 ← e

See Also

log

fix

```
y = fix ( x )
y = x.fix ( )
```

Round toward zero

arg	type	Description	Unit or Value scale
`x`	SCAL, VCT or AUDIO	n/a	

Outputs

- If you take out the decimal portion of `x`, you get `y`.

Notes

Examples

- fix(3.99) → 3
- fix(-3.99) → -3

See Also

ceil | floor | round

floor

```
y = floor ( x )
y = x.floor ( )
```

Round toward negative infinity

arg	type	Description	Unit or Value scale
`x`	SCAL, VCT or AUDIO	n/a	

Outputs

- `y` is a "round-down" version of `x`.

Notes

Examples

- floor(3.99) → 3
- floor(-3.99) → -4

See Also

ceil | fix | round

imag

```
y = imag ( Z )
y = Z.imag ( )
```

Imaginary part of complex number

arg	type	Description	Unit or Value scale
`Z`	SCAL or VCT or AUDIO	complex value(s)	

Outputs

- `y` is the imaginary part of `Z`

Notes

Examples

- (3).imag → 0
  (2+sqrt(-1)).imag → 1

See Also

conj | real | abs | angle

log

```
y = log ( x )
y = x.log ( )
```

Natural logarithm

arg	type	Description	Unit or Value scale
`x`	SCAL, VCT or AUDIO	n/a	

Outputs

- `y` is natural log of `x`

Notes

Examples

See Also

exp | log10

log10

```
y = log10 ( x )
y = x.log10 ( )
```

Common logarithm

arg	type	Description	Unit or Value scale
x	SCAL, VCT or AUDIO	n/a	

Outputs

- **y** is the common log of **x**

Notes

Examples

See Also

log

mod

```
y = mod ( x,div )
y = x.mod ( div )
y = x % div
```

Remainder after division (modulo operation)

arg	type	Description	Unit or Value scale
x	SCAL, VCT or AUDIO	n/a	
div	SCAL	n/a	

Outputs

- **y** is the remainder after division of **x** by **div**

Notes

Examples

- mod(17,5) rarr; 2
- mod(-1,3) rarr; -1
- (12).mod(5) rarr; 2

See Also

pow

```
y = pow ( x )
y = x.pow ( )
```

Raise to power

arg	type	Description	Unit or Value scale
x	SCAL, VCT or AUDIO	n/a	

Outputs

Notes

- base raised to the power exponent; $base^{power}$

Examples

See Also

log

real

```
y = real ( Z )
y = Z.real ( )
```

Real part of complex number

arg	type	Description	Unit or Value scale
`Z`	SCAL or VCT or AUDIO	complex value(s)	

Outputs

- `y` is the real part of `Z`

Notes

Examples

- (3).imag → 3
  (2+sqrt(-1)).imag → 2

See Also

imag | conj | abs | angle

round

```
y = round ( x )
y = x.round ( )
```

Round to nearest decimal or integer

arg	type	Description	Unit or Value scale
x	SCAL, VCT or AUDIO	n/a	

Outputs

- **y** is a "half-round-up" version of **x.**

Notes

Examples

- round(2.6) → 3
- round(2.4) → 2
- round(-2.6) → -3
- round(-2.4) → -2

See Also

ceil | fix | floor

sign

```
y = sign ( x )
y = x.sign ( )
```

Sign function (signum function)

arg	type	Description	Unit or Value scale
x	SCAL, VCT or AUDIO	n/a	

Outputs

- **y** is

Notes
Examples
See Also

sin

```
y = sin ( x )
y = x.sin ( )
```

Sine of argument in radians

arg	type	Description	Unit or Value scale
`x`	SCAL, VCT or AUDIO	n/a	

Outputs

- `y` is Sine of `x`.

Notes

Examples

See Also

cos | tan | acos | asin | atan

sqrt

```
y = sqrt ( x )
y = x.sqrt ( )
```

Square root

arg	type	Description	Unit or Value scale
x	SCAL or VCT or AUDIO	n/a	

Outputs

- **y** is square root of **x**.

Notes

- If **x** is non-audio, square root of negative values produce imaginary values. If **x** is audio, this produces sign(sqrt(**x**)).

Examples

See Also

tan

```
y = tan ( x )
y = x.tan ( )
```

Tangent of argument in radians

arg	type	Description	Unit or Value scale
x	SCAL, VCT or AUDIO	n/a	

Outputs

- **y** is tangent of **x.**

Notes
Examples
See Also

sin | cos | acos | asin | atan

Audio Playback Functions

In AUXLAB, you can play an audio signal, pause, resume or stop it prematurely. Multiple audio playback events can exist concurrently and can be controlled or managed. The handle to audio playback, the output of each play call, is used to specify the particular audio event to handling it. Note: play is non-blocking and will return immediately.

For example,

```
AUX> noi = noise(1000).bpf(500,2000);
```
To play just one and only one, no need to have the output
```
AUX> noi.play
```

But when playing a signal with a long duration, it is necessary to have the output,

```
AUX> noi = noise(10000).bpf(500,2000);x =
tone(500,10000)@-20;
AUX> h=noi.play
AUX> h2=x.play
```
Then,
```
AUX> h.pause or h2.pause
```
or
```
AUX> h.stop
```
in the middle of the playback will do the trick.

The audio handle, h or h2 in this example, contains the information about the audio play back including the progress of the playback and is constantly updated in the background. The user can view the status of the audio event in the variable view window or by retrieving it in the AUXLAB command window.

pause

```
y = pause ( h )
y = h.pause ( )
```

Pause an on-going audio playback

arg	type	Description	Unit or Value scale
`h`	AUDIOHANDLE	Audio playback event handle to pause	

Outputs

- `y` is the audio playback handle

Notes

Examples

See Also

resume | play | stop | qstop | status

play

```
y = play ( x,repeat[=1],devID[=""] )
y = play ( handle,x,repeat[=1],devID[=""] )
y = handle.play ( x,repeat[=1],devID[=""] )
```

Audio playback

arg	type	Description	Unit or Value scale
handle	HAUD	handle to audio playback	
x	AUD	audio signal	
repeat	SCAL	number of repeats	
devID	TXT	string identifier of the playback device (NOT YET IMPLEMENTED--as of AUXLAB 1.44)	

Outputs

- **y** is either an audio handle, either newly created or existing. **y** is -1 if the playback fails or the specified handle is invalid.

Notes

- If devID is not specified, the default device or the last device selected for playback will be used.
- If play is called for the audio handle, **handle** is queued in the back of the playback list and played when the existing list is exhausted.
  If play is called for an audio signal, a new audio handle is generated.

Examples

- To play x (and don't care about asynchronous playing),
  play(x)
- To play x and, then, y in sequence
  h=play(x)
  h.play(y) while x is played.
  If h.play(y) is given after playing x is done, it doesn't play and returns -1

See Also

pause | resume | status | stop | qstop

qstop

```
y = qstop ( h )
y = h.qstop ( )
```

Stop an on-going audio playback

arg	type	Description	Unit or Value scale
`h`	AUDIOHANDLE	Audio playback event handle to stop	

Outputs

- `y` is the audio playback handle

Notes

- Unlike stop, this function terminates the on-going audio event immediately.

Examples

See Also

play | stop | pause | resume | status

resume

```
y = resume ( h )
y = h.resume ( )
```

Resume a paused audio playback

arg	type	Description	Unit or Value scale
`h`	AUDIOHANDLE	Audio playback event handle to resume	

Outputs

- `y` is the audio playback handle

Notes

Examples

See Also

pause | play | stop | qstop | status

status

```
y = status ( h )
y = h.status ( )
```

Displays the status of the audio playback handle

arg	type	Description	Unit or Value scale
`h`	AUDIOHANDLE	Audio playback event handle	

Outputs

- `y` is an object variable showing the status of the audio handle.

Notes

Examples

See Also

stop

```
y = stop ( h )
y = h.stop ( )
```

Stop an on-going audio playback

arg	type	Description	Unit or Value scale
`h`	AUDIOHANDLE	Audio playback event handle to stop	

Outputs

- `y` is the audio playback handle

Notes

- The audio playback fades out (i.e., "ramps" out) with a cosine amplitude function for 350 milliseconds.

Examples

See Also

play | qstop | pause | resume | status

Graphics Functions

In addition to displaying the signal as a graph by pressing the enter key in the variable show window, graphic windows can be created and controlled with AUX functions. As of 9/20/2018, while there are five functions--plot, figure, axes, delete, text, each graphic object is equipped with numerous properties and the apperanace and display can be controlled by modifying the graphic properties.

Example

```
AUX> x=(1:10).sqrt;
AUX> hLine=plot(x);// plot returns the handle to the line object
AUX> hAx = hLine.parent;// hAx is the handle to the axes object
AUX> hFig = hAx.parent;// hFig is the handle to the figure object
AUX> hAx.pos(4) /=2;// The axes size adjusted--the height becomes half
AUX> hAx2 = hFig.axes(hAx.pos+[0 .45 0 0]);// A new axis is added to the figure window
AUX> hAx2.plot((1:10).log); // The second plot
AUX> hTxt = hFig.text(.5,.55,"Two plots") // A text added
AUX> hTxt.fontsize(15) // Fonts ize adjusted
```

axes

```
y = axes ( h_or_pos )
y = h_or_pos.axes ( )
```

Set the current axes or create a new axes

arg	type	Description	Unit or Value scale
`h_or_pos`	HGO or VECT	handle to the axes or 4-element vector with the position to create a new axes in	

Outputs

- `y` is the handle to the axes object

Notes

- The position is a 4-element vector in proportion to the window size showing the following: top left width height, with the reference of the bottom left corner of the screen.
- This function creates an axes in the current figure window. If no current figure window is available, a new one is created.

Examples

- h = axes([.08 .18 .86 .5]) // create a new axes in the current figure window
- axes(hPrev) // set hPrev as the current axes

See Also

figure | plot | text | delete

delete

```
y = delete ( object )
y = object.delete ( )
```

Delete the graphic object

arg	type	Description	Unit or Value scale
`object`	HGO	The graphic handle to the object to delete	

Outputs

- `y` is empty.

Notes

- Upon success, the variable `object` will become empty.

Examples

See Also

figure | plot | axes | text

figure

```
y = figure ( h_or_pos )
```

Get the handle of an existing figure window or create a blank figure window

arg	type	Description	Unit or Value scale
`h_or_pos`	HGO or VECT	handle to the figure window or 4-element vector with the position to create a new figure at	

Outputs

- `y` is the handle to the figure window

Notes

- The position is a 4-element vector in pixel count showing the following: top left width height, with the reference of the top left corner of the screen.

Examples

See Also

plot | axes | text | delete

plot

```
y = plot ( x,y[=""],opt[=""] )
y = plot ( handle,x,y[=""],opt[=""] )
y = handle.plot ( x,y[=""],opt[=""] )
```

Plot the data

arg	type	Description	Unit or Value scale
handle	HGO	handle to the figure window or the axes	
x	VECT or AUD	x-data	
y	VECT or AUD	y-data	
opt	TXT	plot option string--specifying color marker and line style (see notes)	

Outputs

- **y** is the handle to the line object

Notes

- If **handle** is omitted, this function creates a new figure window and plots
- If **y** is specified, **y** is plotted as a function of **x**.
- If **y** is omitted, **x** is plotted with the index (if non-audio) or the time (if audio) on the x-axis.
- **opt** is a text not exceeding 4 characters specifying the color, the marker type and the line style.
  Color: r(ed) g(reen) b(lue) y(ellow) c(yan) m(agenta) h(white) (blac)k
  Marker: o circle s square . point * asterisk x cross + plussign d diamond ^ upward-pointingtriangle v downward-pointingtriangle > right-pointingtriangle < left-pointingtriangle
  Linestyle: - solid : dotted -- dashed -. dash-dot
- The default marker type is . (point)
- The default line style is solid, but if only the marker type is specified, the line style will become none (i.e., in order to make the line tyle none, specify ".")
- The default color is blue
- All the other graphic properties can be directly manipulated with the relevant member variables.

Examples

- plot(x) // plot x with the default parameters

- plot(x,y) // plot y as a function of x with the default parameters
- plot(x,"r") // plot x with the red line
- plot(x,"o") // plot x with the marker "o" and no line
- plot(x,"*":) // plot x with the marker "*" and dotted line
- plot(x,y,"o") // plot y as a function of x with the marker "o" and no line
- plot(x,y,"*:") // plot y as a function of x with the marker "*" and dotted line
- plot(h,x) // plot x in h (either a figure window handle or axes handle)
- plot(h,x,y)
- plot(h,x,y,"r")

See Also

figure | axes | text | delete

text

```
y = text ( h,pos_x,pos_y,text )
y = _h.text ( pos_x,pos_y,text )
```

Display a text in the figure window or axes.

arg	type	Description	Unit or Value scale
`_h`	HGO	Graphic handle, either figure or axes handle	
`pos_x`	SCAL	x position proportion of the window or axes	
`pos_y`	SCAL	y position proportion of the window or axes	
`text`	TXT	Text to display	

Outputs

- `y` is the handle to the text displayed.

Notes

- If `_h` is omitted, the text is displayed in the current figure window. If the current figure window doesn't exist, it creates one.

Examples

- text(.5, .5, "hello world!") // hello world! is displayed at the center point in the current window (left-aligned)

See Also

figure | plot | axes | delete

Logical Functions

and

```
y = and ( b1,b2 )
y = b1.and ( b2 )
```

Point-wise boolean operation && for two arguments; Check if all logical elements are true.

arg	type	Description	Unit or Value scale
`b1`	LOGICAL VCT or SCL	Logical array or logical scalar	
`b2`	LOGICAL VCT or SCL	Logical array or logical scalar	

Outputs

- `y` is a logical array for two arguments with the length of a shorter argument or a logical scalar for one argument.

Notes

- If two arguments have different length, the boolean operation is applied only to the common length.

Examples

See Also

or

isaudio

```
y = isaudio ( x )
y = x.isaudio ( )
```

Checks if the input array is an audio array.

arg	type	Description	Unit or Value scale
`x`	anything		

Outputs

- `y` is true if `x` is audio.

Notes

Examples

See Also

isvector

isbool

```
y = isbool ( x )
y = x.isbool ( )
```

Checks if the input is logical.

arg	type	Description	Unit or Value scale
`x`	anything		

Outputs

- `y` is true if `x` is logical.

Notes

Examples

See Also

iscell

```
y = iscell ( x )
y = x.iscell ( )
```

Checks if the input is a cell array.

arg	type	Description	Unit or Value scale
`x`	anything		

Outputs

- `y` is true if `x` is a cell array.

Notes

Examples

See Also

isclass

```
y = isclass ( x )
y = x.isclass ( )
```

Checks if the variable is a class object.

arg	type	Description	Unit or Value scale
`x`	anything		

Outputs

- `y` is boolean

Notes

Examples

- isaudio isvector isstring isbool isempty

See Also

isempty

```
y = isempty ( x )
y = x.isempty ( )
```

Checks whether the input array is empty.

arg	type	Description	Unit or Value scale
`x`	anything		

Outputs

- `y` is true if the input is empty.

Notes

- For a time-seq variable, the array can have zero length but shouldn't be considered empty because it carries the tmark information.

Examples

See Also

isstereo

```
y = isstereo ( x )
y = x.isstereo ( )
```

Checks if the audio signal is stereo.

arg	type	Description	Unit or Value scale
`x`	AUD	n/a	

Outputs

- `y` is logical indicating whether the signal is stereo.

Notes

Examples

See Also

isstring

```
y = isstring ( x )
y = x.isstring ( )
```

Checks if the input is string.

arg	type	Description	Unit or Value scale
`x`	anything		

Outputs

- `y` is true if `x` is string.

Notes

Examples

- x="bj kwon";
- x.isstring → true

See Also

isvector

```
y = isvector ( x )
y = x.isvector ( )
```

Checks if the input array is a vector (non-audio array).

arg	type	Description	Unit or Value scale
`x`	anything		

Outputs

- `y` is true if `x` is a vector (non-audio array).

Notes

Examples

See Also

isaudio

or

```
y = or ( b1,b2 )
y = b1.or ( b2 )
```

Point-wise boolean operation || for two arguments; Check if any one element is true.

arg	type	Description	Unit or Value scale
`b1`	LOGICAL VCT or SCL	Logical array or logical scalar	
`b2`	LOGICAL VCT or SCL	Logical array or logical scalar	

Outputs

- `y` is a logical array for two arguments with the length of a shorter argument or a logical scalar for one argument.

Notes

- If two arguments have different length, the boolean operation is applied only to the common length.

Examples

See Also

and

File/Directory Functions

dir

```
y = dir ( s )
```

Retrieve the files in the specified directory

arg	type	Description	Unit or Value scale
s	STR	File name with or without a wild card, or directory	

Outputs

- y is a cell array showing the directory contents. Each cell is a directory class with the following members: name, ext, path, isdir, date, and bytes.

Notes

Examples

- y = dir ("c:\Temp\auxlab\1.42\") or
- y = dir ("c:\Temp\auxlab\1.42*.ini")
- y{1} =
  .bytes = 1341
  .date = "06/18/2018, 04:18:28"
  .ext = ".ini"
  .isdir = (logical) 0
  .name = "auxcon32.AUDITORY"
  .path = "C:\Temp\auxlab\1.42\"
- y{2} =
  .bytes = 160
  .date = "06/18/2018, 04:22:06"
  .ext = ".ini"
  .isdir = (logical) 0
  .name = "auxlab32.AUDITORY"
  .path = "C:\Temp\auxlab\1.42\"

See Also

fclose

```
y = fclose ( file_id )
```

Closes a file stream specified by the file identifier

arg	type	Description	Unit or Value scale
`file_id`	SCAL	File identifier from the fopen call	

Outputs

- `y` is SCAL. 0 for success, -1 for failure.

Notes

Examples

See Also

fopen | fprintf

fdelete

```
y = fdelete ( filename )
```

Delete a file

arg	type	Description	Unit or Value scale
`filename`	STR	The file name to delete	

Outputs

- `y` is 1 for success, 0 for failure.

Notes

- The file name may include the path.

Examples

- fdelete("c:\delete.me")

See Also

fopen | fprintf | fclose

file

```
y = file ( filename )
```

Read from a file (TO BE DONE)

arg	type	Description	Unit or Value scale
`filename`	TEXT	.....	

Outputs

- `y` is anything

Notes

Examples

See Also

fopen

```
y = fopen ( filename,mode )
```

Opens a file with the given mode

arg	type	Description	Unit or Value scale
`filename`	TXT	Name of the file to open	
`mode`	TXT	File open mode, such as "r" "w" "a"	

Outputs

- `y` is SCAL indicating the file identifier. `y` is -1 if there's an error.

Notes

- Internally, the C fopen function is called with the given arguments. This means that this follows all the conventions in C language.

Examples

See Also

fclose | fprintf | fread | fwrite

fprintf

```
y = fprintf ( file,format,... )
```

Write formatted data to a file

arg	type	Description	Unit or Value scale
`file`	TXT or SCAL	file name string or file identifier	
`format`	TXT	String that contains the text to be writeen to the file, following the same convention as in C language.	
`...`	...	n/a	

Outputs

- `y` is the number of characters written. If an error occurs, `y` is negative and one of the following:
  -1: fopen error
  -2: invalid file identifier
  -3: fwrite error
  -4: fclose error
  -999: Unknown error

Notes

- `file` can be either a text of the file name (if an extension is not specified, .txt is added by default) or a file identifier, which is the output of fopen
- When the file name is specified for `file`, fprintf opens the file with the mode "at," meaning that the content will be appended to the existing content in the file, writes the content as specified, and closes the file.
- If a file identifier is used for `file`, fprintf writes the content to the file stream opened by a prior call to fopen.

Examples

See Also

fopen | fclose | sprintf

include

```
include ( filename )
```

Run a script

arg	type	Description	Unit or Value scale
`filename`	TXT	file name	

Outputs

- No output

Notes

- This is simply to run a batch AUX script, which is different running a UDF (user-defined function).

Examples

See Also

wavwrite

```
y = wavwrite ( x,filename,opt )
y = x.wavwrite ( filename,opt )
```

Generate a .wav file from the audio signal

arg	type	Description	Unit or Value scale
`x`	AUDIO		
`filename`	STR	.wav audio file name	
`opt`	STR	file encoding format	

Outputs

- The output `y`, if specified, is the audio signal `x`.

Notes

- If `filename` doesn't indicate the extension, .wav is used.
- `opt` is one of the following
  - "8" 8-bit PCM
  - "16" 16-bit PCM (default)
  - "24" 24-bit PCM
  - "32" 32-bit PCM
  - "alaw" a-law encoding
  - "ulaw" μ-law encoding
  - "adpcm1" ADPCM encoding 1
  - "adpcm2" ADPCM encoding 2

Examples

See Also

wave

Miscellaneous Functions

clear

```
clear ( x )
x.clear ( )
```

Clear the variable (or the member variable) from the workspace

arg	type	Description	Unit or Value scale
x		any variable or a member variable of a class variable	

Outputs

- No output

Notes

Examples

- a.clear or clear(a)→ clear the variable a
  a.obj.clear or clear(a.obj) → clear the member obj from the class variable a

See Also

eval

```
n/a = eval ( expression )
```

Execute AUXLAB expression in text string

arg	type	Description	Unit or Value scale
expression	TXT	String that contains a valid AUXLAB expression.	

Outputs

- n/a

Notes

- Similar to the eval function in MATLAB

Examples

See Also

getfs

```
y = getfs ( )
```

Retrieve the sample rate in the AUXLAB workspace.

arg	type	Description	Unit or Value scale

Outputs

- `y` is the sample rate in the AUXLAB workspace.

Notes

- no argument

Examples

See Also

setfs

inputdlg

```
y = inputdlg ( title,content )
y = title.inputdlg ( content )
```

Create a simple message dialog box with OK and cancel buttons and an edit box for user input.

arg	type	Description	Unit or Value scale
`title`	TEXT	The title of the dialog box	
`content`	TEXT	The content to display in the dialog box. May use the printf format	

Outputs

- `y` is the string typed by the user.

Notes

- This does not return until the user responds with OK or cancel.

Examples

See Also

input

msgbox

msgbox (format,arg)

Display a messagebox

arg	type	Description	Unit or Value scale
format	TEXT	printf-style format	
arg	anything ...	variables	

Outputs

- No output

Notes

Examples

- for k=1:100, value = 2^k; if value > 1000 msgbox("2 raised by the power of %d is greater than 1000.", k); break; end; end

See Also

input | inputdlg

setfs

```
y = setfs ( new_Fs )
```

Adjusts the sample rate to the specified value

arg	type	Description	Unit or Value scale
`new_Fs`	SCL	The new sample rate	

Outputs

- `y` is undefined. Don't try to use it, as in setfs(16000)+1

Notes

- The usage of this function is limited to the user-defined functions or the auxcon module. In general, it is better to use a hook command (as in #setfs 16000), because it involves UI and expressions. This will not update existing variables according to the new sample rate, as done by the hook command, so this functionality is pretty limited.

Examples

See Also