AUXLAB

**Table of contents**

[Introduction](#_topic_Introduction) 5

[Credits](#_topic_Credits) 8

[System Requirements](#_topic_SystemRequirements) 11

[Data Types](#_topic_DataTypes) 14

[Time Sequence](#_topic_TimeSequence) 17

[Operators](#_topic_Operators) 21

[+ -](#_topic_Newtopic) 24

[\* /](#_topic_Newtopic1) 28

[:](#_topic_Newtopic2) 31

[: (indexing)](#_topic_indexing) 34

[~ (indexing)](#_topic_indexing1) 37

['](#_topic_Newtopic3) 40

[%](#_topic_Newtopic4) 43

[^](#_topic_Newtopic5) 46

[@](#_topic_Newtopic6) 49

[>>](#_topic_Newtopic7) 52

[stereo](#_topic_stereo) 55

[~](#_topic_Newtopic8) 58

[->](#_topic_Newtopic9) 61

[<>](#_topic_Newtopic10) 64

[#](#_topic_Newtopic11) 67

[Signal/Vector Generation](#_topic_SignalVectorGeneration) 70

[cell](#_topic_cell) 73

[dc](#_topic_dc) 76

[fm](#_topic_fm) 79

[gnoise](#_topic_gnoise) 82

[input](#_topic_input) 85

[irand](#_topic_irand) 88

[noise](#_topic_noise) 91

[ones](#_topic_ones) 94

[rand](#_topic_rand) 97

[randperm](#_topic_randperm) 100

[silence](#_topic_silence) 103

[sprintf](#_topic_sprintf) 106

[str2num](#_topic_str2num) 109

[tone](#_topic_tone) 112

[wave](#_topic_wave) 115

[zeros](#_topic_zeros) 118

[Modification of Signal/Vector](#_topic_ModificationofSignalVector) 121

[audio](#_topic_audio) 124

[blackman](#_topic_blackman) 127

[bpf](#_topic_bpf) 130

[bsf](#_topic_bsf) 133

[filt](#_topic_filt) 136

[filtfilt](#_topic_filtfilt) 139

[fscale](#_topic_fscale) 142

[hamming](#_topic_hamming) 145

[hann](#_topic_hann) 148

[hpf](#_topic_hpf) 151

[interp](#_topic_interp) 154

[lpf](#_topic_lpf) 157

[matrix](#_topic_matrix) 160

[movespec](#_topic_movespec) 163

[ramp](#_topic_ramp) 166

[sam](#_topic_sam) 169

[sort](#_topic_sort) 172

[tscale](#_topic_tscale) 175

[vector](#_topic_vector) 178

[Computation Functions](#_topic_ComputationFunctions) 181

[begint](#_topic_begint) 184

[cumsum](#_topic_cumsum) 187

[diff](#_topic_diff) 190

[dur](#_topic_dur) 193

[endt](#_topic_endt) 196

[envelope](#_topic_envelope) 199

[envelope](#_topic_envelope1) 202

[fft](#_topic_fft) 205

[hilbert](#_topic_hilbert) 208

[ifft](#_topic_ifft) 211

[left](#_topic_left) 214

[length](#_topic_length) 217

[max](#_topic_max) 220

[min](#_topic_min) 223

[right](#_topic_right) 226

[rms](#_topic_rms) 229

[rmsall](#_topic_rmsall) 232

[size](#_topic_size) 235

[sort](#_topic_sort1) 238

[std](#_topic_std) 241

[sum](#_topic_sum) 244

[Math Functions](#_topic_MathFunctions) 247

[abs](#_topic_abs) 250

[acos](#_topic_acos) 253

[angle](#_topic_angle) 256

[asin](#_topic_asin) 259

[atan](#_topic_atan) 262

[ceil](#_topic_ceil) 265

[conj](#_topic_conj) 268

[cos](#_topic_cos) 271

[exp](#_topic_exp) 274

[fix](#_topic_fix) 277

[floor](#_topic_floor) 280

[imag](#_topic_imag) 283

[log](#_topic_log) 286

[log10](#_topic_log10) 289

[mod](#_topic_mod) 292

[real](#_topic_real) 295

[round](#_topic_round) 298

[sign](#_topic_sign) 301

[sin](#_topic_sin) 304

[sqrt](#_topic_sqrt) 307

[tan](#_topic_tan) 310

[Audio playback Functions](#_topic_AudioplaybackFunctions) 313

[pause](#_topic_pause) 316

[play](#_topic_play) 319

[qstop](#_topic_qstop) 322

[resume](#_topic_resume) 325

[status](#_topic_status) 328

[stop](#_topic_stop) 331

[Graphic Functions](#_topic_GraphicFunctions) 334

[axes](#_topic_axes) 337

[delete](#_topic_delete) 340

[figure](#_topic_figure) 343

[plot](#_topic_plot) 346

[text](#_topic_text) 349

[Logical Functions](#_topic_LogicalFunctions) 352

[and](#_topic_and) 355

[isaudio](#_topic_isaudio) 358

[isbool](#_topic_isbool) 361

[iscell](#_topic_iscell) 364

[isclass](#_topic_isclass) 367

[isempty](#_topic_isempty) 370

[isstereo](#_topic_isstereo) 373

[isstring](#_topic_isstring) 376

[isvector](#_topic_isvector) 379

[or](#_topic_or) 382

[File Handling Functions](#_topic_FileHandlingFunctions) 385

[dir](#_topic_dir) 388

[fclose](#_topic_fclose) 391

[fdelete](#_topic_fdelete) 394

[file](#_topic_file) 397

[fopen](#_topic_fopen) 400

[fprintf](#_topic_fprintf) 403

[include](#_topic_include) 406

[wavwrite](#_topic_wavwrite) 409

[Miscellaneous Functions](#_topic_MiscellaneousFunctions) 412

[clear](#_topic_clear) 415

[eval](#_topic_eval) 418

[getfs](#_topic_getfs) 421

[input](#_topic_input1) 424

[inputdlg](#_topic_inputdlg) 427

[msgbox](#_topic_msgbox) 430

[setfs](#_topic_setfs) 433

[sprintf](#_topic_sprintf1) 436

[str2num](#_topic_str2num1) 439

**Introduction**

**Introduction to AUXLAB**

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.

The most outstanding weakness of AUXLAB so far is the lack of documentation. But this help file will be a quick guide to those who are capable.

However crass it is, this document covers most of functionalities available in the current version of AUXLAB v1.48, except

* User defined functions
* Debugger--so you can use it to develop audio processing algorithms.
* 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---They are smilar to MATLAB, but still there are numerous differences...*OK, some graphic functions are still somewhat shaky, probably I need to work on them first.*

The documentation for the above features will be added soon.

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**Credits**

**License & Credit**

AUXLAB is released under Academic Free License 3.0.

This program is free software; you can redistribute it and/or modify it under the terms of the Academic Free License (AFL) v.3.0 as published by the Open Source Initiative (OSI). This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. To view the complete license of AFL v.3.0: https://opensource.org/licenses/AFL-3.0

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 git 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

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**System Requirements**

**System Requirements**

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

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**Data Types**

**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](#_topic_TimeSequence) |
| 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))

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**Time Sequence**

**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**

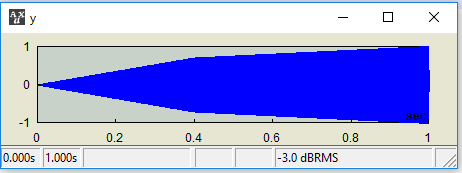
Sometimes it is very useful to have time values relative to another audio signal. In such cases, define a relative time sequence as follows:

|  |
| --- |
| [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.

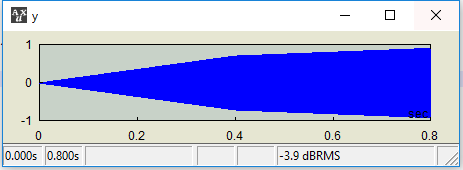
|  |
| --- |
| AUX> x = tone(500,1000); |
| AUX> ts = [0 250 500 1000][0 .7 .3 1]; |
| AUX> y = ts \* x; |



**Example 2**

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

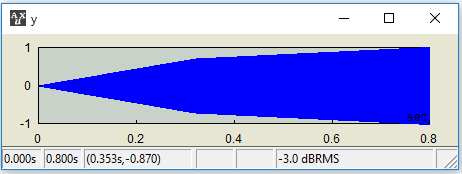
|  |
| --- |
| AUX> x = tone(500,800); |
| AUX> ts = [0 250 500 1000][0 .7 .3 1]; |
| AUX> y = ts \* x; |



**Example 3**

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

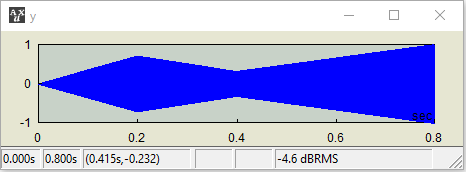
|  |
| --- |
| AUX> x = tone(500,800); |
| AUX> ts = [0 .25 .5 1;][0 .7 .3 1]; |
| AUX> y = ts \* x; |



**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; |



|  |
| --- |
|  |
|  |
|  |
|  |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**Operators**

**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).

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**+ -**

**+ -**

|  |
| --- |
| **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)  example1  AUX> tone(100,100)@-6 + tone(200,100)@-6 >>50 z = audio (0.0ms~150.0ms)  example2 | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**\* /**

**\* /**

|  |
| --- |
| **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. | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**:**

**:**

|  |
| --- |
| **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 | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**: (indexing)**

**: (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 | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**~ (indexing)**

**~ (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); | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**'**

**'**

|  |
| --- |
| **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 | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**%**

**%**

|  |
| --- |
| **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](#_topic_hamming)for examples. | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**^**

**^**

|  |
| --- |
| **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. | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**@**

**@**

|  |
| --- |
| **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] | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**>>**

**>>**

|  |
| --- |
| **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 | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**stereo**

**[ ; ]**

|  |
| --- |
| **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) | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**~**

**~**

|  |
| --- |
| **z = x ~ y** |

Time-Compress, or Spectrum-Expand, 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** | | |
|  | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**->**

**->**

|  |
| --- |
| **z = x -> y** |

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 | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**<>**

**<>**

|  |
| --- |
| **z = x <> y** |

Change the duration of an audio signal x by a factor of y without changing the spectrum

|  |  |  |
| --- | --- | --- |
| **Commutative** | | |
| No | | |
| **Data Types** | | |
| **x** | **y** | **z** |
| AUD | SCAL | AUD |
| **Notes** | | |
| * This is equivalent to z=tscale(x,y) or z=x.tscale(y) * The exact rendition of this operation may depend on the algorithm used. | | |
| **Examples** | | |
| * A tone glide from 400 Hz to 1000 Hz with arbitrary amplitude fluctuations.   AUX> x=tone([400 1000],1000).sqrt@-3 AUX> tseq=[0 .2 .3 .7 .8 1;][-100 -6 -30 -10 -5 0]; AUX> y=(x@tseq).ramp(20)    AUX> z=y<>.8 | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**#**

**#**

|  |
| --- |
| **z = x # y** |

Change the pitch of an audio signal x by y semitones without changing the duration

|  |  |  |
| --- | --- | --- |
| **Commutative** | | |
| No | | |
| **Data Types** | | |
| **x** | **y** | **z** |
| AUD | SCAL | AUD |
| **Notes** | | |
| * This is equivalent to z=fscale(x,y) or z=x.fscale(y) * The exact rendition of this operation may depend on the algorithm used. | | |
| **Examples** | | |
| AUX> x=tone(500,50).sqrt@-6    AUX> y=x#7 //7 semitones:frequencies increased by a factor of 1.5 | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**Signal/Vector Generation**

**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](#_topic_tone) (to create a pure tone), [noise](#_topic_gnoise) (to create white noise), [wave](#_topic_wave) (to read .wav file), and [rand](#_topic_irand) (to create an array of random numbers).

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**cell**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**dc**

**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](#_topic_ones) | [silence](#_topic_silence) | [zeros](#_topic_zeros)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**fm**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**gnoise**

**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](#_topic_gnoise)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**input**

**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](#_topic_inputdlg)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**irand**

**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](#_topic_irand) | [randperm](#_topic_randperm)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**noise**

**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](#_topic_gnoise)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**ones**

**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](#_topic_zeros)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**rand**

**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](#_topic_irand) | [randperm](#_topic_randperm)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**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](#_topic_irand) | [rand](#_topic_irand)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**silence**

**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](#_topic_dc) | [zeros](#_topic_zeros)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**sprintf**

**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](#_topic_fprintf)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**str2num**

**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](#_topic_eval) | [sprintf](#_topic_sprintf)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**tone**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**wave**

**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](#_topic_wavwrite)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**zeros**

**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](#_topic_ones)

*Created with the Personal Edition of :*

**Modification of Signal/Vector**

**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](#_topic_filt), [filtfilt](#_topic_filtfilt), [lpf](#_topic_lpf), [hpf](#_topic_hpf), [bpf](#_topic_bpf), [bsf](#_topic_bsf)), windowing ([ramp](#_topic_ramp), [blackman](#_topic_blackman), [hann](#_topic_hann), [hamming](#_topic_hamming)), or altering features of audio signals, such as time-scaling [tscale](#_topic_Newtopic10), frequency-scaling [fscale](#_topic_Newtopic11) or spectrum-shifting [movespec](#_topic_Newtopic9).

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**audio**

**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](#_topic_vector)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**blackman**

**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](#_topic_blackman) | [hamming](#_topic_hamming)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**bpf**

**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](#_topic_hpf) | [bpf](#_topic_bpf) | [bsf](#_topic_bsf)

|  |
| --- |
| **Algorithm** |

[ELLF Digital Filter Calculator](http://www.moshier.net/ellfdoc.html)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**bsf**

**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](#_topic_lpf) | [hpf](#_topic_hpf) | [bpf](#_topic_bpf)

|  |
| --- |
| **Algorithm** |

[ELLF Digital Filter Calculator](http://www.moshier.net/ellfdoc.html)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**filt**

**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](#_topic_filtfilt)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**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](#_topic_filt)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**fscale**

**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](#_topic_Newtopic10) | [movespec](#_topic_Newtopic9)

|  |
| --- |
| **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.

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**hamming**

**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](#_topic_hann) | [blackman](#_topic_blackman)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**hann**

**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](#_topic_hamming) | [blackman](#_topic_blackman)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**hpf**

**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](#_topic_lpf) | [bpf](#_topic_bpf) | [bsf](#_topic_bsf)

|  |
| --- |
| **Algorithm** |

[ELLF Digital Filter Calculator](http://www.moshier.net/ellfdoc.html)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**interp**

**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)](http://www.mega-nerd.com/SRC/) is used.

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**lpf**

**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](#_topic_hpf) | [bpf](#_topic_bpf) | [bsf](#_topic_bsf)

|  |
| --- |
| **Algorithm** |

[ELLF Digital Filter Calculator](http://www.moshier.net/ellfdoc.html)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**matrix**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**movespec**

**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](#_topic_Newtopic11)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**ramp**

**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](#_topic_hann) | [hamming](#_topic_hamming) | [blackman](#_topic_blackman)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**sam**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**sort**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**tscale**

**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](#_topic_Newtopic11) | [movespec](#_topic_Newtopic9)

|  |
| --- |
| **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.

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**vector**

**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](#_topic_audio)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**Computation Functions**

**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](#_topic_rms)), duration ([dur](#_topic_dur)), the timing information ([begint](#_topic_begint), [endt](#_topic_endt)), statistics ([mean](mean.html), [std](#_topic_std)), [length](#_topic_length), [size](#_topic_size)).

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**begint**

**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](#_topic_dur) | [endt](#_topic_endt)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**cumsum**

**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](#_topic_diff) | [reshape](reshape.html)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**diff**

**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](#_topic_cumsum)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**dur**

**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](#_topic_begint) | [endt](#_topic_endt)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**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](#_topic_begint) | [dur](#_topic_dur)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**envelope**

**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](#_topic_hilbert)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**envelope**

**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](#_topic_hilbert)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**fft**

**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](#_topic_ifft)

|  |
| --- |
| **Algorithm** |

[FFTW](http://www.fftw.org/)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**hilbert**

**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](#_topic_envelope)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**ifft**

**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](#_topic_fft)

|  |
| --- |
| **Algorithm** |

FFTW

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**left**

**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** | | | |

[right](#_topic_right)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**length**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**max**

**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](#_topic_min)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**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](#_topic_max)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**right**

**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** | | | |

[left](#_topic_left)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**rms**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**rmsall**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**size**

**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](#_topic_length)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**sort**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**std**

**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](#_topic_cumsum)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**Math Functions**

**Mathematical Functions**

These are the collections of mathematical functions.

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**abs**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**acos**

**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](#_topic_asin) | [cos](#_topic_acos) | [tan](#_topic_atan) | [asin](#_topic_asin) | [atan](#_topic_atan)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**angle**

**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](#_topic_abs) | [atan](#_topic_atan)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**asin**

**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](#_topic_asin) | [cos](#_topic_acos) | [tan](#_topic_atan) | [acos](#_topic_acos) | [atan](#_topic_atan)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**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](#_topic_asin) | [cos](#_topic_acos) | [tan](#_topic_atan) | [asin](#_topic_asin) | [acos](#_topic_acos)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**ceil**

**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](#_topic_fix) | [floor](#_topic_floor) | [round](#_topic_round)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**conj**

**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](#_topic_imag) | [real](#_topic_real)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**cos**

**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](#_topic_asin) | [tan](#_topic_atan) | [acos](#_topic_acos) | [asin](#_topic_asin) | [atan](#_topic_atan)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**exp**

**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](#_topic_log)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**fix**

**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](#_topic_ceil) | [floor](#_topic_floor) | [round](#_topic_round)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**floor**

**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](#_topic_ceil) | [fix](#_topic_fix) | [round](#_topic_round)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**imag**

**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](#_topic_conj) | [real](#_topic_real) | [abs](#_topic_abs) | [angle](#_topic_angle)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**log**

**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](#_topic_exp) | [log10](#_topic_log10)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**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](#_topic_log)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**mod**

**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) → 2 * mod(-1,3) → -1 * (12).mod(5) → 2 | | | |
| **See Also** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**real**

**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](#_topic_imag) | [conj](#_topic_conj) | [abs](#_topic_abs) | [angle](#_topic_angle)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**round**

**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](#_topic_ceil) | [fix](#_topic_fix) | [floor](#_topic_floor)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**sign**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**sin**

**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](#_topic_acos) | [tan](#_topic_atan) | [acos](#_topic_acos) | [asin](#_topic_asin) | [atan](#_topic_atan)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**sqrt**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**tan**

**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](#_topic_asin) | [cos](#_topic_acos) | [acos](#_topic_acos) | [asin](#_topic_asin) | [atan](#_topic_atan)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**Audio playback Functions**

**Audio Playback Functions**

In AUXLAB, you can [play](#_topic_play) an audio signal, [pause](#_topic_pause), [resume](#_topic_resume) or [stop](#_topic_qstop) 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](#_topic_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.

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**pause**

**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](#_topic_resume) | [play](#_topic_play) | [stop](#_topic_qstop) | [qstop](#_topic_qstop) | [status](#_topic_status)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**play**

**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](#_topic_pause) | [resume](#_topic_resume) | [status](#_topic_status) | [stop](#_topic_qstop) | [qstop](#_topic_qstop)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**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](#_topic_play) | [stop](#_topic_qstop) | [pause](#_topic_pause) | [resume](#_topic_resume) | [status](#_topic_status)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**resume**

**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](#_topic_pause) | [play](#_topic_play) | [stop](#_topic_qstop) | [qstop](#_topic_qstop) | [status](#_topic_status)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**stop**

**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](#_topic_play) | [qstop](#_topic_qstop) | [pause](#_topic_pause) | [resume](#_topic_resume) | [status](#_topic_status)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**Graphic Functions**

**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](#_topic_plot), [figure](#_topic_figure), [axes](#_topic_axes), [delete](#_topic_delete), [text](#_topic_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

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**axes**

**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](#_topic_figure) | [plot](#_topic_plot) | [text](#_topic_text) | [delete](#_topic_delete)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**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](#_topic_figure) | [plot](#_topic_plot) | [axes](#_topic_axes) | [text](#_topic_text)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**figure**

**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](#_topic_plot) | [axes](#_topic_axes) | [text](#_topic_text) | [delete](#_topic_delete)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**plot**

**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](#_topic_figure) | [axes](#_topic_axes) | [text](#_topic_text) | [delete](#_topic_delete)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**text**

**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](#_topic_figure) | [plot](#_topic_plot) | [axes](#_topic_axes) | [delete](#_topic_delete)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**Logical Functions**

**Logical Functions**

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**and**

**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](#_topic_vector)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**isaudio**

**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](#_topic_isvector)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**isbool**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**iscell**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**isclass**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**isempty**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**isstereo**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**isstring**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**isvector**

**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](#_topic_isaudio)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**or**

**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](#_topic_irand)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**File Handling Functions**

**File/Directory Functions**

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**dir**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**fclose**

**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](#_topic_fopen) | [fprintf](#_topic_fprintf)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**fdelete**

**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](#_topic_fopen) | [fprintf](#_topic_fprintf) | [fclose](#_topic_fclose)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**file**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**fopen**

**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](#_topic_fclose) | [fprintf](#_topic_fprintf) | [fread](fread.html) | [fwrite](fwrite.html)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**fprintf**

**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 written 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](#_topic_fopen) | [fclose](#_topic_fclose) | [sprintf](#_topic_sprintf)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**include**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**wavwrite**

**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](#_topic_wave)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**Miscellaneous Functions**

**Miscellaneous Functions**

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**clear**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**eval**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**getfs**

**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](#_topic_setfs)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**input**

**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](#_topic_inputdlg)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**inputdlg**

**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](#_topic_input)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**msgbox**

**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](#_topic_input) | [inputdlg](#_topic_inputdlg)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**setfs**

**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** | | | |

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**sprintf**

**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](#_topic_fprintf)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*

**str2num**

**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](#_topic_eval) | [sprintf](#_topic_sprintf)

*Audio Processing Made Easy --- auxlab.org. Last Update 9/25/2018*