IMPULSE AUDIO LAB

Expression System for Audio Weaver™

Operator Reference

05.04.2023

Version 1.2.1 (v1.2.1-95-gda85f90)

Impulse Audio Lab GmbH

A simple C-like Expression language for realtime dynamic CAN processing in a running Audio Weaver™ system.

1 Operator Reference

1.1 Complete Operators Listing

Group	Operator Name	In Language	Alias	Notes
Arithmetic	add	+		
	subtract	-		
	multiply	*		
	divide	/		
	modulo	%	mod()	
Comparison	less than	<		
	greater than	>		
	equal to	==		
	not equal to	!=		
	less than or equal to	<=		
	greater than or equal to	>=		
Logical	and	&&		
	or	||		
	not	!		
Numeric	minimum	minimum()	min()	
	maximum	maximum()	max()	
	Sign of input	sign()	SIGN()	Output -1 or 1
	Absolute value	abs()	ABS()	
	lower integer	floor()		

	upper integer	ceiling()	ceil()	
	nearest integer	round()		
	integer part	integer()		Integer part with sign of (float) input
	fractional part	fraction()	fract()	Fractional part with sign of (float) input
Functions TRIG	sine	sin()	SIN()	
	cosine	cos()	COS()	
	tangent	tan()	TAN()	
	arc sine	asin()	ASIN()	
	arc cosine	acos()	ACOS()	
	arc tangent	atan()	ATAN()	
	arc tangent 2	atan2()	ATAN2()	
	hyperbolic sine	sinh()	SINH()	
	hyperbolic cosine	cosh()	COSH()	
	hyperbolic tangent	tanh()	TANH()	
Functions EXP	logarithm base 10	log10()	LOG10()	
	logarithm base 2	log2()	LOG2()	
	natural logarithm	ln()	LN()	
	Real exponential function	exp()	EXP()	

	power (base to exponent)	pow()	^	
	square root	sqrt()	SQRT()	
	factorial	factorial()	FACTORIAL()	
Constants	Pi	PI		3.1415926535898
	Two Pi	TWOPI		6.2831853071796
	Half Pi	HALFPI		1.5707963267949
	1. / PI	INVPI		0.3183098861838
	Euler	E		2.718281828459
	Golden ratio	PHI		1.6180339887499
	Square root of 2.	SQRT2		1.414213562373
	0.5 * square root 2.	SQRT1_2		0.707106781187
	Log_e_2	LN2		0.6931471805599
	Log_e_10	LN10		2.302585092994
	180. / PI	RADTODEG		57.295779513082
	PI / 180.	DEGTORAD		0.0174532925199
	Epsilon	EPSILON		1.192092896e-07
Functions SPECIAL	C random number	random()	rand()	
	Clip input	clip()	CLIP()	
	Wrap input	wrap()	WRAP()	

	Map input	map()	MAP()	
	Scale with exponent	scale()	SCALE()	
	Delta of input	delta()	DELTA()	
	Change (delta/sign)	change()	CHANGE()	Changed? (output -1, 0, 1)
	Sample and Hold	sah()	SAH()	2nd input is trigger
Converters	dB to Amplitude	DbToLin()	dbtoa()	
	Amplitude to dB	LinToDb()	atodb()	
	Pitch To Frequency	MidiToFreq()	mtof()	
	Frequency to Pitch	FreqToMidi()	ftom()	
Timebased Functions	Ramp oscillator	osc_ramp()	ramp~()	
	Sine oscillator	osc_sin()	sin~()	
	Triangle oscillator	osc_tri()	tri~()	
	Rectangular oscillator	osc_rect()	rect~()	
	Sawtooth oscillator	osc_saw()	saw~()	
	Attack hold release envelope	env_ahr()		
	Delayed attack hold helease			

	envelope	env_dahr()
	Trigger an envelope	env_trigger()
	Get the current envelope value	env_get()
	Create a smooth object	smooth_create()
	Get the interpolated value	smooth_apply()

1.2 Operator Specific Notes

1.2.1 sin()

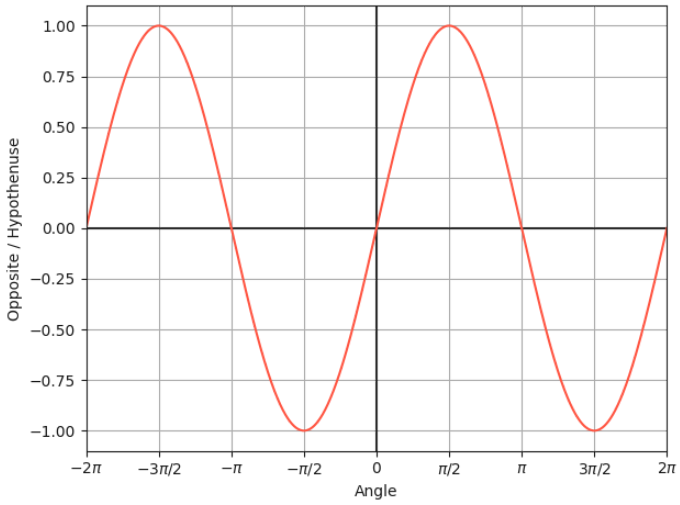
Arguments:

- Angle

Result:

- Opposite / Hypotenuse

Plot:



Examples:

```
// Result: 0
a = sin(3.141592653589793);

// Result: 0
a = sin(0);

// Result: -1
a = sin(-1.5707963267948966);

// Result: 0.7071067811865519
a = sin(134.30308594096365);
```

1.2.2 cos()

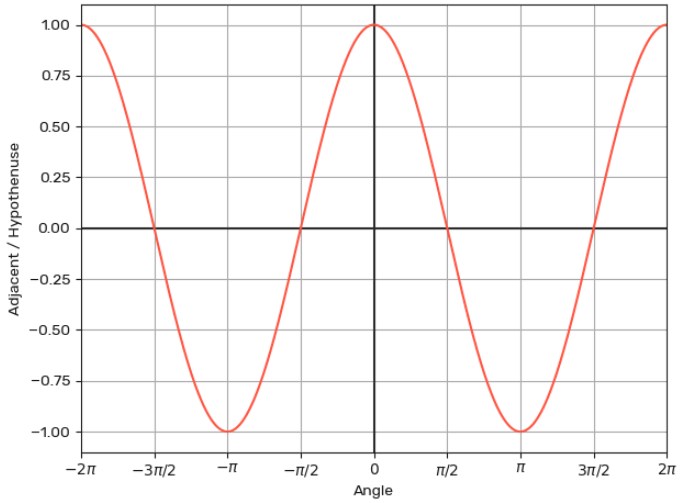
Arguments:

- Angle

Result:

- Adjacent / Hypotenuse

Plot:



Examples:

```
// Result: -1
a = cos(3.141592653589793);

// Result: 1
a = cos(0);

// Result: 0
a = cos(-1.5707963267948966);

// Result: -0.7071067811865519
a = cos(134.30308594096365);
```

1.2.3 tan()

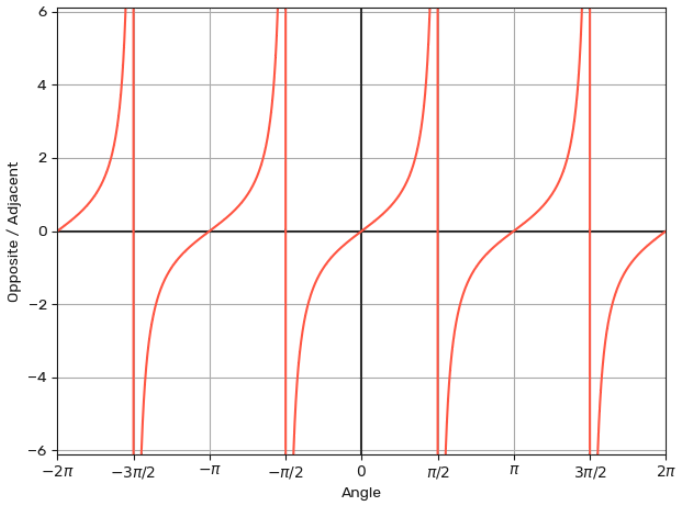
Arguments:

- Angle

Result:

- Opposite / Adjacent

Plot:



Examples:

```
// Result: 1.5574077246549023
a = tan(1);

// Result: 0
a = tan(0);

// Result: -5.4489833146366715
a = tan(-42.23);
```

1.2.4 asin()

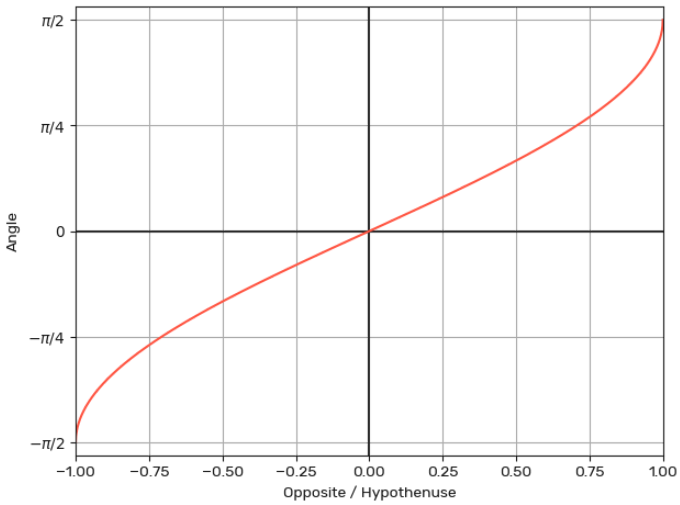
Arguments:

- Opposite / Hypotenuse

Result:

- Angle

Plot:



Examples:

```
// Result: 0
a = asin(0);

// Result: -1.5707963267948966
a = asin(-1);

// Result: 0.7853981633974545
a = asin(0.7071067811865519);
```

1.2.5 acos()

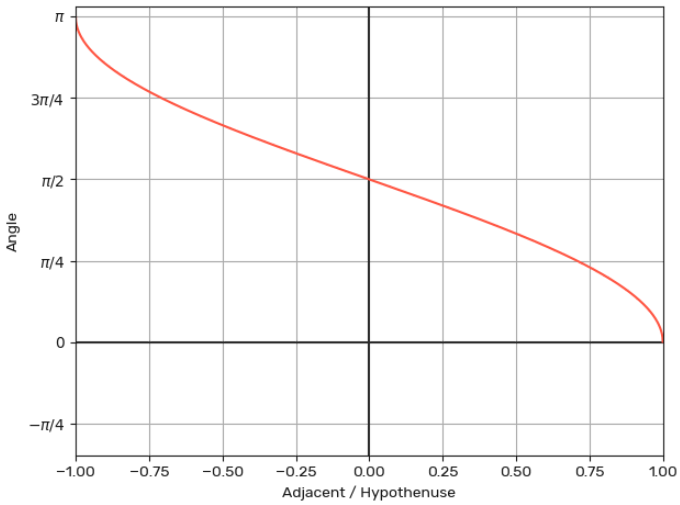
Arguments:

- Adjacent / Hypotenuse

Result:

- Angle

Plot:



Examples:

```
// Result: 0
a = acos(1);

// Result: 3.141592653589793
a = acos(-1);

// Result: 1.5707963267948966
a = acos(0);

// Result: 0.7853981633974545
a = acos(0.7071067811865519);
```

1.2.6 atan()

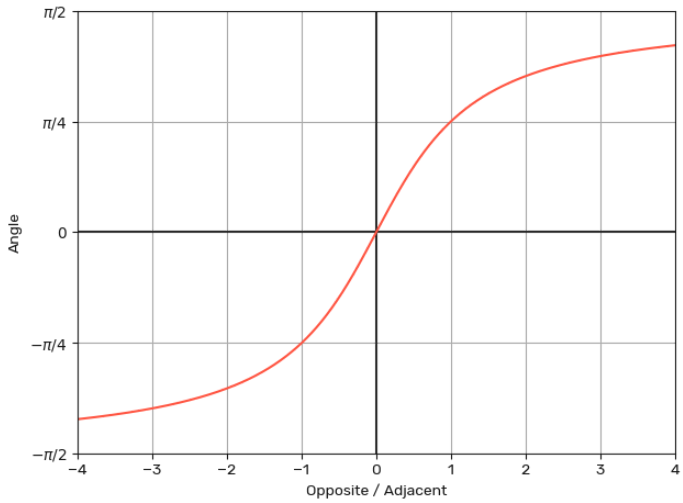
Arguments:

- Opposite / Adjacent

Result:

- Angle

Plot:



Examples:

```
// Result: 1
a = atan(1.5574077246549023);

// Result: 0
a = atan(0);

// Result: -1.3892955033326848
a = atan(-5.4489833146366715);
```

1.2.7 atan2()

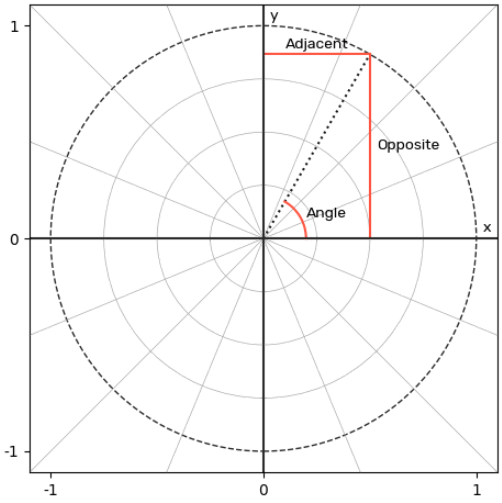
Arguments:

- Opposite
- Adjacent

Result:

- Angle

Plot:



Examples:

```
// Result: 0.7853981633974483
a = atan2(1, 1);

// Result: 1.5584969470275967
a = atan2(100, 1.23);

// Result: -1.064442034506451
a = atan2(-42.23, 23.42);
```

1.2.8 sinh()

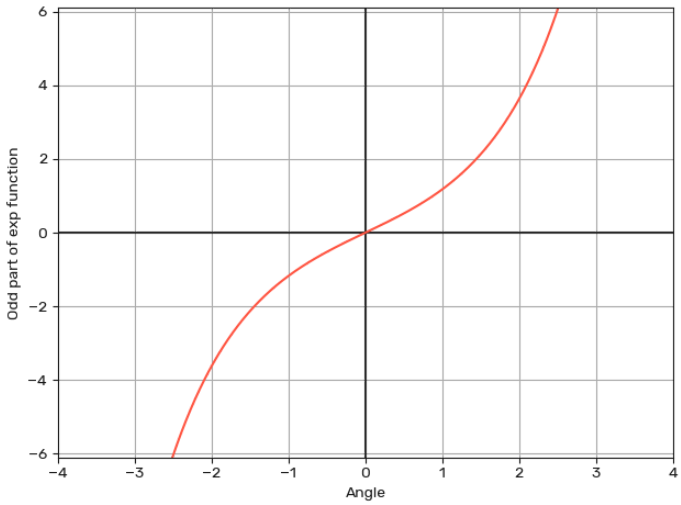
Arguments:

- Angle

Result:

- Odd part of the exponential function

Plot:



Examples:

```
// Result: 0
a = sinh(0);

// Result: -1.1752011936438014
a = sinh(-1);

// Result: 3.1175745404058084e+27
a = sinh(64);
```

1.2.9 cosh()

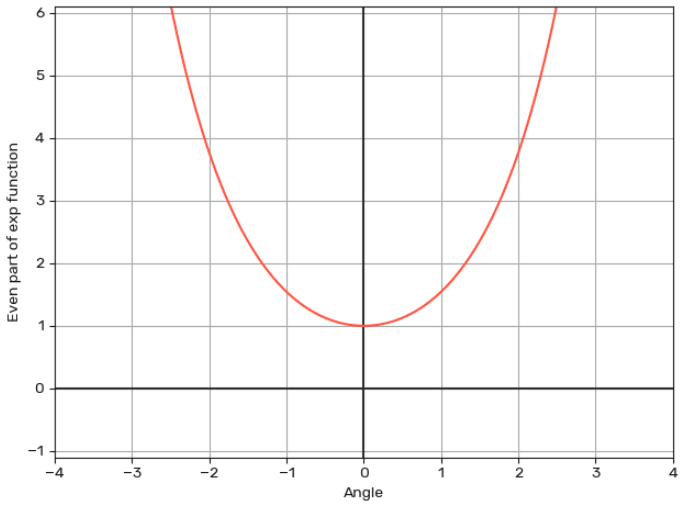
Arguments:

- Angle

Result:

- Even part of the exponential function

Plot:



Examples:

```
// Result: 1
a = cosh(0);

// Result: 33.35066330887282
a = cosh(4.2);

// Result: 33.35066330887282
a = cosh(-4.2);

// Result: 3.1175745404058084e+27
a = cosh(64);
```

1.2.10 tanh()

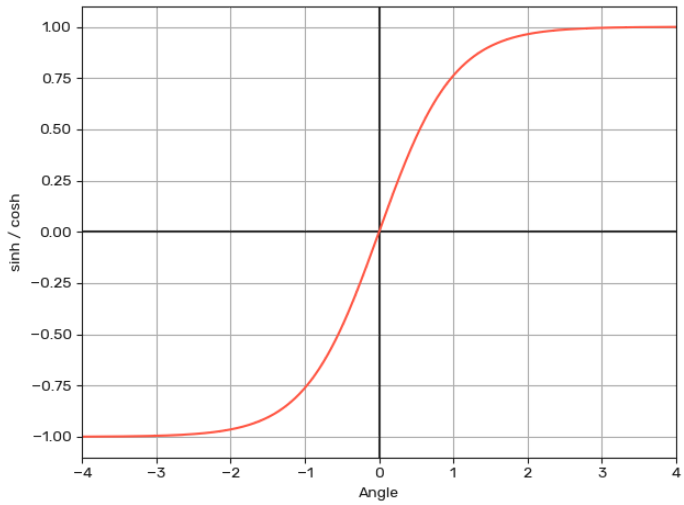
Arguments:

- Angle

Result:

- sinh / cosh

Plot:



Examples:

```
// Result: 0
a = tanh(0);

// Result: 0.7615941559557649
a = tanh(1);

// Result: -0.7615941559557649
a = tanh(-1);

// Result: 1
a = tanh(230.42);
```

1.2.11 log10()

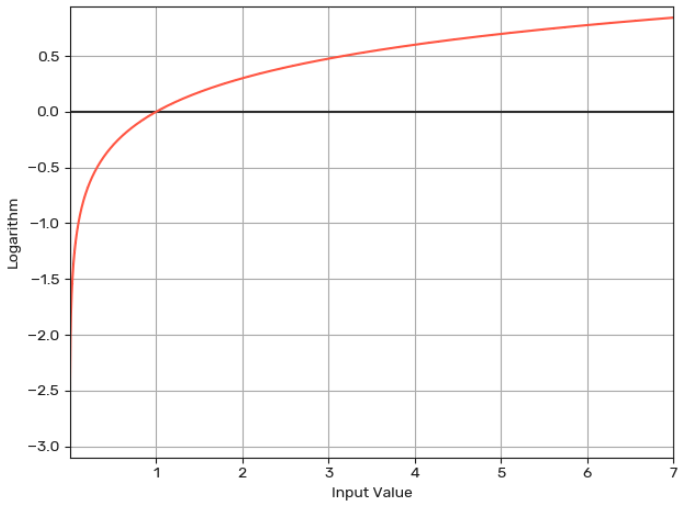
Arguments:

- Input value

Result:

- Logarithm

Plot:



Examples:

```
// Result: 0
a = log10(1);

// Result: -7
a = log10(1e-7);

// Result: 5
a = log10(100000);

// Result: 0.9817733186277472
a = log10(9.589);
```

1.2.12 log2()

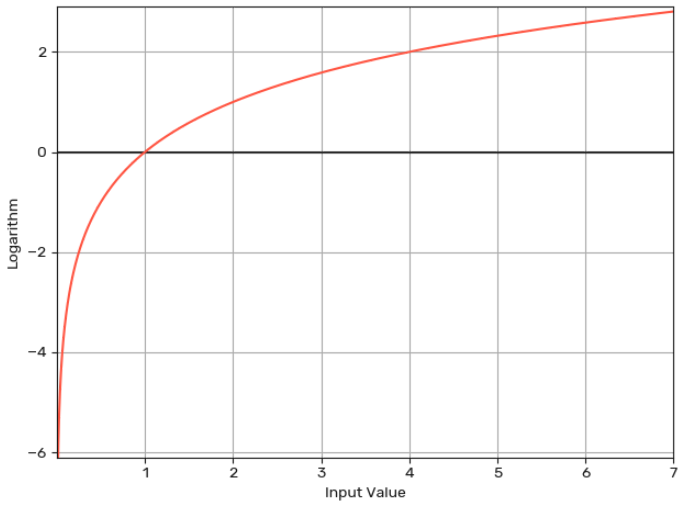
Arguments:

- Input value

Result:

- Logarithm

Plot:



Examples:

```
// Result: 0
a = log2(1);

// Result: -4
a = log2(0.0625);

// Result: 10
a = log2(1024);

// Result: 3.2613803699603157
a = log2(9.589);
```

1.2.13 ln()

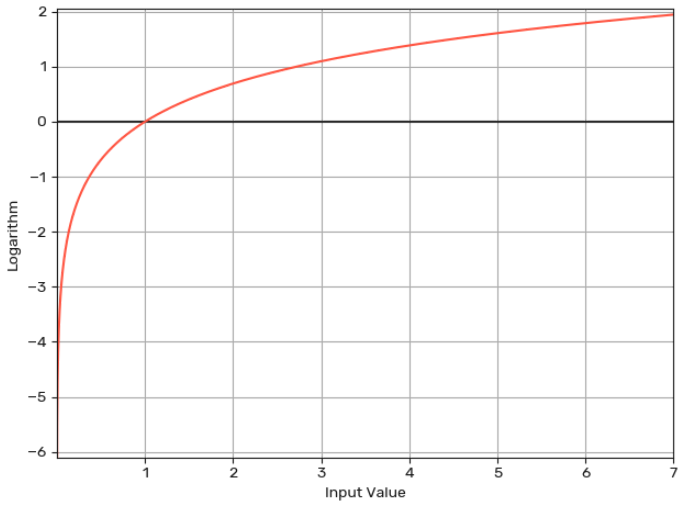
Arguments:

- Input value

Result:

- Logarithm

Plot:



Examples:

```
// Result: 0
a = ln(1);

// Result: 1
a = ln(2.718281828459045);

// Result: -2.3025850929940455
a = ln(0.1);

// Result: 10
a = ln(22026.465794806718);

// Result: 2.260616608171544
a = ln(9.589);
```

1.2.14 exp()

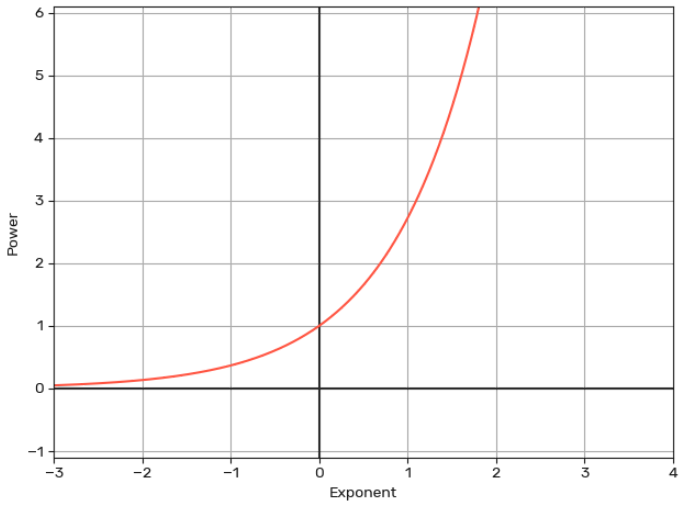
Arguments:

- Exponent

Result:

- Power

Plot:



Examples:

```
// Result: 1
a = exp(0);

// Result: 2.718281828459045
a = exp(1);

// Result: 0.1
a = exp(-2.3025850929940455);

// Result: 22026.465794806718
a = exp(10);

// Result: 9.589
a = exp(2.260616608171544);
```

1.2.15 pow()

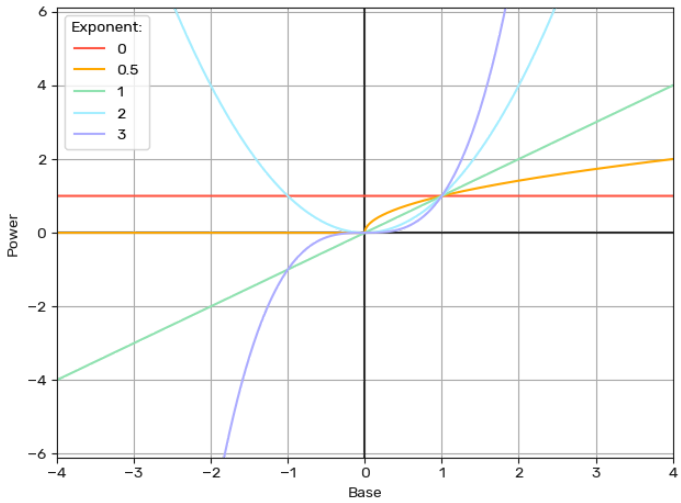
Arguments:

- Base
- Exponent

Result:

- Power

Plot:



Examples:

```
// Result: 1
a = pow(1, 42.23);

// Result: 151.7824
a = pow(-12.32, 2);

// Result: 35119.872820389246
a = pow(2, 15.1);

// Result: 0.4105896719158511
a = pow(1.23, -4.3);
```

1.2.16 sqrt()

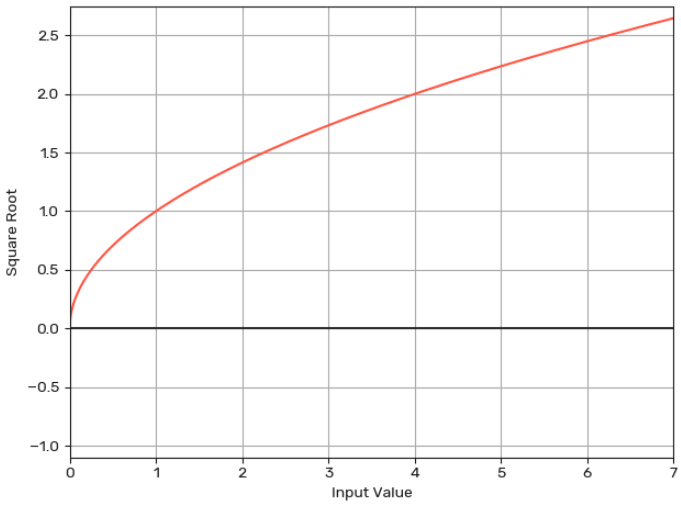
Arguments:

- Non-negative input value

Result:

- Square root of input value

Plot:



Examples:

```
// Result: 2
a = sqrt(4);

// Result: 0.7071067811865476
a = sqrt(0.5);

// Result: 0
a = sqrt(0);
```

1.2.17 random()

Arguments:

- Scaling factor

Result:

- random number between 0 and scaling factor

Description:

This function returns a random value between `0.0` and `1.0` multiplied by the scaling factor provided as argument.

1.2.18 clip()

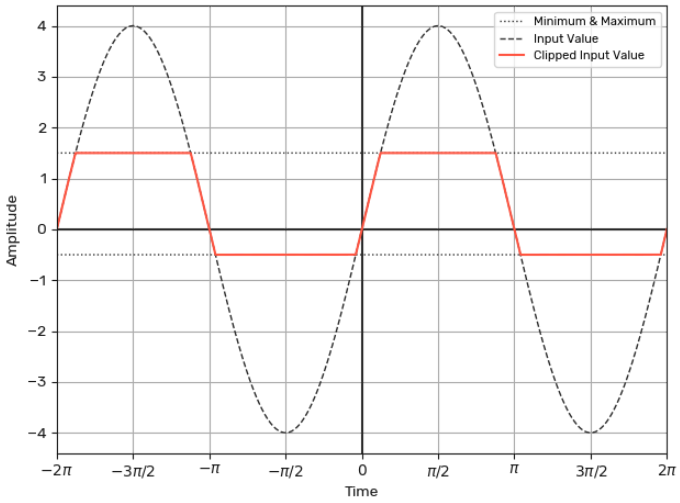
Arguments:

- Input value
- Minumum
- Maximum

Result:

- Clipped input value

Plot:



Examples:

```
// Result: 3
a = clip(4.23, -1, 3);

// Result: 25
a = clip(25, 11, 27);

// Result: -0.5
a = clip(-9.859, -0.5, 6.32);

// Result: 0.5
a = clip(1, -0.5, 0.5);
```

1.2.19 wrap()

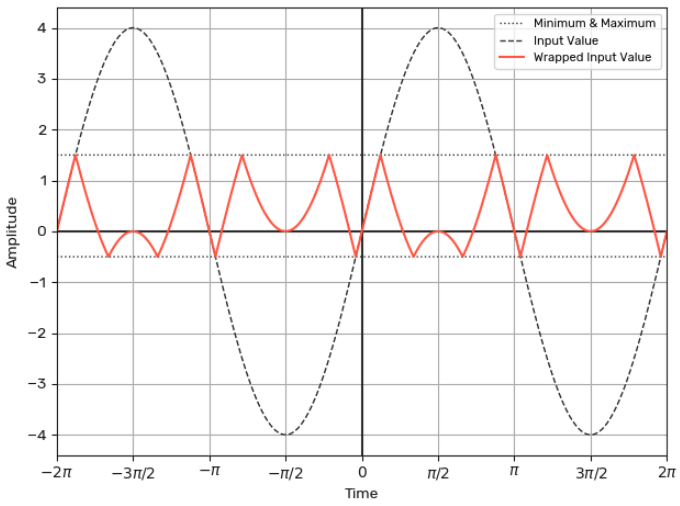
Arguments:

- Input value
- Minumum
- Maximum

Result:

- Wrapped input value

Plot:



Description:

Wraps any input value to a output range defined by `out low` and `out high`. In the following example the input value is wrapped once at the value provided by the third argument `out high`:

```
// Result: -0.5
a = wrap(1.5, -1.0, 1.0);
```

In this example the input value is wrapped twice at the value provided by the second argument `out low`:

```
// Result: -2.11
b = wrap(-22.11, -5, 5);
```

1.2.20 map()

Arguments:

- Input value
- Input minumum
- Input maximum
- Output minumum
- Output maximum

Result:

- Mapped input value

Description:

Maps any input value from a given low/high input range to any output low/high range and clamps its input. Linear. E.g.:

```
out3 = map(in2, -1.0, 1.0, 20.0, 12000.);
```

1.2.21 scale()

Arguments:

- Input value
- Input minumum
- Input maximum
- Output minumum
- Output maximum
- Exponent

Result:

- Scaled input value

Description:

Scales any input value in a given low/high input range into any output low/high range, with an optional exponent. Does not clamp input and continues operating outside of given in/out ranges. E.g.:

```
out5 = scale(in4, 0.0, 1.0, -512., 512., 2.);
```

If exponent is '1.', it behaves like `map()` with no clamp.

1.2.22 delta()

Arguments:

- Buffer array
- Input signal

Result:

- Difference between input signal and oldest buffer element

Description:

Delta is using a buffer to store a certain number of samples (N) of the input signal and then compares the input signal with the oldest element in the buffer and returns the difference.

The buffer must be declared as an array type with size N: `buffer = array(N)`

For a buffer size of `N = 5`, a typical usage of delta function would be like this:

```
buffer = array(5);
// extracts the delta from a delayed buffer (5 taps) of in1 input
signal.
out1 = delta(buffer, in1);
```

1.2.23 change()

Arguments:

- Buffer array
- Input signal

Result:

- `1`: Input signal > oldest buffer element
- `0`: Input signal == oldest buffer element
- `-1`: Input signal < oldest buffer element

Description:

Change is using a buffer to store a certain number of samples (N) of the input signal and then compares the input signal with the oldest element in the buffer.

The buffer must be declared as an array type with size N: `buffer = array(N)`

For a buffer size of `N = 2`, a typical usage of the change function would be like this:

```
buffer = array(2);


out1 = delta(buffer, in1);
```

1.2.24 sah()

Arguments:

- Buffer array
- Input signal
- Trigger signal
- Trigger threshold

Result:

- Input signal or stored value

Description:

SAH (Sample and Hold) is using a buffer to store a certain amount of samples (N) of the input signal. The function outputs the input signal (2nd argument) if the Trigger (3rd argument) is higher than the Threshold (4th argument). If Trigger input is lower than Threshold, then the SAH function outputs the previous internal stored value.

The buffer must be declared as an array type with size N:

```
buffer = array(N)
```

For a buffer size of `N = 1`, a typical usage of the sample and hold function would be like this:

```
buffer = array(1)
out1 = sah(buffer, in1, in2, thresholdValue );
```

1.2.25 osc_ramp()

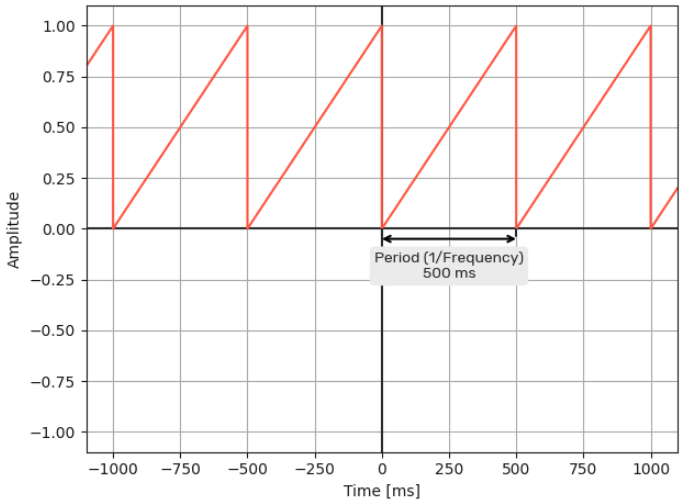
Arguments:

- Frequency (Hz)

Result:

- Amplitude

Plot:



1.2.26 osc_sin()

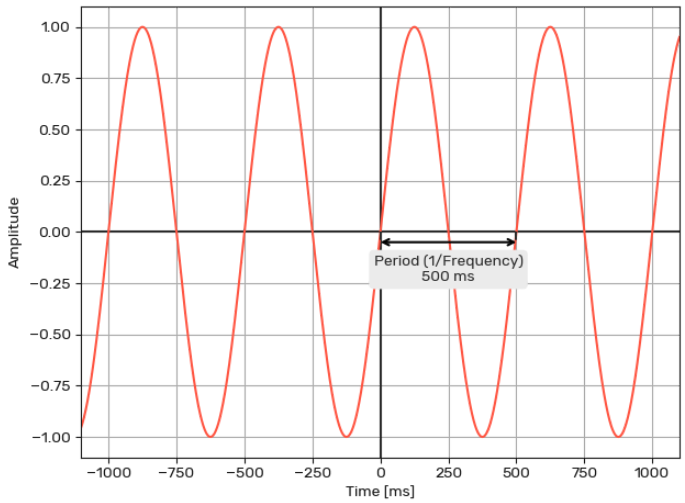
Arguments:

- Frequency (Hz)

Result:

- Amplitude

Plot:



1.2.27 osc_tri()

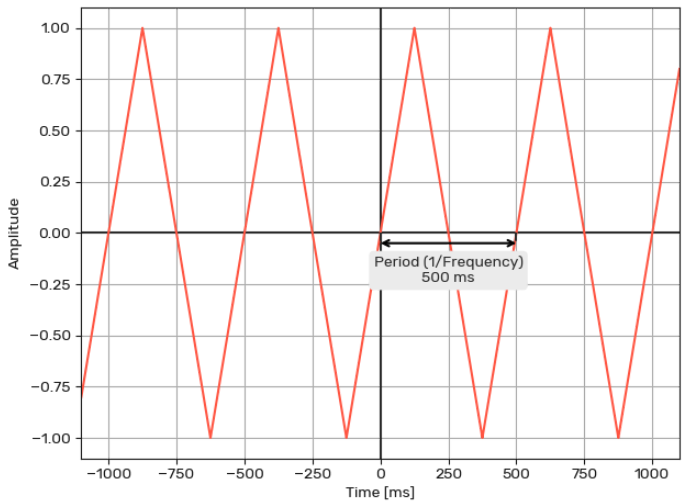
Arguments:

- Frequency (Hz)

Result:

- Amplitude

Plot:



1.2.28 osc_rect()

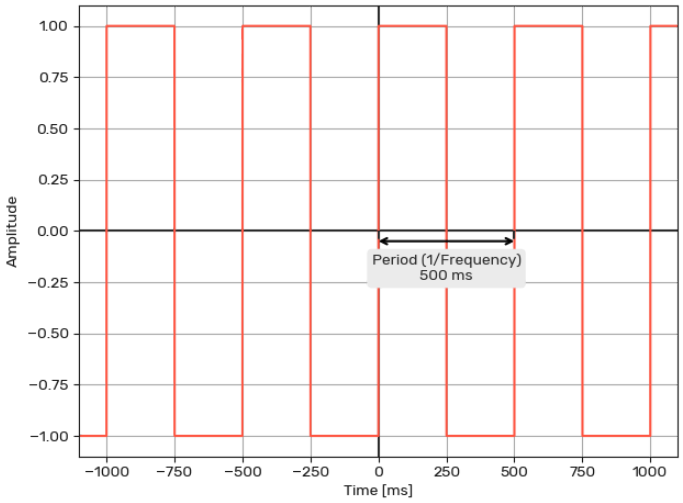
Arguments:

- Frequency (Hz)

Result:

- Amplitude

Plot:



1.2.29 osc_saw()

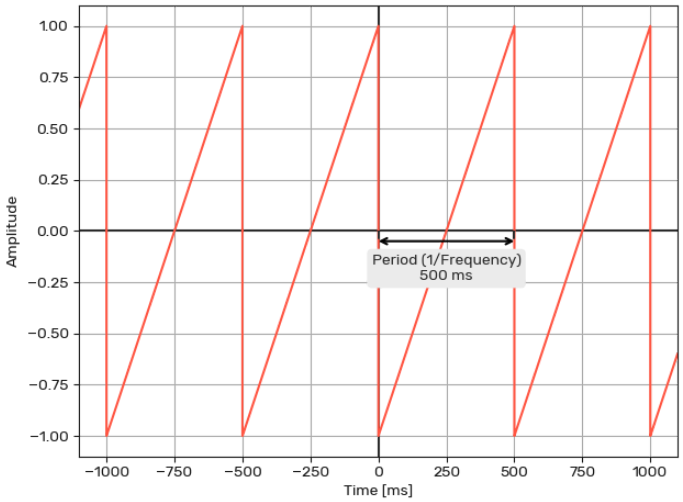
Arguments:

- Frequency (Hz)

Result:

- Amplitude

Plot:



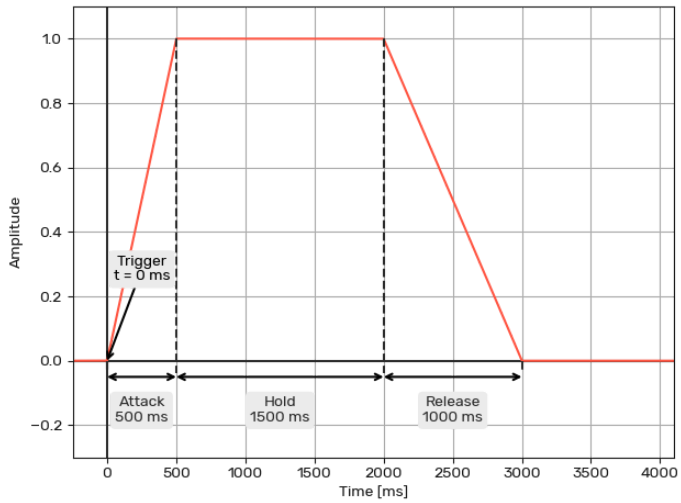
Arguments:

- Attack time (ms)
- Hold time (ms)
- Release time (ms)

Result:

- Envelope object

Plot:



1.2.31 env_dahr()

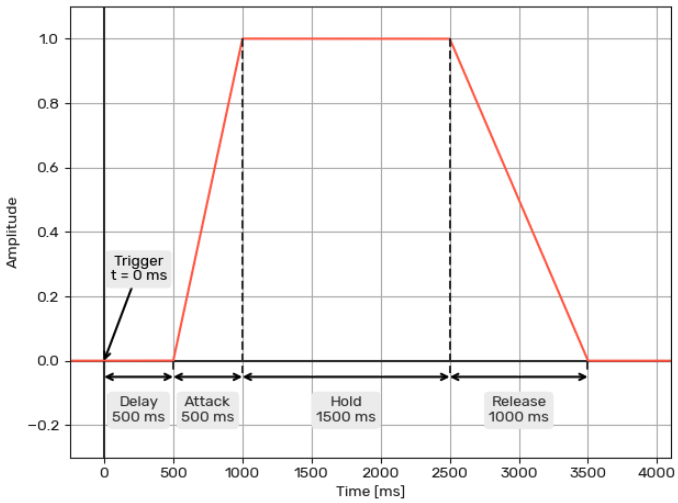
Arguments:

- Delay time (ms)
- Attack time (ms)
- Hold time (ms)
- Release time (ms)

Result:

- Envelope object

Plot:


1.0
0.8
0.6
0.4
0.2
0.0
−0.2
Amplitude
Trigger
t = 0 ms
Delay
500 ms
Attack
500 ms
Hold
1500 ms
Release
1000 ms
0
500
1000
1500
2000
2500
3000
3500
4000
Time [ms]