Conversions

midiToHz

midiToHz(*x*)

Converts a midi note value to frequency in Hertz.

Args: x: int or float

Midi note. *x* can be a number, a list or a tuple, otherwise the function returns None.

```
>>> a = (48, 60, 62, 67)
>>> b = midiToHz(a)
>>> print(b)
(130.8127826503271, 261.62556530066814, 293.66476791748823, 391.9954359818656)
>>> a = [48, 60, 62, 67]
>>> b = midiToHz(a)
>>> print(b)
[130.8127826503271, 261.62556530066814, 293.66476791748823, 391.9954359818656]
>>> b = midiToHz(60.0)
>>> print(b)
261.625565301
```

midiToTranspo

midiToTranspo(*x*)

Converts a midi note value to transposition factor (central key = 60).

Args: x: int or float

Midi note. *x* can be a number, a list or a tuple, otherwise the function returns None.

```
>>> a = (48, 60, 62, 67)
>>> b = midiToTranspo(a)
>>> print(b)
    (0.49999999999997335, 1.0, 1.122462048309383, 1.4983070768767281)
>>> a = [48, 60, 62, 67]
>>> b = midiToTranspo(a)
>>> print(b)
[0.49999999999997335, 1.0, 1.122462048309383, 1.4983070768767281]
>>> b = midiToTranspo(60.0)
>>> print(b)
1.0
```

sampsToSec

sampsToSec(*x*)

Returns the duration in seconds equivalent to the number of samples given as an argument.

Args: x: int or float

Duration in samples. *x* can be a number, a list or a tuple, otherwise function returns None.

```
>>> s = Server().boot()
>>> a = (64, 128, 256)
>>> b = sampsToSec(a)
>>> print(b)
(0.0014512471655328798, 0.0029024943310657597, 0.0058049886621315194)
>>> a = [64, 128, 256]
>>> b = sampsToSec(a)
>>> print(b)
[0.0014512471655328798, 0.0029024943310657597, 0.0058049886621315194]
>>> b = sampsToSec(8192)
>>> print(b)
0.185759637188
```

secToSamps

secToSamps(*x*)

Returns the number of samples equivalent to the duration in seconds given as an argument.

Args: x: int or float

Duration in seconds. *x* can be a number, a list or a tuple, otherwise function returns None.

```
>>> s = Server().boot()
>>> a = (0.1, 0.25, 0.5, 1)
>>> b = secToSamps(a)
>>> print(b)
(4410, 11025, 22050, 44100)
>>> a = [0.1, 0.25, 0.5, 1]
>>> b = secToSamps(a)
>>> print(b)
[4410, 11025, 22050, 44100]
>>> b = secToSamps(2.5)
>>> print(b)
110250
```

linToCosCurve

linToCosCurve(*data, yrange=[0, 1], totaldur=1, points=1024, log=False*)

Creates a cosinus interpolated curve from a list of points.

A point is a tuple (or a list) of two floats, time and value.

Args: data: list of points

Set of points between which will be inserted interpolated segments.

yrange: list of 2 floats, optional

Minimum and maximum values on the Y axis. Defaults to [0., 1.].

totaldur: float, optional

X axis duration. Defaults to 1.

points: int, optional

Number of points in the output list. Defaults to 1024.

log: boolean, optional

Set this value to True if the Y axis has a logarithmic scale. Defaults to False

```
>>> s = Server().boot()
>>> a = [(0,0), (0.25, 1), (0.33, 1), (1,0)]
>>> b = linToCosCurve(a, yrange=[0, 1], totaldur=1, points=8192)
>>> t = DataTable(size=len(b), init=[x[1] for x in b])
>>> t.view()
```

rescale

rescale(*data*, *xmin=0.0*, *xmax=1.0*, *ymin=0.0*, *ymax=1.0*, *xlog=False*, *ylog=False*)

Converts values from an input range to an output range.

This function takes data in the range *xmin* - *xmax* and returns corresponding values in the range *ymin* - *ymax*.

data can be either a number or a list. Return value is of the same type as *data* with all values rescaled.

Argss: data: float or list of floats

Values to convert.

xmin: float, optional

Minimum value of the input range.

xmax: float, optional

Maximum value of the input range.

ymin: float, optional

Minimum value of the output range.

ymax: float, optional

Maximum value of the output range.

xlog: boolean, optional

Set this argument to True if the input range has a logarithmic scaling.

ylog: boolean, optional

Set this argument to True if the output range has a logarithmic scaling.

```
>>> a = 0.5
>>> b = rescale(a, 0, 1, 20, 20000, False, True)
>>> print(b)
632.453369141
>>> a = [0, .4, .8]
>>> b = rescale(a, 0, 1, 20, 20000, False, True)
>>> print(b)
[20.000001907348633, 316.97738647460938, 5023.7705078125]
```

floatmap

floatmap(*x*, *min=0.0*, *max=1.0*, *exp=1.0*)

Converts values from a 0-1 range to an output range.

This function takes data in the range *0* - *1* and returns corresponding values in the range *min* - *max*.

Argss: x: float

Value to convert, in the range 0 to 1.

min: float, optional

Minimum value of the output range. Defaults to 0.

max: float, optional

Maximum value of the output range. Defaults to 1.

exp: float, optional

Power factor (1 (default) is linear, les than 1 is logarithmic, greter than 1 is exponential).

```
>>> a = 0.5
>>> b = floatmap(a, 0, 1, 4)
>>> print(b)
0.0625
```

distanceToSegment

distanceToSegment(*p*, *p1*, *p2*, *xmin=0.0*, *xmax=1.0*, *ymin=0.0*, *ymax=1.0*, *xlog=False*, *ylog=False*)

Find the distance from a point to a line or line segment.

This function returns the shortest distance from a point to a line segment normalized between 0 and 1.

A point is a tuple (or a list) of two floats, time and value. *p* is the point for which to find the distance from line *p1* to *p2*.

Args: p: list or tuple

Point for which to find the distance.

p1: list or tuple

First point of the segment.

p2: list or tuple

Second point of the segment.

xmin: float, optional

Minimum value on the X axis.

xmax: float, optional

Maximum value on the X axis.

ymin: float, optional

Minimum value on the Y axis.

ymax: float, optional

Maximum value on the Y axis.

xlog: boolean, optional

Set this argument to True if X axis has a logarithmic scaling.

ylog: boolean, optional

Set this argument to True if Y axis has a logarithmic scaling.

reducePoints

`reducePoints`(*pointlist*, *tolerance=0.02*)

Douglas-Peucker curve reduction algorithm.

This function receives a list of points as input and returns a simplified list by eliminating redundancies.

A point is a tuple (or a list) of two floats, time and value. A list of points looks like:

[(0, 0), (0.1, 0.7), (0.2, 0.5), ...]

Args: pointlist: list of lists or list of tuples

List of points (time, value) to filter.

tolerance: float, optional

Normalized distance threshold under which a point is excluded from the list. Defaults to 0.02.