Control Signals

Objects designed to create parameter's control at audio rate.

These objects can be used to create envelopes, line segments and conversion from python number to audio signal.

The audio streams of these objects can't be sent to the output soundcard.

Fader

class **Fader**(*fadein=0.01*, *fadeout=0.1*, *dur=0*, *mul=1*, *add=0*) [source]

Fadein - fadeout envelope generator.

Generate an amplitude envelope between 0 and 1 with control on fade times and total duration of the envelope.

The play() method starts the envelope and is not called at the object creation time.

Parent: PyoObject

Args: fadein: float, optional

Rising time of the envelope in seconds. Defaults to 0.01.

fadeout: float, optional

Falling time of the envelope in seconds. Defaults to 0.1.

dur: float, optional

Total duration of the envelope. Defaults to 0, which means wait for the stop() method to start the fadeout.

Note: The out() method is bypassed. Fader's signal can not be sent to audio outs.
The play() method starts the envelope.

The stop() method calls the envelope's release phase if *dur* = 0.

As of version 0.8.0, exponential or logarithmic envelopes can be created with the exponent factor (see setExp() method).

```
>>> s = Server().boot()
>>> s.start()
>>> f = Fader(fadein=0.5, fadeout=0.5, dur=2, mul=.5)
>>> a = BrownNoise(mul=f).mix(2).out()
>>> def repeat():
...     f.play()
>>> pat = Pattern(function=repeat, time=2).play()
```

`setFadein`(*x*) [source]

Replace the *fadein* attribute.

Args: x: float
new *fadein* attribute.

`setFadeout`(*x*) [source]

Replace the *fadeout* attribute.

Args: x: float
new *fadeout* attribute.

`setDur`(*x*) [source]

Replace the *dur* attribute.

Args: x: float
new *dur* attribute.

`setExp`(*x*) [source]

Sets an exponent factor to create exponential or logarithmic envelope.

The default value is 1.0, which means linear segments. A value higher than 1.0 will produce exponential segments while a value between 0 and 1 will produce logarithmic segments. Must be > 0.0.

Args: x: float
new *exp* attribute.

`fadein`

float. Rising time of the envelope in seconds.

`fadeout`

float. Falling time of the envelope in seconds.

`dur`

float. Total duration of the envelope.

`exp`

float. Exponent factor of the envelope.

Adsr

class `Adsr`(*attack=0.01*, *decay=0.05*, *sustain=0.707*, *release=0.1*, *dur=0*, *mul=1*, *add=0*) [source]

Attack - Decay - Sustain - Release envelope generator.

Calculates the classical ADSR envelope using linear segments. Duration can be set to 0 to give an infinite sustain. In this case, the stop() method calls the envelope release part.

The play() method starts the envelope and is not called at the object creation time.

Parent: PyoObject

Args: attack: float, optional

Duration of the attack phase in seconds. Defaults to 0.01.

decay: float, optional

Duration of the decay in seconds. Defaults to 0.05.

sustain: float, optional

Amplitude of the sustain phase. Defaults to 0.707.

release: float, optional

Duration of the release in seconds. Defaults to 0.1.

dur: float, optional

Total duration of the envelope. Defaults to 0, which means wait for the stop() method to start the release phase.

Note: The out() method is bypassed. Adsr's signal can not be sent to audio outs.
The play() method starts the envelope.

The stop() method calls the envelope's release phase if *dur* = 0.

As of version 0.8.0, exponential or logarithmic envelopes can be created with the exponent factor (see setExp() method).

```
>>> s = Server().boot()
>>> s.start()
>>> f = Adsr(attack=.01, decay=.2, sustain=.5, release=.1, dur=2, mul=.5)
>>> a = BrownNoise(mul=f).mix(2).out()
>>> def repeat():
...     f.play()
>>> pat = Pattern(function=repeat, time=2).play()
```

`setAttack`(*x*) [source]

Replace the *attack* attribute.

Args: x: float

new *attack* attribute.

`setDecay`(*x*) [source]

Replace the *decay* attribute.

Args: x: float

new *decay* attribute.

setSustain(*x*) [source]

Replace the *sustain* attribute.

Args: x: float
new *sustain* attribute.

setRelease(*x*) [source]

Replace the *sustain* attribute.

Args: x: float
new *sustain* attribute.

setDur(*x*) [source]

Replace the *dur* attribute.

Args: x: float
new *dur* attribute.

setExp(*x*) [source]

Sets an exponent factor to create exponential or logarithmic envelope.

The default value is 1.0, which means linear segments. A value higher than 1.0 will produce exponential segments while a value between 0 and 1 will produce logarithmic segments. Must be > 0.0.

Args: x: float
new *exp* attribute.

attack

float. Duration of the attack phase in seconds.

decay

float. Duration of the decay phase in seconds.

sustain

float. Amplitude of the sustain phase.

release

float. Duration of the release phase in seconds.

dur

float. Total duration of the envelope.

exp

float. Exponent factor of the envelope.

Linseg

class **Linseg**(*list*, *loop=False*, *initToFirstVal=False*, *mul=1*, *add=0*) [source]

Draw a series of line segments between specified break-points.

The play() method starts the envelope and is not called at the object creation time.

Parent: PyoObject

Args: list: list of tuples

Points used to construct the line segments. Each tuple is a new point in the form (time, value).
Times are given in seconds and must be in increasing order.

loop: boolean, optional

Looping mode. Defaults to False.

initToFirstVal: boolean, optional

If True, audio buffer will be filled at initialization with the first value of the line. Defaults to False.

Note: The out() method is bypassed. Linseg's signal can not be sent to audio outs.

```
>>> s = Server().boot()
>>> s.start()
>>> l = Linseg([(0,500),(.03,1000),(.1,700),(1,500),(2,500)], loop=True)
>>> a = Sine(freq=l, mul=.3).mix(2).out()
>>> # then call:
>>> l.play()
```

setList(*x*) [source]

Replace the *list* attribute.

Args: x: list of tuples

new *list* attribute.

replace(*x*) [source]

Alias for *setList* method.

Args: x: list of tuples

new *list* attribute.

setLoop(*x*) [source]

Replace the *loop* attribute.

Args: x: boolean

new *loop* attribute.

pause() [source]

Toggles between play and stop mode without reset.

graph(*xlen=None*, *yrange=None*, *title=None*, *wxnoserver=False*) [source]

Opens a grapher window to control the shape of the envelope.

When editing the grapher with the mouse, the new set of points will be send to the object on mouse up.

Ctrl+C with focus on the grapher will copy the list of points to the clipboard, giving an easy way to insert the new shape in a script.

Args: xlen: float, optional

Set the maximum value of the X axis of the graph. If None, the maximum value is retrieve from the current list of points.

yrange: tuple, optional

Set the min and max values of the Y axis of the graph. If None, min and max are retrieve from the current list of points.

title: string, optional

Title of the window. If none is provided, the name of the class is used.

wxnoserver: boolean, optional

With wxPython graphical toolkit, if True, tells the interpreter that there will be no server window.

If *wxnoserver* is set to True, the interpreter will not wait for the server GUI before showing the controller window.

list

float. List of points (time, value).

loop

boolean. Looping mode.

Expseg

class **Expseg**(*list*, *loop=False*, *exp=10*, *inverse=True*, *initToFirstVal=False*, *mul=1*, *add=0*) [source]

Draw a series of exponential segments between specified break-points.

The play() method starts the envelope and is not called at the object creation time.

Parent: PyoObject

Args: list: list of tuples

Points used to construct the line segments. Each tuple is a new point in the form (time, value).
Times are given in seconds and must be in increasing order.

loop: boolean, optional

Looping mode. Defaults to False.

exp: float, optional

Exponent factor. Used to control the slope of the curves. Defaults to 10.

inverse: boolean, optional

If True, downward slope will be inversed. Useful to create biexponential curves. Defaults to True.

initToFirstVal: boolean, optional

If True, audio buffer will be filled at initialization with the first value of the line. Defaults to False.

Note: The out() method is bypassed. Expseg's signal can not be sent to audio outs.

```
>>> s = Server().boot()
>>> s.start()
>>> l = Expseg([(0,500),(.03,1000),(.1,700),(1,500),(2,500)], loop=True)
>>> a = Sine(freq=l, mul=.3).mix(2).out()
>>> # then call:
>>> l.play()
```

`setList(x)` [source]

Replace the *list* attribute.

Args: x: list of tuples

new *list* attribute.

`setLoop(x)` [source]

Replace the *loop* attribute.

Args: x: boolean

new *loop* attribute.

`setExp(x)` [source]

Replace the *exp* attribute.

Args: x: float

new *exp* attribute.

setInverse(*x*) [source]

Replace the *inverse* attribute.

Args: x: boolean

new *inverse* attribute.

replace(*x*) [source]

Alias for *setList* method.

Args: x: list of tuples

new *list* attribute.

pause() [source]

Toggles between play and stop mode without reset.

graph(*xlen=None*, *yrange=None*, *title=None*, *wxnoserver=False*) [source]

Opens a grapher window to control the shape of the envelope.

When editing the grapher with the mouse, the new set of points will be send to the object on mouse up.

Ctrl+C with focus on the grapher will copy the list of points to the clipboard, giving an easy way to insert the new shape in a script.

Args: xlen: float, optional

Set the maximum value of the X axis of the graph. If None, the maximum value is retrieve from the current list of points. Defaults to None.

yrange: tuple, optional

Set the min and max values of the Y axis of the graph. If None, min and max are retrieve from the current list of points. Defaults to None.

title: string, optional

Title of the window. If none is provided, the name of the class is used.

wxnoserver: boolean, optional

With wxPython graphical toolkit, if True, tells the interpreter that there will be no server window.

If *wxnoserver* is set to True, the interpreter will not wait for the server GUI before showing the controller window.

list

float. List of points (time, value).

loop

boolean. Looping mode.

exp

float. Exponent factor.

inverse

boolean. Inverse downward slope.

Sig

class **Sig**(*value*, *mul=1*, *add=0*) [source]

Convert numeric value to PyoObject signal.

Parent: PyoObject

Args: value: float or PyoObject
Numerical value to convert.

```
>>> import random
>>> s = Server().boot()
>>> s.start()
>>> fr = Sig(value=400)
>>> p = Port(fr, risetime=0.001, falltime=0.001)
>>> a = SineLoop(freq=p, feedback=0.08, mul=.3).out()
>>> b = SineLoop(freq=p*1.005, feedback=0.08, mul=.3).out(1)
>>> def pick_new_freq():
...     fr.value = random.randrange(300,601,50)
>>> pat = Pattern(function=pick_new_freq, time=0.5).play()
```

setValue(*x*) [source]

Changes the value of the signal stream.

Args: x: float or PyoObject
Numerical value to convert.

value

float or PyoObject. Numerical value to convert.

SigTo

class **SigTo**(*value*, *time=0.025*, *init=0.0*, *mul=1*, *add=0*) [source]

Convert numeric value to PyoObject signal with portamento.

When *value* is changed, a ramp is applied from the current value to the new value. Can be used with PyoObject to apply a linear portamento on an audio signal.

Parent: PyoObject

Args: value: float or PyoObject

Numerical value to convert.

time: float or PyoObject, optional

Ramp time, in seconds, to reach the new value. Defaults to 0.025.

init: float, optional

Initial value of the internal memory. Defaults to 0.

Note: The out() method is bypassed. SigTo's signal can not be sent to audio outs.

```
>>> import random
>>> s = Server().boot()
>>> s.start()
>>> fr = SigTo(value=200, time=0.5, init=200)
>>> a = SineLoop(freq=fr, feedback=0.08, mul=.3).out()
>>> b = SineLoop(freq=fr*1.005, feedback=0.08, mul=.3).out(1)
>>> def pick_new_freq():
...     fr.value = random.randrange(200,501,50)
>>> pat = Pattern(function=pick_new_freq, time=1).play()
```

`setValue`(*x*) [source]

Changes the value of the signal stream.

Args: x: float or PyoObject

Numerical value to convert.

`setTime`(*x*) [source]

Changes the ramp time of the object.

Args: x: float or PyoObject

New ramp time in seconds.

`value`

float or PyoObject. Numerical value to convert.

`time`

floator PyoObject. Ramp time in seconds.