Litter Power Package

Documentation Addenda

New objects since Litter Power 1.0
Current Litter Power Package version: 1.1.1
Last modified March 17, 2003

Litter Power Thesaurus

Thesaurus

1/f distribution	**lp.sss**, **lp.zzz**, **lp.sss~**, **lp.zzz~**
Aliasing	**lp.nn~**
Amesian feedback	**lp.ernie**
Arc sine distribution	**lp.abbie**
Bernoulli Trials	**lp.bernie**
Beta distribution	**lp.abbie**
Bilateral exponential distribution	**lp.expo**
Binary Choice	**lp.bernie**
Bit resolution reduction	**lp.nn~**
Black noise	**lp.phhh**, **lp.phhh~**
Brown noise	**lp.pfff**, **lp.pfff~**
Brownian motion	**lp.pfff**, **lp.pfff~**
Cartesion to Polar coordinates	**lp.c2p~**
Cauchy distribution	**lp.coshy**
Chaos	**lp.ccc**, **lp.ccc~**, **lp.lya**, **lp.poppy**, **lp.poppy~**
Chi-Square distribution	**lp.chichi**
Clipping values to range	**lp.scampf**, **lp.scampi**, **lp.scamp~**
Coin tosses	**lp.bernie**, **lp.ginger**
Colored noise	**lp.pfff~**, **lp.phhh~**, **lp.pvvv~**, **lp.shhh~**, **lp.sss~**, **lp.zzz~**
Conversion	**lp.c2p~**, **lp.p2c~**
Count of events	**lp.stacey**
Dice	**lp.dicey**
Dither	**lp.gsss~**, **lp.nn~**, **lp.trrr~**
Dust noise	**lp.ppp~**
Erlang distribution	**lp.gammer**
Exponential distribution	**lp.expo**, **lp.y**, **lp.gammer**
First Law of Laplace	**lp.expo**
Fisher distribution	**lp.fishie**
Floating-point interval mutation	**lp.vim**
Fractal noise	**lp.pfff**, **lp.pfff~**
Frequency-domain interval mutation	**lp.frim~**
Gamma distribution	**lp.gammer**
Gauss distribution	**lp.gsss~**, **lp.norm**
Gray noise	**lp.grrr**, **lp.grrr~**
Hyperbolic cosine distribution	**lp.hyppie**
I Ching	**lp.ginger**, **lp.kg**, **lp.i**
Interval mutation	**lp.frim~**, **lp.tim~**, **lp.vim**
Kurtosis	**lp.stacey**
Laplace distribution	**lp.expo**
Limiting to range	**lp.scampf**, **lp.scampi**, **lp.scamp~**
Linear congruence	**lp.lili**, **lp.lll~**
Linear distribution	**lp.linnie**
Logistic distribution	**lp.loggie**
Log-normal distribution	**lp.lonnie**
Low frequency noise	**lp.frrr~**
Lyapunov space	**lp.lya**
Map values	**lp.expo**, **lp.hyppie**, **lp.linnie**, **lp.loggie**, **lp.scampf**, **lp.scampi**, **lp.scamp~**, **lp.kg**
Maximum	**lp.stacey**
McCartney Pink noise	**lp.zzz**, **lp.zzz~**
Mean	**lp.stacey**
Minimum	**lp.stacey**
Morphological mutation	**lp.frim~**, **lp.tim~**, **lp.vim**
Negative Cauchy distribution	**lp.coshy**

Litter Power Thesaurus

Negative exponential distribution	**lp.expo**
Noise	**lp.frrr~**, **lp.grrr**, **lp.grrr~**, **lp.gsss~**, **lp.norm**, **lp.pfff**, **lp.pfff~**, **lp.phhh**, **lp.phhh~**, **lp.ppp~**, **lp.shhh**, **lp.shhh~**, **lp.sss**, **lp.sss~**, **lp.tata**, **lp.titi**, **lp.trrr~**, **lp.zzz**, **lp.zzz~**
Normal distribution	**lp.gsss~**, **lp.norm**
Parametric linear congruence	**lp.lili**, **lp.lll~**
Phase unwrapping	**lp.grl~**
Pink Noise	**lp.sss**, **lp.zzz**, **lp.sss~**, **lp.zzz~**
Poisson distribution	**lp.pfishie**
Polar to Cartesian coordinates	**lp.p2c~**
Popcorn noise	**lp.ppp~**
Population growth model	**lp.poppy**, **lp.poppy~**
Positive Cauchy distribution	**lp.coshy**
Random walk	**lp.pfff**, **lp.pfff~**
Range limiting	**lp. scampf**, **lp.scampi**, **lp.scamp~**
Rayleigh distribution	**lp.y**
Red noise	**lp.pfff**, **lp.pfff~**
Reflecting values into range	**lp.scampf**, **lp.scampi**, **lp.scamp~**
Sample-and-hold noise	**lp.frrr~**
Sample rate reduction	**lp.nn~**
Scale values	**lp.scampf**, **lp.scampi**
Schuster/Procaccia algorithm	**lp.ccc**, **lp.ccc~**
Signal degradation	**lp.nn~**
Skew (statistical)	**lp.stacey**
Spectral mutation	**lp.frim~**
Standard deviation	**lp.stacey**
Statistics	**lp.stacey**
Student's "T" distribution	**lp.stu**
"T" distribution	**lp.stu**
Tausworthe 88 random number algorithm	**lp.tata**
Time domain mutation	**lp.tim~**
Triangular distribution	**lp.linnie**, **lp.trrr~**
TT800 random number algorithm	**lp.titi**
Uniform distribution	**lp.shhh**, **lp.shhh~**, **lp.tata**, **lp.titi**
Urn model	**lp.ernie**
Variable color noise	**lp.pvvv**, **lp.pvvv~**
Voss/Gardner algorithm	**lp.sss**, **lp.sss~**
Voss/McCartney algorithm	**lp.zzz**, **lp.zzz~**
Weibull distribution	**lp.y**
White noise	**lp.titi**, **lp.shhh**, **lp.shhh~**
Wrapping values into range	**lp.scampf**, **lp.scampi**, **lp.scampi~**

lp.ccc

Pro Bundles only

Generate chaotic sequences with 1/f distribution (Schuster/Proccacia algorithm)

Generate sequences of values in the range 0 < x < 1 using the interative formula

$$x' = \langle x + x^2 \rangle \bmod 1$$

proposed by Schuster and Proccacia as a method for generating a 1/f spectrum.

The sequence has a rather different look from other 1/f generators (such as **lp.sss** and **lp.zzz**). The **lp.ccc** sequence is characterized by long series of values close to zero with sudden chaotic outbursts of larger values. The implementation of the Schuster/Proccacia algorithm used here is guaranteed never to degenerate to an infinite sequence of zeros, although there may be times when you start to doubt this. Trust me.

Input

bang	Generate next value in the sequence and send it out the outlet.
float	Set *x* to the input value, generate the following value in the sequence, and send it out the outlet. Values of *x* outside the range $0 < x < 1$ will not be used.

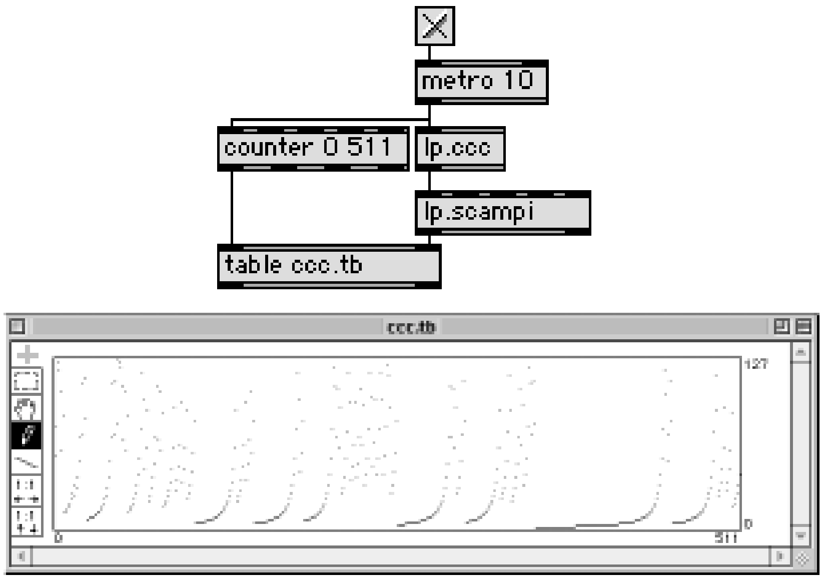
Arguments

float	Optional initial value for *x* (the "seed"). If specified, the value must be in the range $0 < x < 1$. If not specified, **lp.ccc** will generate a seed based on an algorithm so non-deterministic that your computer would have a nervous breakdown if I revealed it here.

Output

float	The next value from the Schuster/Proccacia sequence.

Examples



Lp.ccc in action.

What’s in a name?

Kaspar T. Toeplitz wondered if the name was a clipped form of Cosmic Coincidence Control Center. In fact, the name is simply in the tradition of the other 1/f generators, **lp.sss** and **lp.zzz**. But it is a cosmic coincidence.

See Also

lp.ccc~	Chaotic noise with 1/f spectrum (Schuster/Procaccia algorithm)
lp.sss	Generate random numbers from a 1/f (“pink”) distribution (Voss/Gardner algorithm)
lp.zzz	Generate random numbers from a 1/f (“pink”) distribution (McCartney algorithm)

Schuster, H.G., *Deterministic Chaos: An Introduction*. Physik Verlag, Weinheim, 1984.

Procaccia, I. and Schuster, H.G. “Functional renormalisation group theory of universal 1/f noise in dynamical systems,“ *Physics Review* 28 A, 1983.

Noise with a 1/f spectrum using the Schuster/Procaccia chaotic algorithm

The sequence has a rather different look from other 1/f generators (such as **lp.sss~** and **lp.zzz~**). Noise from **lp.ccc~** is characterized by long periods of values close to zero with sudden chaotic outbursts of noise. The implementation of the Schuster/Proccacia algorithm used here is guaranteed never to degenerate to a zero signal, although there may be times when you wonder. Patience.

Input

None.

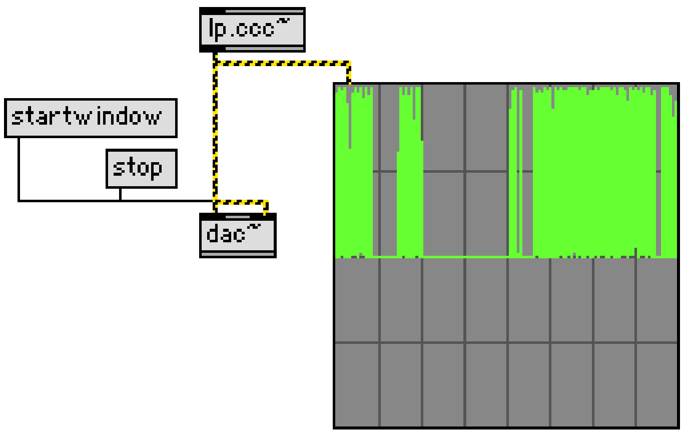
Arguments

None.

Output

signal	1/f noise.

Examples



Lp.ccc~ in action.

What's in a name?

See **lp.ccc**.

See Also

lp.ccc	Generate chaotic sequences with 1/f distribution (Schuster/Proccacia algorithm)
lp.sss~	Generate random numbers from a 1/f ("pink") distribution (Voss/Gardner algorithm)
lp.zzz~	Generate random numbers from a 1/f ("pink") distribution (McCartney algorithm)

Schuster, H.G., *Deterministic Chaos: An Introduction*. Physik Verlag, Weinheim, 1984.

Procaccia, I. and Schuster, H.G. "Functional renormalisation group theory of universal 1/f noise in dynamical systems," *Physics Review* 28 A, 1983.

lp.delta

Pro Bundles only

Input

int	An int in either inlet calculates the differences (left - right, right - left, and absolute difference), sending the result through the three outlets.
float	A float in either inlet calculates the differences (left - right, right - left, and absolute difference), sending the result through the three outlets. If at least one initialization argument was a float, all calculations will be performed in floating point and the result will be sent through the outlets as floats. Otherwise any incoming floats will be truncated to an integer value before the subtraction is performed
bang	Send last difference calculated through the outlets.
set	The symbol set followed by a number updates the value associated with the inlet to be updated without sending the resulting differences through the outlets.
list	You can send a list of two numbers (floats or ints) to the left inlet. The first number will be sent to the left inlet, the second number to the right inlet. The differences are only calculated and sent through the outlets once.

Arguments

int
float

The number of initialization arguments determines the intial values of the inlet and the type of calculation (integer of floating point)..

If there are no initialization arguments both inlets are initialized to zero. If there is one argument, it initializes the value stored at the first inlet. Specify two arguments to initialize the values stored at both inlets.

If any of the initialization arguments is a float, then all calculation will be performed in floating point and the result will be sent through the outlet as a float. Otherwise all calculations will be performed with integers and floats arriving at the inlets will be truncated.

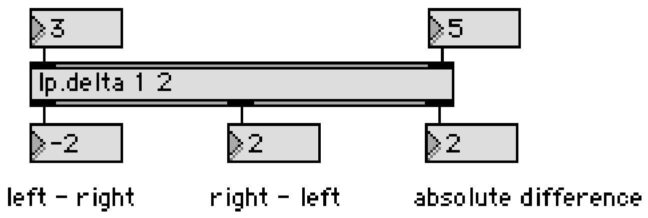
Output

int
float

The outlets, from left to right, are: right inlet subtracted from left inlet; left inlet subtracted from right inlet; absolute difference.

The type of value (int or float) depends upon the initialization arguments.

Examples

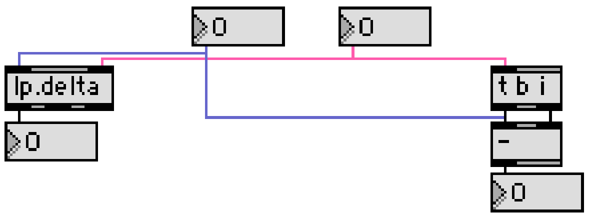


Lp.delta *provides all the difference you could want.*

lp.delta

Calculate differences between two numbers

Pro Bundles only



I got tired of having to include trigger objects to get the effect I wanted.

What's in a name?

Name of the Greek letter used to represent differences.

See Also

-	Subtract two numbers
abs	Absolute value
lp.sigma	Calculate differences between two numbers
lp.logos	Calculate quotient and remainder of two numbers
lp.pi	Multiply numbers
trigger	Sends its input to many places, in right-to-left order

lp.gsss~

Noise source with Gaussian distribution. When used as a dither signal, the result is slightly less rough than **lp.trrr~**. This external is, however, computationally more expensive. Also, the output signal is not bounded, although it is extremely rare for samples with more than about ±3 times the standard deviation to be produced.

Input

float	In left inlet: set the mean (central) value of the Gaussian distribution. This is effectively a DC offset to the signal.
	In the right inlet: set the standard deviation of the Gaussian distribution. This is effectively a scaling factor.
signal	In left inlet: override the mean of the Gaussian distribution. This is a simple way of adding Gaussian noise to the incoming signal
	In right inlet: override the standard deviation of the Gaussian distribution. This is an unusual thing to do in dithering applications, but you are welcome to use this feature as you see fit.

Argument

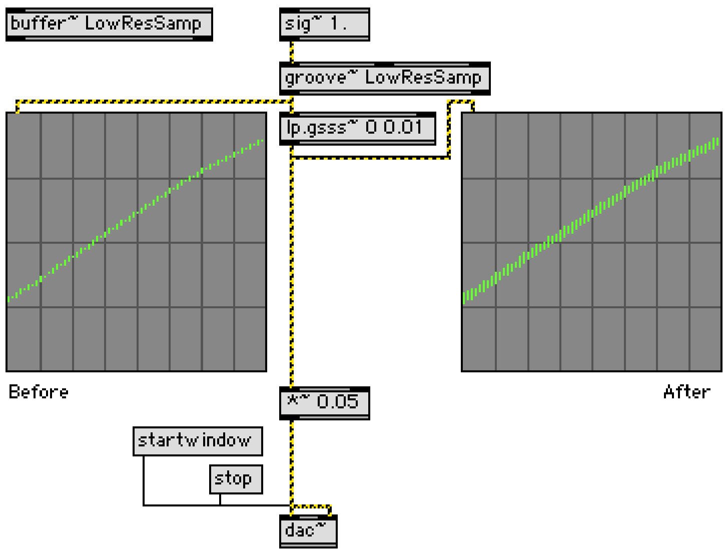
You can initialize an **lp.gsss~** object with up to two optional arguments. You must specify the first argument if you want to specify the second.

float	First argument: initial mean. Default: zero.
	Second argument: initial standard deviation. Default: 0.40828, $\sqrt{\frac{1}{6}}$. This is the same standard deviation as produced by an unscaled triangular noise signal.

Output

signal	Gaussian noise

Examples



Adding dither can take the edge off signals originally sampled with low resolution.

What’s in a name?

Named after the famous mathematician.

See Also

lp.nn~	G3nral-purpoz s!gnl degrd8!on: rduz kap!talizt resolut!on | d!ther | m0d!fy faz!st smpl-r8
lp.norm	Generate random numbers using the Tausworthe 88 algorithm
lp.pfff~	Brown noise
lp.phhh~	Black noise
lp.pvvv~	Colored noise with variable Hurst exponent
lp.sss~	Pink noise (Gardner/Voss algorithm)
lp.trrr~	Triangular (dither) noise
lp.zzz~	Pink noise (McCartney algorithm)

Calculate quotient and remainder of two numbers

lp.logos

Pro Bundles only

Input

int	An int in either inlet calculates the quotient and remainder of two numbers, sending the quotient through the left outlet and the remainder through the right outlet.
float	A float in either inlet calculates the quotient and remainder of two numbers. If at least one initialization argument was a float, the calculations will be performed in floating point and the results will be sent through the outlets as floats. Otherwise any incoming floats will be truncated to an integer value before the division is performed
bang	Send last quotient and remainder calculated through the outlets.
set	The symbol set followed by a number updates the value associated with the inlet to be updated without sending the resulting values through the outlets.
list	You can send a list of two numbers (floats or ints) to the left inlet. The first number will be sent to the left inlet, the second number to the right inlet. The quotient and remainder are calculated and sent through the outlets once.

Arguments

int
float

The number of initialization arguments determines the initial values of the inlet and the type of calculation (integer of floating point).

If there are no initialization arguments both inlets are initialized to one. If there is one argument, it initializes the value stored at the first inlet. Specify two arguments to initialize the values stored at both inlets.

If any of the initialization arguments is a float, then all calculation will be performed in floating point and the result will be sent through the outlet as a float. Otherwise all calculations will be performed with integers and floats arriving at the inlets will be truncated.

Yes, Virginia: the remainder (modulo) for floating point values is calculated.

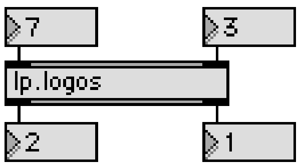
Output

int
float

The quotient of the left inlet divided by the right inlet is sent through the left outlet. The remainder is sent through the right outlet.

The type of value (int or float) depends upon the initialization arguments.

Examples

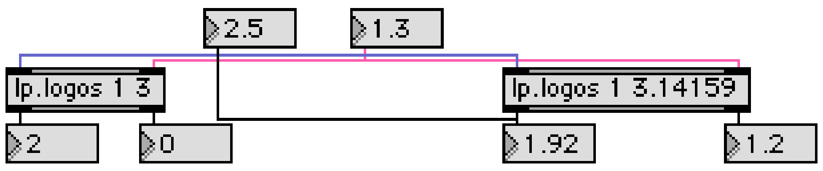


Lp.logos *in action.*

lp.logos

Calculate quotient and remainder of two numbers

Pro Bundles only



We can do both integer and floating point. Remainder in floating point, too!

What’s in a name?

Since mathematicians have not adopted a Greek letter to represent division, I took the Greek word for quotient (or ratio).

> *’Εν ἀρχῇ ἦν ὁ λόγος, καὶ ὁ λόγος ἦν πρὸς τὸν θεόν, καὶ θεὸς ἦν ὁ λόγος… πάντα δι’ αὐτοῦ ἐγένετο, καὶ χωρὶσ αὐτοῦ ἐγένετο οὐδὲ ἕν.*
>
> *Kata Iwannen*

See Also

/	Divide two numbers
%	Modulo
fmod	Modulo
lp.delta	Calculate ratios
lp.sigma	Calculate differences
lp.pi	Multiply numbers
trigger	Sends its input to many places, in right-to-left order

lp.lya

Calculate Lyapunov exponent for population growth models

Pro Bundles only

The Lyapunov exponent is a measure of how chaotic a dynamic system is. Negative values indicate a stable system and positive values indicate chaotic behavior.

Calculation of the Lyapunov exponent for arbitrary dynamic systems can be quite complex, but there is a relatively simple algorithm for approximating the value for dynamic systems based on the simple population growth model (see **lp.poppy** for further details on this model). The algorithm involves calculating several thousand iterations of the population growth, so it's not exactly computationally cheap.

Lp.lya is particularly interesting for population growth models which cycle through multiple growth rates. See lp.lya.help for musical and visual examples. The self-similarity of Lyapunov spaces is particularly apparent in two-dimensional images (see the example below).

Input

bang	Send the value of the Lyapunov exponent for the current population growth model out the outlet.
float	In the left inlet sets the growth rate, calculates the Lyapunov exponent, and sends the result out the outlet. Input values are clipped to the range $0 \le r \le 4$v
list	This is where **lp.lya** gets interesting. You can send a list of floats to the left inlet thereby defining a cycle of growth rates. The Lyapunov exponent is calculated and sent out the outlet. All values in the list are clipped to the range $0 \le r \le 4$.
int	An integer in the right inlet specifies the precision with which the Lyapunov exponent is estimated.The value, multiplied by one thousand, is the number of iterations of the population growth model that must be calculated. Default is one, which is also the minimum value. Non-positive input resets this to the default.

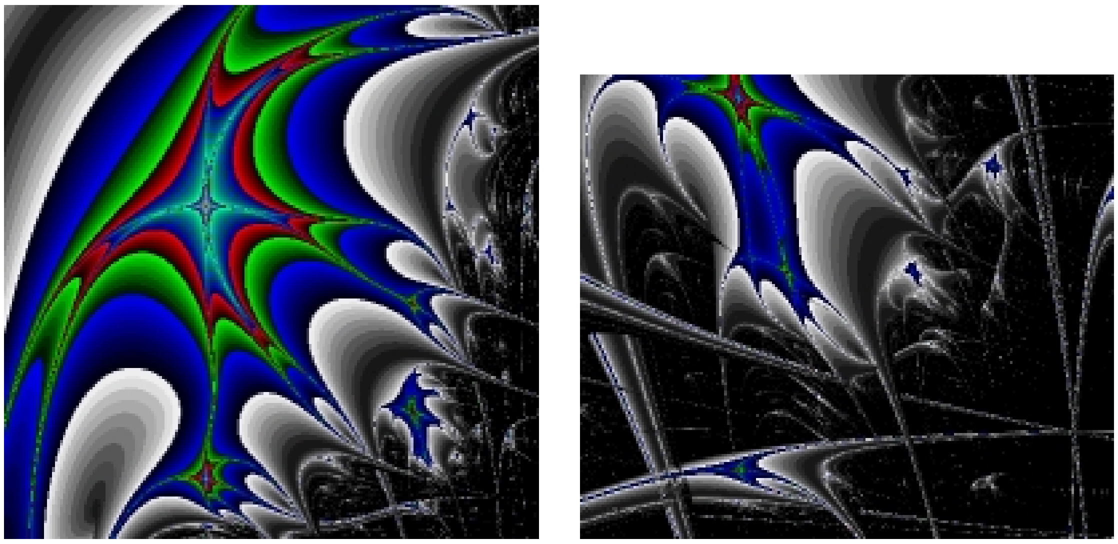
Arguments

int	Optional. Initial value for precision. Default is one.

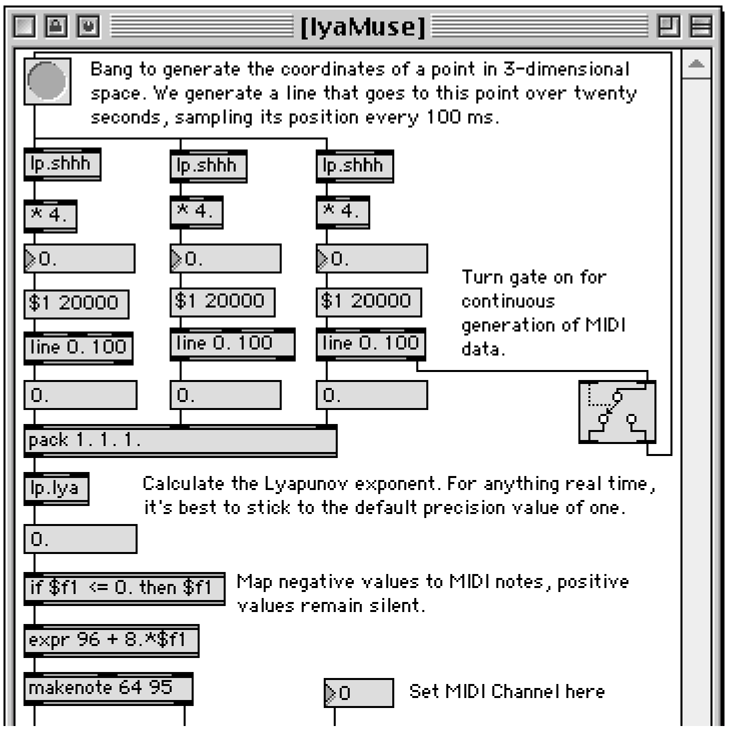
Output

float	The Lyapunov exponent of the current population growth model.

Examples



Images generated by **lp.lya***. Pixels representing chaotic parameters for the underlying population growth model are rendered in black; other pixels are colored according to how stable the growth model is at that point. The image at the right is a magnification of the lower-right quadrant of the image at the left.*



[lyaMuse]

Bang to generate the coordinates of a point in 3-dimensional space. We generate a line that goes to this point over twenty seconds, sampling its position every 100 ms.

```
lp.shhh        lp.shhh        lp.shhh
* 4.           * 4.           * 4.
0.             0.             0.
$1 20000       $1 20000       $1 20000
line 0. 100    line 0. 100    line 0. 100
0.             0.             0.
pack 1. 1. 1.
lp.lya
0.
if $f1 <= 0. then $f1
expr 96 + 8.*$f1
makenote 64 95                0
```

Turn gate on for continuous generation of MIDI data.

Calculate the Lyapunov exponent. For anything real time, it's best to stick to the default precision value of one.

Map negative values to MIDI notes, positive values remain silent.

Set MIDI Channel here

Traversing a three-dimensional Lyapunov space with **lp.lya** *to generate MIDI data.*

What’s in a name?

Clipped from the name of the venerable Alexander Mikhailovich Lyapunov.

See Also

lp.poppy	Simple population growth model.
lp.poppy~	Simple population growth model as signal.

Dewdney, A. Kee. “Mathematical recreations: Leaping into Lyapunov Space,” *Scientific American*. 1991 (CCLXV:3) pp. 130-132.

Markus, Mario. “Chaos in maps with continuous and discontinuous maxima,” *Computers in Physics*, 1990 pp. 481-493.

lp.nn~

This object is an extended implementation of the nn factor, provided with many of the Litter Power noise generators. In addition to reducing sample resolution, **lp.nn~** also allows you to add a dither component to the least-significant bits. While we're at it, we've added sample-rate reduction.

Although in many ways similar to the **degrade~** object, there are subtle differences in the processing. These become increasingly noticeable when scaling signals before or after processing. We are rather chaffed about the dither business, not to mention fractional bit-resolution.

Input

signal	In left inlet: signal to degrade.
int float	In middle inlet: nn factor. Positive values degrade bit resolution, with maximum degradation at *nn* = 31 (effectively one bit of signal). Negative values overwrite low-order bits with dithering noise; maximum noise is reached at *nn* = -31. All nn values in the range -1 ≤ *nn* ≤ 1 leave the signal unaltered. Other NN values close to zero generally do not produce much audible degradation, but rest assured that degradation is, indeed, taking place. This will become clear when applied to very low-energy signals to which high gain is subsequently applied. The floating-point NN factor allows for smooth transitions between the integer bit-resolution values. This feature is still somewhat experimental, and implementation details may be modified in the future.
float	In right inlet: effective sample rate in Hz. If set to zero or a value higher than the current sampling rate, no down-sampling takes place. For values between zero and the current sampling rate, samples from the incoming signal are repeated (skipping incoming samples as needed) to emulate a signal sampled at the lower rate.

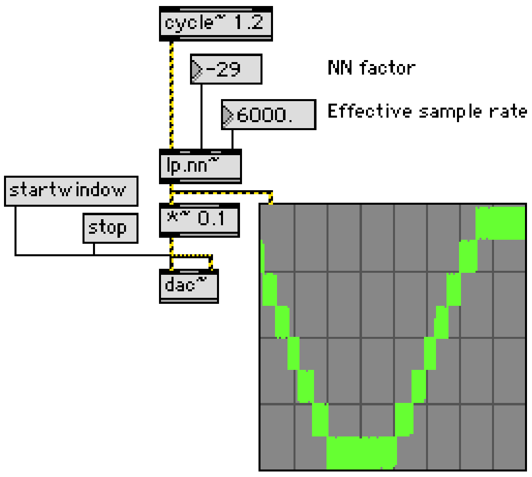
Arguments

int	Optional first argument: sets initial NN factor. Zero by default.
float	Optional second argument: sets initial effective sample rate. Zero by default (i.e., no down-sampling).

Output

signal	Degraded signal.

Examples



Degrading a low-frequency signal.

What's in a name?

A token of esteem for a dear friend.

See Also

degrade~	If you have to ask why, you probably don't need this object.
lp.gsss~	G3nral-purpoz s!gnl degrd8!on: rduz kap!talizt resolut!on | d!ther | m0d!fy faz!st smpl-r8
lp.pfff~	Brown noise
lp.phhh~	Black noise
lp.pvvv~	Colored noise with variable Hurst exponent
lp.shhh~	White noise
lp.sss~	Pink noise (Gardner/Voss algorithm)
lp.trrr~	Triangular (dither) noise
lp.zzz~	Pink noise (McCartney algorithm)

This is a control-domain version of the **lp.phhh~** black noise signal generator. It generates values in the range $0 \le x \le 1$.

Input

bang	Generate a random value from a "black" ($1/f^3$) distribution.
seed	The symbol seed followed by an integer reseeds the internal random number generator. (Only available if the object was initialized with a seed parameter.)
int	In second inlet: sets the NN factor. This is a value in the range $0 \le nn \le 31$ that controls the "granularity" of the random numbers. For a NN factor of zero (the default), all bits of the random numbers are random. For other values, NN indicates the number of low-order bits to mask out before converting to a floating-point value.

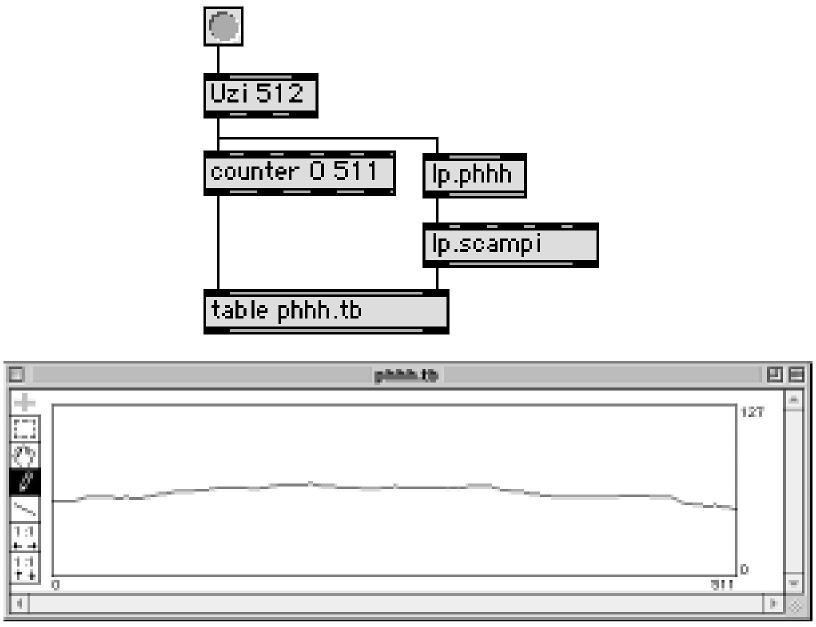
Arguments

inst	Set the value for the seed of the core random number generator. The generator is auto-seeded if this value is zero (the default).

Output

float	A random value in the range $0 \le x \le 1$.

Examples



Generating random numbers with a $1/f^3$ distribution.

lp.phhh

Generate random numbers with a $1/f^3$ (black) distribution

What's in a name?

See **lp.phhh~**

See Also

lp.grrr	"Gray" noise (control domain)
lp.pfff	Generate random numbers from a Brownian distribution
lp.phhh~	"Black" ($1/f^3$) noise
lp.shhh	Generate random numbers from a "white" distribution
lp.sss	Generate random numbers from a 1/f ("pink") distribution
lp.tata	Generate random numbers using the Tausworthe 88 algorithm
lp.titi	Generate random numbers using the TT800 algorithm
lp.zzz	Generate random numbers from a 1/f ("pink") distribution

lp.pi

Pro Bundles only

Input

int	An int in any inlet causes the product of values currently at all inlets to be calculated and sent through the outlet.
float	A float in any inlet causes the product of values currently at all inlets to be calculated and sent through the outlet. If at least one initialization argument was a float, all calculations will be performed in floating point and the result will be sent through the outlet as a float. Otherwise any incoming floats will be truncated to an integer value before the addition is performed
bang	Send last sum calculated through outlet
set	The symbol set followed by a number updates the value associated with the inlet to be updated without sending the resulting sum through the outlet.
list	You can send a list of numbers (floats or ints) to any inlet. The first number will be sent to the receiving inlet, the second number to the following inlet on the right, and so on. After the numbers have been distributed, the new product of all inlets is calculated and sent through the outlet.

Arguments

int
float

The number of initialization arguments determines the number of inlets, their initial values, and the type of calculation (integer of floating point)..

If there are no initialization arguments, **lp.pi** will have two inlets, both initialized to one. If there is one argument, it initializes the value stored at the first inlet. Specify two arguments to initialize the values stored at both inlets. If there are three or more arguments, there will be the same number of inlets as arguments and the inlets will take initial values as specified.

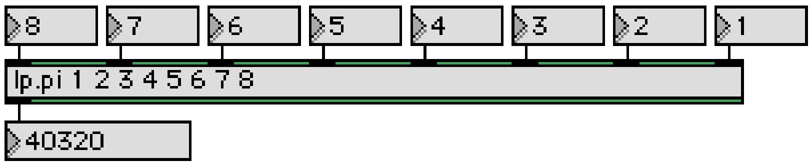
If any of the initialization arguments is a float, then all calculation will be performed in floating point and the result will be sent through the outlet as a float. Otherwise all calculations will be performed with integers and floats arriving at the inlets will be truncated.

Output

int
float

The current sum. The type of value (int or float) depends upon the initialization arguments

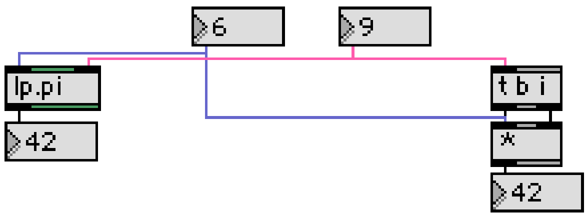
Examples



Lp.pi *can multiply as many numbers as you want.*

lp.pi

Pro Bundles only



I got tired of having to include trigger objects to get the effect I wanted.

What's in a name?

Name of the Greek letter used to represent multiplication.

See Also

*****	Multiply two numbers
lp.delta	Multiply numbers
lp.logos	Calculate quotient and remainder of two numbers
lp.sigma	Add numbers
trigger	Sends its input to many places, in right-to-left order

lp.poppy

Generate sequences of values in the range $0 < x < 1$ using the interative formula:

$$p' = rp(1-p)$$

P is in the range $0 \le p \le 1$, r can take on values in the range $0 \le r \le 4$.

This iterative formula is often used to model population growth under the assumption that there is some theoretical maximum population, limited by availability of food or other constraints which are assumed to be constant. Under this model, p represents the current population as a fraction of the maximum population and r represents the growth rate in each period of growth (for instance, a year). The model assumes that, as the population becomes large, it will die off at faster rates due to scarcity of food (or to other constraints). This is reflected in the multiplication of the current population, p, with (1-p).

If the growth rate is less than two, the population will eventually die off. Growth rates between two and three lead to constant population levels. Starting at growth rates of three, the population will alternate between two different levels. The two levels get further and further apart as the growth rate increases. When the growth rate exceeds 3.43, the population pattern bifurcates again to alternate between four different levels. With increasing growth rates the population behavior becomes increasingly complex. Once the growth rate reaches 3.569946 the population behavior becomes chaotic.

Input

bang	Generate next population level and send it out the outlet.
float	In the left inlet sets the growth rate, generates the next population value, and sends the result out the outlet. Input values are clipped to the range $0 \le r \le 4$ In the right inlet sets the current population level.Input values are clipped to the range $0 \le p \le 1$.
list	I think this is cool. You can send a list of floats to the left inlet. Then lp.poppy will cycle through the values of the list, using them as changing growth rates on each bang.Like sending a single float, the next population level is calculated and send out the outlet.
set	The symbol set followed by one or more floats sets the population growth pattern without generating the next population level. Nothing is sent out the outlet.
reset	Set the population level back to the initial population (or the last value specified in the right outlet).

Arguments

float Optional. As many float values as you want.

If only one float is given, it specifies the initial growth rate.

If two floats are given, the first one specifies the initial growth rate and the second one specifies the initial population level.

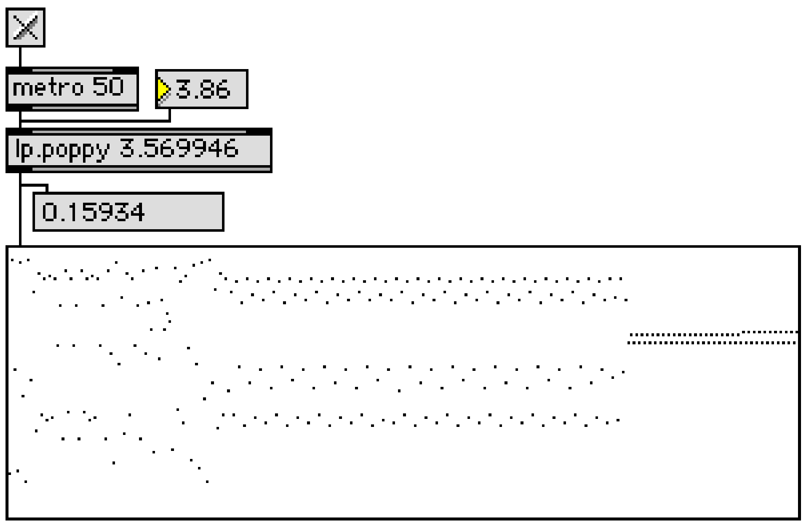
If more floats are given, the last one specifies the initial population level and the rest define a cycle of growth rates.

If no values are given, lp.poppy defaults to an initial population levelof 2/3 and a growth rate of 3.569946.

Output

float The next population level.

Examples



Population rate of 3 in the righthand portion of the **multiSlider**, *3.569946 in the middle, and 3.86 at the left.*

What's in a name?

In honor of Madam Pomfrey, who does her best to maintain the population level of her charges.

See Also

lp.lya	Calculate Lyapunov exponent for population growth models.
lp.poppy~	Simple population growth model as signal.

Dewdney, A Kee. "Computer recreations: Probing the strange attractions of chaos," Scientific American 1987 (CCLXXXVII:1) pp. 90-93

Glieck, James. *Chaos: The Making of a New Science.* Penguin, New York, 1987.

Li, Tien-Yien and James Yorke. "Period three implies chaos" American Mathematical Monthly, 1975 (LXXXII) pp. 985-992.

lp.poppy~

Simple population growth model as signal

All Bundles

Generate audio signals using the chaotic population growth iteration formula.

Think of this object as a cross between **lp.poppy** and **lp.frrr~**.

Input

float	In the left inlet sets the growth rate. Input values are clipped to the range $0 \le r \le 4$
	In the middle inlet sets the current population level.Input values are clipped to the range $0 \le p \le 1$.
	In the right inlet sets the base frequency. Note that the actual frequency used may be adjusted by **lp.poppy~** to match an integral sub-harmonic of the sampling rate.
signal	In the left inlet: the growth rate can be controlled by a signal.
int	In the right inlet: zero, one, two, or minus one. Zero indicates no interpolation between generated values, one indicates linear interpolation, a value of two indicates quadratic interpolation, and minus one indicates geometric interpolation. Any other negative values are treated as minus one; values larger than two are clipped.
reset	Sets the population level back to the initial population (or the last value specified in the middle outlet. Useful if the signal has degenerated, which can happen in somewhat unusual circumstances.

Arguments

Up to three floats followed by an int. All arguments are optional. You must specify the first argument if you want to specify the second; similarly for the third and fourth arguments.

float	The first argument specifies the initial growth rate. The default value is 3.56994571869. (Yes, the default value is double-precision, the internal arithmetic is performed with double-precision throughout.).
	The second argument specifies the initial ("seed") population. The default value is the smallest positive number that can be represented as a single-precision floating point value. (Why single-precision here? You'll be bored if I go into the details.).
	The third argument specifies an (approximate) initial setting in Hz for the base frequency at which new population values are generated. The default value is the sampling rate (ie, a new population value is generated every sample).
int	The fourth argument specifies an initial value for interpolation, which should be either zero, one, or two. The default value is zero (no interpolation).

Output

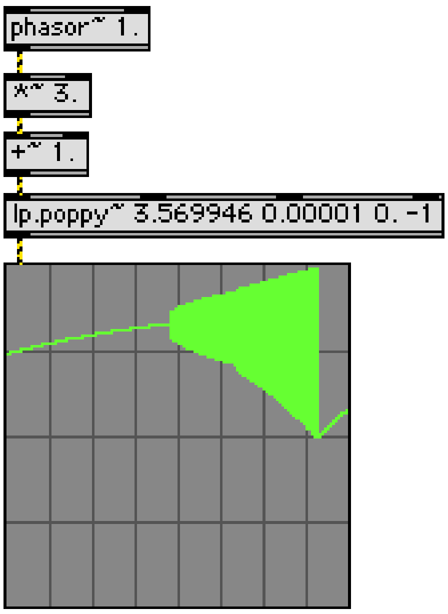
signal	The population level.

Simple population growth model as signal

lp.poppy~

Pro Bundles only

Examples



```
phasor~ 1.
*~ 3.
+~ 1.
lp.poppy~ 3.569946 0.00001 0. -1
```

Signal as population growth rate goes from one to four.

What's in a name?

See **lp.poppy**.

See Also

lp.frrr~	Low frequency noise generator.
lp.lya	Calculate Lyapunov exponent for population growth models.
lp.poppy	Simple population growth model.

lp.pvvv

Colored noise with variable Hurst exponent

Pro Bundles

This is a control-domain version of the **lp.pvvv~** variable-color noise signal generator. It generates values in the range $0 \le x \le 1$.

The “color” (or “persistance”) of the random numbers generated is controlled by a *Hurst Exponent*. A Hurst Exponentof 0 generates a 1/f distribution (pink noise), 0.5 generates a $1/f^2$ distribution (brown noise), and 1 generates a $1/f^3$ distribution (black noise). You can experiment with values in between and beyond…

Input

bang	Generate a random value from a “colored” distribution.
float	In left inlet: set the Hurst Exponent. This is normally a value in the range $0 \le h \le 1$, but values outside the range can be handled by **lp.pvvv**. Values larger than one produce random numbers with very little local variation **NB:** In the current implementation, negative values for the Hurst Exponent sometimes produce results that do not match the mathematical model, due to arithmetic overflow in intermediate calculations. This issue will be dealt with in a future version of **lp.pvvv**. Please be aware that the characteristics of the random numbers generated for negative Hurst Exponents may change in future..
int	In second inlet: sets the NN factor. This is a value in the range $0 \le nn \le 31$ that controls the "granularity" of the random numbers. For a NN factor of zero (the default), all bits of the random numbers are random. For other values, NN indicates the number of low-order bits to mask out before converting to a floating-point value.
seed	The symbol seed followed by an integer reseeds the internal random number generator. (Only available if the object was initialized with a seed parameter.)

Arguments

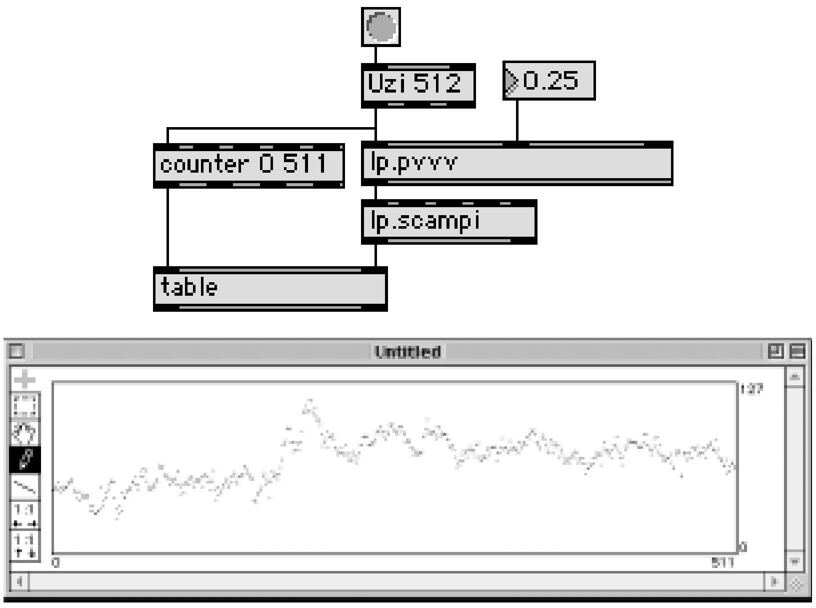
You can initialize an **lp.pvvv** object with up to four optional arguments. You must specify the first argument if you want to specify the second argument , and so on. The arguments, in order, are:

int	Length of cycle, which must be a power of two. **Lp.pvvv** precalculates this number of random numbers in advance, and the "color" of the distribution applies, strictly speaking, only to the distribution over one cycle. So, longer cycles give a more "accurate" long-term distribution. However, longer cycles require more memory and also mean a greater latency between the time the Hurst Exponent is changed and when The default cycle is 512; this has proven to be a good practical value for most applications. If you specify zero or a negative number for this argument, the default will be used. Any arguments that are not powers of two are rounded down to the next power of two. The maximum value for this argument is limited only by available memory.
float	Set initial value of Hurst Exponent. The default is zero (i.e., pink noise)
int	Set an initial value for the NN factor. The default is zero.
int	Set the value for the seed of the core random number generator. The generator is auto-seeded if this value is zero (the default).

Output

float	A random value in the range $0 \le x \le 1$.

Examples



Generating random numbers with a distribution half way between pink and brown noise.

lp.pvvv

What’s in a name?

See **lp.pvvv~**

See Also

lp.scampi	Scale, offset, and limit numbers; output integers
lp.grrr	"Gray" noise (control domain)
lp.pfff	Generate random numbers from a 1/f^2 (“Brownian”) distribution
lp.pvvv~	Colored noise with variable Hurst exponent
lp.scampi	Scale, offset, and limit numbers; output integers
lp.shhh	Generate random numbers from a “white” distribution
lp.sss	Generate random numbers from a 1/f (“pink”) distribution (Voss/Gardner algorithm)
lp.zzz	Generate random numbers from a 1/f (“pink”) distribution (McCartney algorithm)
lp.tata	Generate random numbers using the Tausworthe 88 algorithm

The “color” (or “persistance”) of the noice produced is controlled by a *Hurst Exponent*. A Hurst Exponentof 0 generates a 1/f distribution (pink noise), 0.5 generates a $1/f^2$ distribution (brown noise), and 1 generates a $1/f^3$ distribution (black noise). You can experiment with values in between and beyond…

Input

float	In left inlet: set the Hurst Exponent. This is normally a value in the range $0 \le h \le 1$, but values outside the range can be handled by **lp.pvvv~**. Values larger than one produce very low-frequency noise; negative values produce high energy signals.
int	In right inlet: sets the NN factor. This is a value in the range $0 \le nn \le 31$ that controls the "granularity" of the random numbers. For a NN factor of zero (the default), all bits of the random numbers are random. For other values, NN indicates the number of low-order bits to mask out before converting to a floating-point value.

NB: In the current implementation, negative values for the Hurst Exponent sometimes produce results that do not match the mathematical model, due to arithmetic overflow in intermediate calculations. This issue will be dealt with in a future version of **lp.pvvv~**. Please be aware that the characteristics of the noise signal for negative Hurst Exponents may change in future.

Arguments

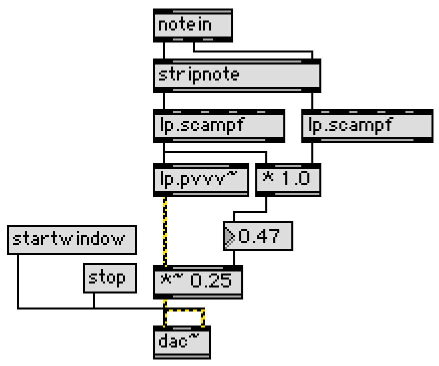
You can initialize an **lp.pvvv~** object with up to two optional arguments. You must specify the first argument if you want to specify the second.

float	Set initial value of Hurst Exponent. The default is zero (i.e., pink noise)
int	Set an initial value for the NN factor. The default is zero.

Output

signal	Colored noise.

Examples



Playing noise colors from a MIDI keyboard.

lp.pvvv~

What’s in a name?

Onomatopoeia.

See Also

lp.grrr	"Gray" noise (control domain)
llp.pfff~	"Brownian" (1/f^2) noise
p.phhh~	Black (1/f^3) noise
lp.shhh~	White noise
lp.sss~	Pink (1/f) noise (Voss/Gardner algorithm)
lp.zzz~	Pink (1/f) noise (McCartney algorithm)

lp.sigma

Input

int	An int in any inlet causes the sum of values currently at all inlets to be calculated and sent through the outlet.
float	A float in any inlet causes the sum of values currently at all inlets to be calculated and sent through the outlet. If at least one initialization argument was a float, all calculations will be performed in floating point and the result will be sent through the outlet as a float. Otherwise any incoming floats will be truncated to an integer value before the addition is performed
bang	Send last sum calculated through outlet
set	The symbol set followed by a number updates the value associated with the inlet to be updated without sending the resulting sum through the outlet.
list	You can send a list of numbers (floats or ints) to any inlet. The first number will be sent to the receiving inlet, the second number to the following inlet on the right, and so on. After the numbers have been distributed, the new sum of all inlets is calculated and sent through the outlet.

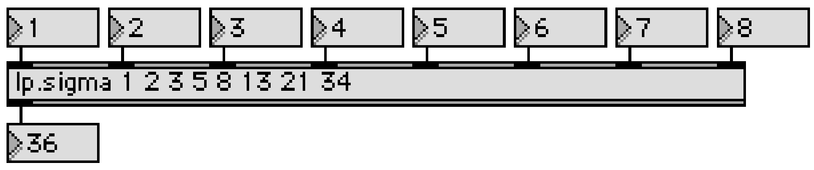
Arguments

int float	The number of initialization arguments determines the number of inlets, their initial values, and the type of calculation (integer of floating point).. If there are no initialization arguments, lp.sigma will have two inlets, both initialized to zero. If there is one argument, it initializes the value stored at the first inlet. Specify two arguments to initialize the values stored at both inlets. If there are three or more arguments, there will be the same number of inlets as arguments and the inlets will take initial values as specified. If any of the initialization arguments is a float, then all calculation will be performed in floating point and the result will be sent through the outlet as a float. Otherwise all calculations will be performed with integers and floats arriving at the inlets will be truncated.

Output

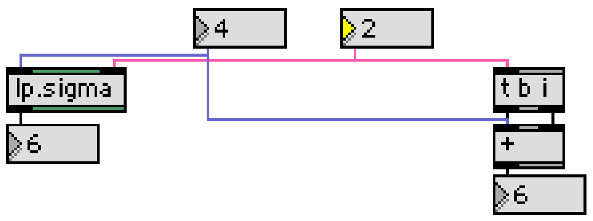
int float	The current sum. The type of value (int or float) depends upon the initialization arguments

Examples



Lp.sigma *can add as many numbers as you want.*

lp.sigma



I got tired of having to include trigger objects to get the effect I wanted.

What's in a name?

Name of the Greek letter used to represent summation.

See Also

+	Add two numbers
lp.delta	Calculate differences between two numbers
lp.logos	Calculate quotient and remainder of two numbers
lp.pi	Multiply numbers
lp.stacey	Collect statistics
trigger	Sends its input to many places, in right-to-left order

Noise with a triangular distribution is frequently used for dithering.

Input

int Sets the NN factor, specifying the number of low-order bits to clear before converting the integer representation to floating-point. The NN factor may be in the range 0 ≤ *nn* ≤ 31.

Since **lp.trrr~** is typically used to counteract effects that resemble the NN factor, it may seem a little odd to NN-ize this object, but why not?

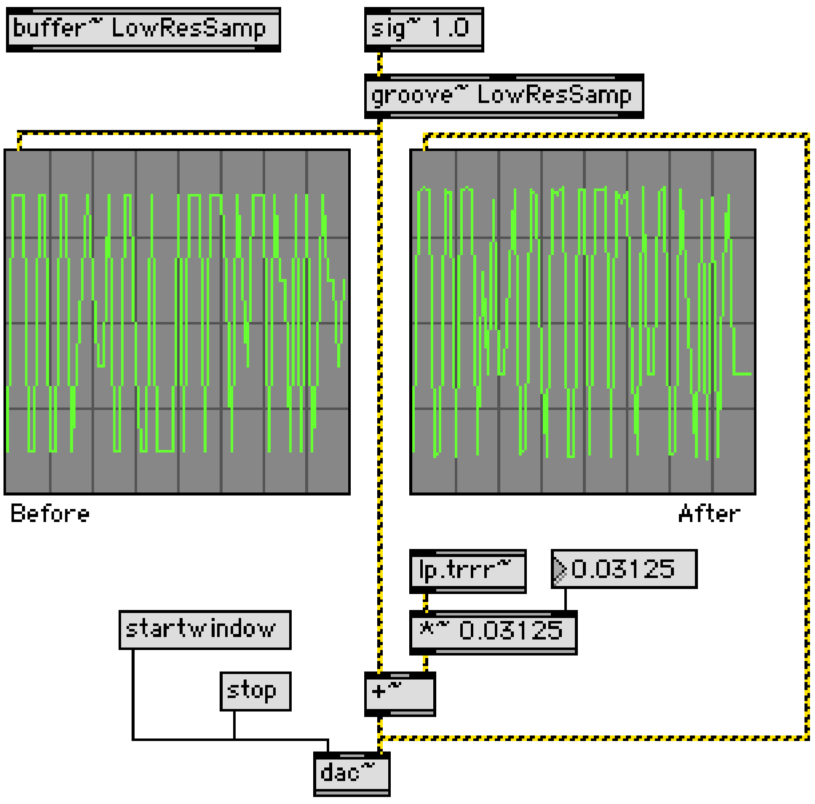
Arguments

int Optional value sets an initial NN factor

Output

signal Dithering noise.

Examples



Adding dither can take the edge off signals originally sampled with low resolution.

lp.trrr~

Triangular (dither) noise

All Bundles

What's in a name?

Trrriangular noise.

See Also

lp.gsss~	Gaussian noise.
lp.nn~	G3nral-purpoz s!gnl degrd8!on: rduz kap!talizt resolut!on | d!ther | m0d!fy faz!st smpl-r8