Litter Power Package: List of Objects &c.

Discrete Distributions

Object	Distributions	Output Range	Options	Parameters (defaults, ranges)	Availability
lp.bernie	Bernoulli Binary Choice (n = 1)	0 .. n		n [1]: Positive integer, number of Bernoulli trials p [0.5]: Probability (i.e between 0 and 1) of positive outcome	*Starter*
lp.bibi	Beta Binomial model	0 .. N		n [1]: Positive integer, number of Bernoulli trials alpha, beta [0.5, 0.5]: Parameters of an underlying binomial distribution generating values of p for a Bernoulli (n, p) distributions	*Pro*
lp.dicey	Dice	0 .. n * m		n [2]: Positive integer, number of dice m [6]: Faces on each die Optional list of integers representing "weights" of each face on the die. All faces for which no individual weighting has been specified are equiprobable	*Pro*
lp.ernie	Arbitrary distributions using the "finite urn" model			L [128]: Length of cycle Other messages: const int, refer symbol (refers to table with data)	*Pro*
lp.ginger	I Ching Produces both “present” and “future” oracles. Supports yarrow stick and coin toss methods.	1 .. 64	coin (default) yarrow	None	*Starter*
lp.lili	Uniform distribution over long integers. Well, sort of uniform. Based on the now largely obsolete Linear Congruence method for generating pseudo-random numbers. Allows you to play with the parameters. Find the shortest cycle!	0 .. mod. Note that lili is working in the unsigned domain; unsigned values ≥ 2^31are interpreted by Max as negative values.		mul [65539]: Seed multiplier between succesive calls add [0]: Constant added to (seed * mul) at each succesive call mod [4294967296]: Modulo base used. 0 is interpreted as 2^32. seed [1]: Initial seed for linear congruence pseudo-random number generator The default values give the same random numbers as Max.	*Pro*
lp.mama	Uniform distribution over long integers. Based on the Generate random numbers using the Marsaglia "Mother of All Random Number Generators" algorithm algorithm: flat distribution, cycle of approx. 2^250, all bits random, robust seeding.	min .. max (See parameters. Note that the range is defined inclusively)		min [-2,147,483,648]: integer less than max max [2,147,483,647]: integer greater than min If only one integer is given as an initialization argument and it is positive, the range is set from 0 to the specified value. If this integer is negative, the maximum is taken as 0.	*Pro*
lp.mrmr	Uniform distribution over long integers. Based on the Mersenne Twister algorithm: flat distribution, unbelievable cycle of over 2^19000, all bits random, robust seeding.	min .. max (See parameters. Note that the range is defined inclusively)		min [-2,147,483,648]: integer less than max max [2,147,483,647]: integer greater than min If only one integer is given as an initialization argument and it is positive, the range is set from 0 to the specified value. If this integer is negative, the maximum is taken as 0.	*Pro*
lp.pfishie	Poisson	0 .. ∞		lambda [1]: Positive real, is both mean and standard deviation	*Pro*
lp.tata	Uniform distribution over long integers. Based on the Taus88 (Tausworthe) algorithm: flat distribution, cycle of just under 2^88, all bits random, robust seeding. All that, and faster than the traditional linear congruence method!	min .. max (See parameters. Note that the range is defined inclusively)		min [-2,147,483,648]: integer less than max max [2,147,483,647]: integer greater than min If only one integer is given as an initialization argument and it is positive, the range is set from 0 to the specified value. If this integer is negative, the maximum is taken as 0.	*Starter*
lp.titi	Uniform distribution over long integers. Based on M. Matsumoto's TT800 algorithm: flat distribution, cycle of 2^800 - 1, all bits random, robust seeding.	min .. max (See parameters. Note that the range is defined inclusively)		min [-2,147,483,648]: integer less than max max [2,147,483,647]: integer greater than min If only one integer is given as an initialization argument and it is positive, the range is set from 0 to the specified value. If this integer is negative, the maximum is taken as 0.	*Pro*
lp.zippie	Zipf distribution	1 .. ∞		Zeta exponent [1.0]. Controls steepness of the distribution curve.	*Pro*
	NB: Output from any of these distributions can be scaled/mapped to non-standard values with lp.scampi				

Continuous Distributions

Object	Distributions	Output Range	Options	Parameters (defaults, ranges)	Availability
lp.abbie	arc sine (a = 0.5 and b = 0.5) beta	0 .. 1		a [0.5]: Positive real; increase tendency towards 0 b [0.5]: Positive real; increase tendency towards 1	*Pro*
lp.chichi	Chi Square	0 .. ∞		f [1]: Positive integer, degrees of freedom	*Pro*

lp.coshy	Cauchy Positive Cauchy Negative Cauchy	-∞ .. ∞ 0 .. ∞ -∞ .. 0	sym (default) pos neg	tau [1]: Positive real, effectively a scaling factor	*Pro*
lp.expo	Exponential Bilateral (“[First Law of] Laplace”) Negative Exponential	0 .. ∞ -∞ .. ∞ -∞ .. 0	pos (default) sym neg	lambda [1]: Positive real, influences mean and standard deviation	*Pro*
lp.fishie	Fisher	0 .. ∞		f1 and f2 [both 1]: Positive integers, two independent degrees of freedom parameters	*Pro*
lp.gammer	Gamma/Erlang distribution	0 .. ∞		nu [1]: Positive real. If nu is an integral value, the Erlang distribution is produced. lambda [1]: Positive real	*Pro*
lp.grrr	“Gray” noise	0 .. 1		None	*Pro*
lp.hyppie	Hyperbolic Cosine & Hyperbolic Secant distributions	-∞ .. ∞	cos sec	None	*Pro*
lp.linnie	Linear (decreasing) Triangular Linear (increasing)	0 .. 1	neg (default) sym pos	None Note: the option names refer to the slope of the distribution. Does that help?	*Starter*
lp.loggie	Logistic	-∞ .. ∞		alpha [1]: Positive real beta [0]: Non-negative real	*Pro*
lp.lonnie	Log-normal	(0) .. ∞		mu*[1]: Positive real sigma* [1]: Positive real	*Pro*
lp.norm	Normalized Gauss distribution (with mean at 0 and standard deviation of 1)	-∞ .. ∞		mu[0]: any real value sigma [1]: Positive real (0 is allowed but counter-productive; negative values give the same results as the complementary positive value)	*Starter*
lp.pfff	Brownian motion (“brown” noise)	0 .. 1			*Starter*
lp.phhh	1/f^3 random distribution (“black” noise)	0 .. 1			*Starter*
lp.pvvv	Variable Hurst exponent for 1/f^h distributions (“white“, “pink”, “brown”, “black” and between and beyond)	0 .. 1		Hurst exponent [0.0]: Controls “color” of random number distribution	*Pro*
lp.shhh	Uniform (“white” noise)	0 .. 1		nn [0]: Integer from 0 to 31; a noisiness factor (number of least significant bits masked out before calculating result)	*Starter*
lp.sss	1/f (“pink” noise: Voss/Gardner algorithm)	0 .. 1		nn [0]: Integer from 0 to 29 a noisiness factor (number of least significant bits masked out before calculating result).	*Starter*
lp.stu	Student’s “T” Distribution	-∞ .. ∞		f [1]: Positive integer, degrees of freedom	*Pro*
lp.y	Weibull/Rayleigh distributions	0 .. ∞		s [1]: Positive real t [1]: Positive real	*Pro*
lp.vilfrie	Pareto distribution	b .. ∞		a [1]: Shape parameter, controls sharpness with which the probability of higher values falls off b [1]: Location parameters, determines smallest possible value	
lp.zzz	1/f (McCartney's improved algorithm for pink noise)	0 .. 1		nn [0]: Integer from 0 to 29 a noisiness factor (number of least significant bits masked out before calculating result).	*Pro*
	NB: Output from any of these distributions can be scaled/mapped to other values (including integers) with lp.scampi				

Signal Generators

Object	Distributions	Output Range	Options	Parameters (defaults, ranges)	
lp.epoisse~	Cheesy noise: part pitch, part noise	signal (-1 .. 1)		frequency pitch factor Hurst exponent	*Pro*
lp.frrr~	Low frequency noise	signal (-1 .. 1)		baseFreq [1000]: Approx. frequency of center freq. (always rounded to integral division of sample rate) Interp [0]: Order of interpolation between generated random values. Range from 0 to 2.	*Pro*
lp.feta~	1-bit white noise	signal (-1 .. 1)		amp [sqrt(1/3)]. Produces power equal to lp.shhh	*Starter*
lp.grrr~	Gray noise	signal (-1 .. 1)			*Pro*
lp.ksks~	Plucked-string noise (without the intonation problems of Karplus-Strong)	signal (-1 .. 1)		frequency	*Pro*
lp.lll~	Linear Congruence Noise	signal (-1 .. 1)		mul add: mod[0]: 0 is interpreted as 2^32. seed [1]: Initial seed for linear congruence pseudo-random number generator	*Pro*
lp.pfff~	Brown noise	signal (-1 .. 1)		nn [0]: Granularity of noise (bit-masking)	*Starter*
lp.phhh~	Black noise	signal (-1 .. 1)		nn [0]: Granularity of noise (bit-masking)	*Pro*

lp.pvvv~	Variable colored noise	signal (-1 .. 1)	Mode: 0 (native) 1 (PPC) 2 (Intel) 3 (PC415)	Hurst exponent [0.0]: Controls 'darkness' of noise. 0 generates a pink noise, 1 generates a black noise. Fractional values produce signals in between, negative values are increasingly 'white' (lower than -0.5 remains white), larger values produce even 'darker' noise. nn [0]: Granularity of noise (bit-masking) in range [0 .. 31]	*Pro*
lp.qvvv~	Variable colored noise using floating-point calculations	signal	stet clip wrap reflect	Hurst exponent [0.0]: Controls 'darkness' of noise. 0 generates a pink noise, 1 generates a black noise. Fractional values produce signals in between, negative values are increasingly 'white' (lower than -0.5 remains white), larger values produce even 'darker' noise. nn [0]: Granularity of noise (bit-masking) [0.0 .. 31.0]. Use float input to specify fractional bit-depth	*Pro*
		Note on options: lp.qvvv~ can produce output outside the nominal signal range of (-1 .. 1). Use clip, wrap, or reflect mode to limit output to the standard signal range.			
lp.ppp~	Dust/popcorn noise	signal (-1 .. 1)	pos (default) sym neg	density [100]: average number of "pops" per second. popWidth [1]: number of samples before and after pop peak.	*Pro*
lp.shhh~	White noise	signal (-1 .. 1)		nn [0]: Granularity of noise (bit-masking)	*Starter*
lp.sss~	Pink noise (Original Voss algorithm)	signal (-1 .. 1)		nn [0]: Granularity of noise (bit-masking)	*Starter*
lp.trrr~	Triangular noise (and Linear?)	signal (-1 .. 1)	**sym (default)** **pos** **neg**	nn [0]: Granularity of noise (bit-masking)	*Starter*
lp.zzz~	Pink noise (McCartney's algorithm)	signal (-1 .. 1)		nn [0]: Granularity of noise (bit-masking)	*Pro*

Mutation and Cross-Synthesis

Object	Distributions	Output Range	Options	Parameters (defaults, ranges)	
lp.emeric~	Cross-synthesis for first years	signal		omega [0]: value in the unit range, proximity to source/target	*Starter*
lp.frim~	Frequency-domain interval mutation	signal	usim, uuim, isim, iuim, lcm, wcm; rel/abs	omega [0]: value in the unit range, proximity to source/target pi [0]: value in the unit range, "clumping factor" (isim/iuim/lcm only) delta [0]: in range [-1 .. 1], "delta emphasis" (relative mutations only)	*Pro*
lp.tim~	Time-domain interval mutation	signal	usim, uuim, isim, iuim, lcm, wcm; rel/abs	omega [0]: value in the unit range, proximity to source/target pi [0]: value in the unit range, "clumping factor" (isim/iuim/lcm only) delta [0]: in range [-1 .. 1], "delta emphasis" (relative mutations only)	*Starter*
lp.vim	Interval mutation for values (ints and floats)	int/float	usim, uuim, isim, iuim, lcm, wcm; rel/abs	omega [0]: value in the unit range, proximity to source/target pi [0]: value in the unit range, "clumping factor" (isim/iuim/lcm only) delta [0]: in range [-1 .. 1], "delta emphasis" (relative mutations only)	*Pro*

Litter Chaos

Object	**Function**	**Output Range**	**Options**	**Parameters, defaults, ranges**	
lp.ccc	Schuster/Procaccia 1/f generator	[0 .. 1]			*Pro*
lp.ccc~	Schuster/Procaccia 1/f noise	[0 .. 1]			*Pro*
lp.poppy	Population growth model	[0 .. 1]		seed growth rate/rates	*Starter*
lp.poppy~	Population growth noise generator	[0 .. 1]		baseFreq [1000]: Approx. frequency of center freq. (always rounded to integral division of sample rate) interp [0]: Order of interpolation between generated random values. Range from 0 to 2. growth rate/rates intial population	*Starter*
lp.lya	Lyapunov spaces	-∞ .. ∞	iterations	growth rate[s]	*Pro*

Utilities	Description			Parameters (defaults, ranges)	
lp.ale	Plug-and-play replacement for the orphaned alea object. Uses a more efficient scrambling algorithm and Litter Power-strength random number generation			An initial list can be specified with arbitrary instantiation arguments.	*Starter*
lp.c2p~	Cartersian to polar coordinate conversion (compatible with MSP1 and MSP2)			None	*Starter*
lp.crabelms	Plug-and-play replacement for the orphaned scramble object. Uses a more efficient scrambling algorithm and Litter Power-strength random number generation			An initial list can be specified with arbitrary instantiation arguments.	*Starter*
lp.grl~	Phase unwrapping			FFT Size	*Pro*
lp.i	Posts fortunes from I Ching to the Max window. Can be used with the ginger object.			None	*Pro*
lp.kg	Maps I Ching results (from the ginger object) to some number (2 to 63) of different items. Bang generates a new distribution pattern.			n [2]: Integer between 2 and 63, number of different choices to make.	*Pro*
lp.nn~	General-purpose quality degradation (NN) function	signal		Resolution degradation [0]: default means no change; positive values in 1 ≤ nn ≤ 31 indicate number of low-order bits to mask; negative values in -1 ≤ nn ≤ -31 indicate number of bits to replace with dithering noise. Fractional values allow a continuous sweep of the bit-depth spectrum. Sample-rate degradation [0 == current SR]: any value in the range 0.0 < esr < SR sets an effective sample rate.	*Pro*
lp.p2c~	Polar to Cartesian coordinate conversion (compatible with MSP1 and MSP2)			None	*Starter*
lp.scampf	Scale, map, and limit floating point values Scampf can limit output to a specific range. Specify one of the limiter options (clip, reflect, wrap, or split) or send it as a message. The nolim message overrides range limits and all values are sent out the left outlet after being scaled and offset. When the split option is in effect, out of range values are sent out the right outlet. In the case of clip, reflet, and wrap, out of range values are "corrected" into range (by clipping, reflection, or wrapping) and sent out the left outlet; the right outlet sends out a 0 if no range correction was necessary, non-zero if correction was necessary (the value sent is actually a count of the number of consecutive values requiring correction).		clip reflect wrap split stet	scale [128]: Any value; input 1 maps to offset + scale. (default value chosen for convenient mapping of MIDI data) offset [0]: Any value that 0 input maps to. limit [stet]: any of the limiting options clip, reflect, wrap, split, or stet. min [0]: Minimum output value (ignored if limit option is set to stet). max [1]: Maximum output value (ignored if limit option is set to stet).	*Starter*
lp.scampi	Scale, map, and limit values Scampi can limit output to a specific range. Specify one of the limiter options (clip, reflect, wrap, or split) or send it as a message. The nolim message overrides range limits and all values are sent out the left outlet after being scaled and offset. When the split option is in effect, out of range values are sent out the right outlet. In the case of clip, reflet, and wrap, out of range values are "corrected" into range (by clipping, reflection, or wrapping) and sent out the left outlet; the right outlet sends out a 0 if no range correction was necessary, non-zero if correction was necessary (the value sent is actually a count of the number of consecutive values requiring correction).		clip reflect wrap split stet trunc round floor ceil	scale [1/128]: Any value; input 1 maps to offset + scale. (default value chosen for convenient mapping of MIDI data) offset [0]: Any value that 0 input maps to. limit [stet]: any of the limiting options clip, reflect, wrap, split, or stet. min [0]: Minimum output value (ignored if limit option is set to stet). max [1]: Maximum output value (ignored if limit option is set to stet). integer-conversion [trunc]: Integer processing truncates by default, but scampi understands floor, ceil, round, and trunc messages to specify direction.	*Starter*
lp.scamp~	Scale, map, and limit signals Scampi~ can limit output to a specific range. Specify one of the limiter options (clip, reflect, wrap, or split) or send it as a message. The nolim message overrides range limits and all values are sent out the left outlet after being scaled and offset. When any of the clip, reflet, and wrap, options are specified, out of range values are "corrected" into range (by clipping, reflection, or wrapping). When scampi~ receives a bang message, a value is sent out the right outlet indicating how many samples required range correction (since object creation or the last bang message). The value sent is actually a count of the number of values requiring correction.		clip reflect wrap stet	scale [-6dB]: Any value; input 1 maps to offset + scale. offset [0]: Any value that 0 input maps to. limit [stet]: any of the limiting options clip, reflect, wrap or stet. (Note that lp.scampi~ does not support split.) min [-1]: Minimum output value (ignored if limit option is set to stet). max [1]: Maximum output value (ignored if limit option is set to stet).	*Pro*
lp.stacey	Basic statistics: Count, Min/Max, Mean, Standard Deviation, Skew, and Kurtosis			n[0]: Window size of data buffered; when this is exceeded, correlation is calculated only for the last n data (reminder: implement as ring buffer, and implement a "remove" command to incrementally remove data). 0 indicates no window	*Starter*
lp.simga **lp.delta** **lp.pi** **lp.logos**	"Active" arithmetic: data in any inlet triggers output. Also: - Plus/times sprout additional inlets according to number of arguments - Minus has three outlets (a-b, b-a, and abs) - Div has three outlets (a/b, a remainder b, a mod b)	-∞ .. ∞	«none»	Two or more optional arguments to specify initial values for each inlet (default: two inlets, each initialized to zero). Type of first argument (int/float) determines type of output.	*Pro* *(lp.sigma is Starter)*