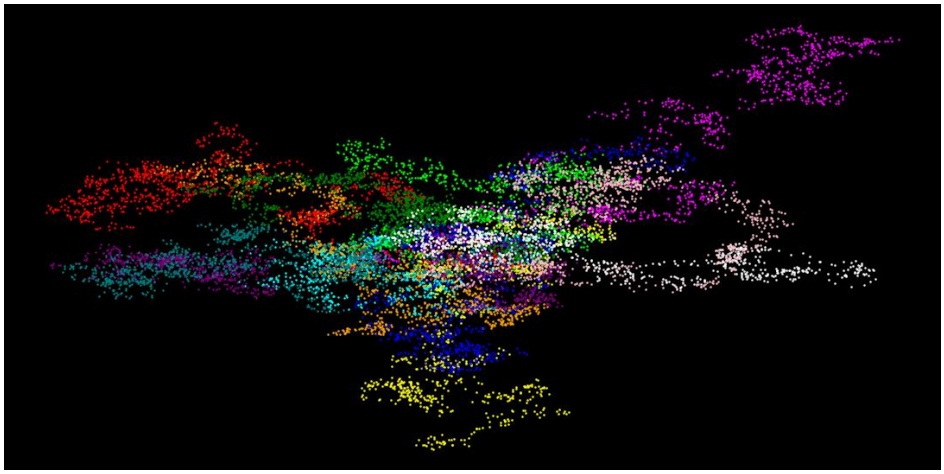


A Stochastic Toolkit for Max

Carl Faia

MaxAlea Documentation

Version 3.0

Plotting problems of points

July 23, 2024

MaxAlea Documentation

Carl Faia

22.07.24

Contents

Introduction

This document provides comprehensive documentation for the **MaxAlea** library, a collection of objects for working with various probability distributions and stochastic processes in the Max environment. **MaxAlea** is a comprehensive collection of stochastic tools and algorithms for the Max environment. Originally inspired by Mikhail Malt's PatchWork library from the early 1990s, **MaxAlea** has evolved into a dynamic, real-time toolkit for experimental and creative work with probabilistic processes.

Key Features

- **Real-time manipulation of stochastic processes**
- **Diverse collection of random distributions and algorithms**
- **Implementations of random walks and 1/f noise**
- **Utility objects designed specifically for stochastic operations**
- **Consistent, high-quality random number generation**

History and Development

Initiated in 1996, **MaxAlea** has undergone several iterations to keep pace with evolving Macintosh architecture. The latest version, completely rewritten using the Min-DevKit in `C++`, ensures compatibility with modern Max environments on both Intel and Apple Silicon Macs. While primarily developed for macOS, the source code is available on GitHub, facilitating potential Windows ports by the community.

Unique Approach to Randomness

MaxAlea employs a seeded random number generator based on the Mersenne Twister algorithm, ensuring high-quality, non-repeating sequences of random numbers. This approach differs from the default random functions in Max and other environments, offering more robust and diverse random behaviors.

Using the Library

1. Place the externals in your Max Packages folder.
2. Access help files directly from Max by right-clicking on objects.
3. Explore the included example patches for practical applications.

Components

- **Random Distributions** (e.g., Gaussian, Poisson, Weibull)
- **Random Walk and 1/f Noise Algorithms**
- **Markov Chain Analysis and Generation**
- **Data Mapping and Scaling Utilities**
- **Bounded Random Value Generators**

Applications

- Algorithmic Composition
- Generative Art and Sound Design
- Interactive Installations
- Data Sonification
- Stochastic Modeling in Various Fields

Future Development

Max.alea is an ongoing project. Planned additions include more stochastic models, enhanced utilities, and comprehensive documentation. Community feedback and contributions are welcome.

Acknowledgments

Special thanks to Mikhail Malt for the original inspiration and guidance, and to Richard Dudas for his invaluable assistance in refining the original code.

Contact

For questions, comments, or contributions, please contact: Carl Faia - cf@carlfaia.com or carl.faia@brunel.ac.uk

Bibliography

- Ames, C. (1991). A catalog of statistical distributions: The techniques for transforming random, determinate and chaotic sequences. *Leonardo Music Journal*, 1(1), 55-70.
- Baffioni, C., Guerra, F., & Laura, T. L. (1989). The theory of stochastic processes and dynamical systems as a basis for models of musical structures. In *ATTI del Convegno di Modena, 1989*.
- Bolognesi, T. (1979). Composizione automatica: Dalla musica 1/f alla musica autosimile. In *ATTI del 3° Colloquio di Informatica Musicale, Univ. di Padova, 2-3 aprile 1979*.
- Di Scipio, A. (1990). Composition by exploration of non-linear dynamic systems. In *ICMC Proceedings 1990*.
- Dodge, C., & Jerse, T. (1985). *Computer Music*. Schirmer Books.
- Lorrain, D. (1980). Une panoplie de canons stochastiques. Rapport IRCAM no. 30, Paris. (Also available in English in *The Music Machine*, edited by Curtis Roads.)
- Siddall, J., & Siddall, J. C. (1977). New developments in stochastic computer music. In *ICMC 1977*.
- Voss, R. F., & Clarke, J. (1978). 1/f noise in music: Music from 1/f noise. *Journal of the Acoustical Society of America*, 63(1), 258-261.
- Xenakis, I. (1981). *Musiques Formelles*. Stock Musique.

Additional Reading

- Biles, J. A. (2013). *GenJam: A genetic algorithm for generating jazz solos*. In R. Kronland-Martinet, S. Ystad, & K. Jensen (Eds.), *Computer Music Modeling and Retrieval* (pp. 2-27). Springer.
- Hoffmann, P. (2019). *The stochastic synthesis of Iannis Xenakis*. In S. P. Souvlaki (Ed.), *Xenakis Matters* (pp. 45-59). MIT Press.
- Lazzarini, V. (2016). *A sound design perspective on stochastic synthesis*. In *Sound and Music Computing Conference 2016* (pp. 10-17).
- Manzo, V. J. (2016). *Max/MSP/Jitter for music: A practical guide to developing interactive music systems for education and more*. Oxford University Press.
- Pearson, M. (2011). *Generative art: A practical guide using Processing*. Manning Publications.
- Roads, C. (2004). *Microsound*. MIT Press.
- DuBois, R. L. (2010). *The sound of numbers: Aural models of data*. In *Data Sonification: Theory, Methods, and Applications* (pp. 133-147). CRC Press.

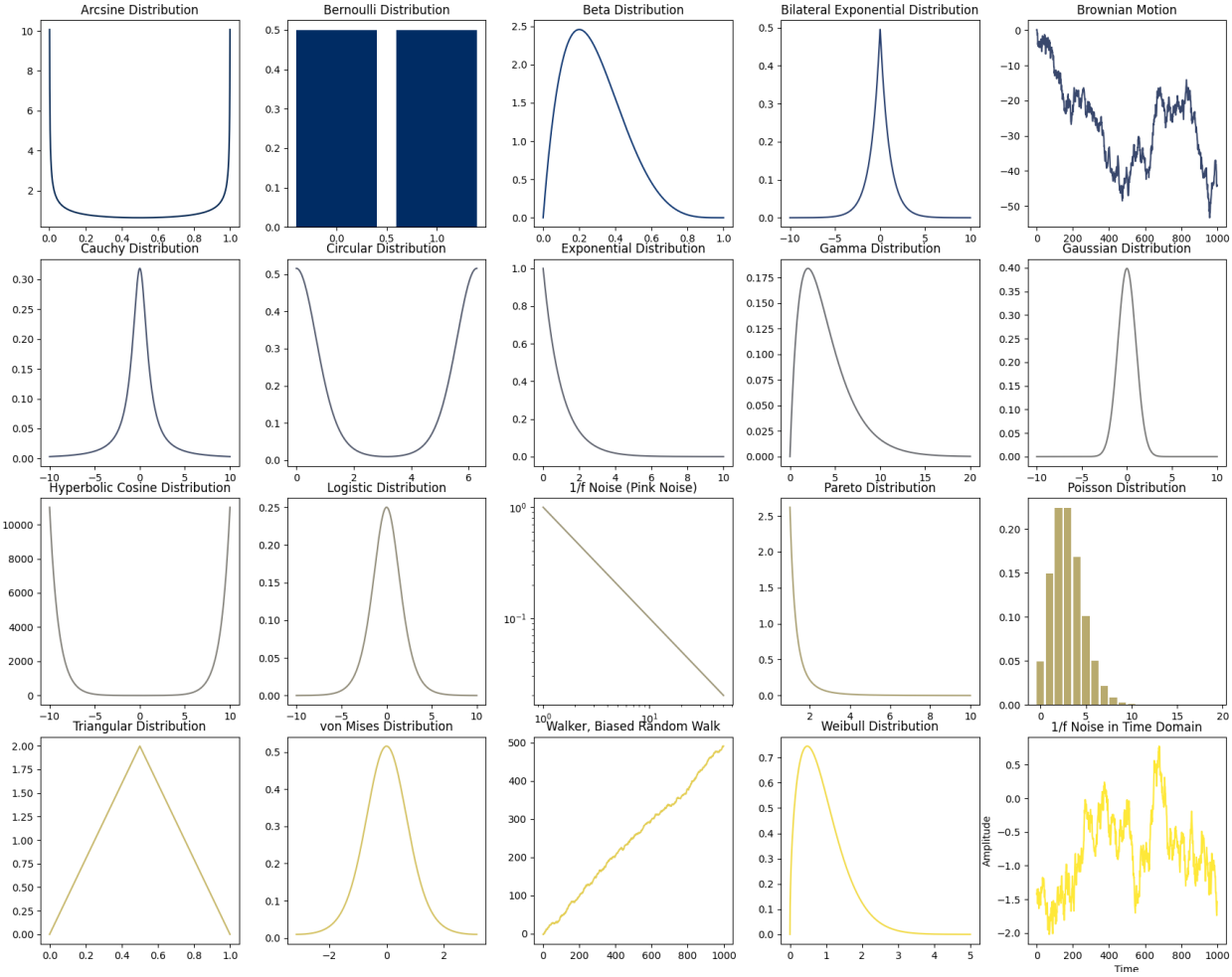
Modern Web Resources

- Shiffman, D. (2012). *The nature of code*. Nature of Code. Retrieved from https://natureofcode.com/
- Taube, H. (2013). *Algorithmic composition: A gentle introduction to music composition using Common LISP and Common Music*. Michigan Publishing. Retrieved from https://quod.lib.umich.edu/s/spobooks/bbv9810.0001.001

Online Resources

- GitHub Repository: https://github.com/carlfiaia/MaxAlea
- Max Package Manager: [-]
- Documentation: [-]

Note: This library is provided as-is, under a MIT License with Attribution. Users are encouraged to explore, modify, and expand upon these tools for their creative and research endeavors.


Arcsine Distribution
Bernoulli Distribution
Beta Distribution
Bilateral Exponential Distribution
Brownian Motion
Cauchy Distribution
Circular Distribution
Exponential Distribution
Gamma Distribution
Gaussian Distribution
Hyperbolic Cosine Distribution
Logistic Distribution
1/f Noise (Pink Noise)
Pareto Distribution
Poisson Distribution
Triangular Distribution
von Mises Distribution
Walker, Biased Random Walk
Weibull Distribution
1/f Noise in Time Domain
Amplitude
Time

1 alea.arcsin

1.1 Distribution: Arcsine Distribution

1.1.1 Brief History

The arcsine distribution is derived from the arcsine function and was first studied in depth by Ronald Fisher in the 1930s. It's a special case of the beta distribution.

1.1.2 General Uses

- Finance: Modeling time spent by a stock price above a certain level
- Physics: Describing the distribution of time a Brownian motion spends on one side of the origin
- Statistics: Used in some statistical tests and random walk problems
- Game Theory: Analyzing certain types of games and betting strategies

1.1.3 How the Object Works in Max

Arguments:

- alpha (optional, default 1.0): Scales the distribution
- beta (optional, default 0.0): Shifts the distribution

Inlets:

- Main inlet: Trigger generation (bang, float, or int)
- Alpha inlet: Set alpha value
- Beta inlet: Set beta value

Outlet:

- Random number output (float or int)

Additional Features:

- mode message: Output as float (0) or int (1)
- seed setting: Set the random seed for reproducibility
- Input mode: Choose between internal or external random number generation
- info message: Print current state information

1.1.4 What the Object Outputs

The object generates random numbers using the arcsine distribution, which can be scaled and shifted using the alpha and beta parameters.

1.1.5 How to Use It in Max

1. Create an `[alea.arcsin]` object
2. Send it a bang to generate a new random number
3. Optionally set alpha and beta values through the respective inlets
4. Receive the output as either a float or int, depending on the mode setting

1.1.6 Key Points

- Scales and shifts the distribution using alpha and beta parameters
- Can output both float and integer values
- Reproducibility through seed setting

1.1.7 Creative Uses

- Generating random durations for musical notes where longer notes are less frequent
- Creating variable time intervals in interactive installations

1.1.8 Original Math Formula

The arcsine distribution can be represented as:

$$f(x; \alpha, \beta) = \frac{1}{\pi \sqrt{x(1-x)}}$$

2 alea.bernoulli

2.1 Distribution: Bernoulli Distribution

2.1.1 Brief History

The Bernoulli distribution is named after Swiss mathematician Jacob Bernoulli and was described in his work "Ars Conjectandi," published posthumously in 1713. It models a binary outcome experiment.

2.1.2 General Uses

- Statistics: Modeling yes/no outcomes or success/failure events
- Quality Control: Modeling defective/non-defective items in manufacturing
- Machine Learning: In logistic regression and neural networks
- Epidemiology: Modeling the occurrence of a disease in a population

2.1.3 How the Object Works in Max

Arguments:

- p (optional, default 0.5): Probability of success

Inlets:

- Main inlet: Trigger generation (bang)
- p inlet: Set probability parameter (float)

Outlet:

- Random number output (int, either 0 or 1)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information

2.1.4 What the Object Outputs

The object generates random binary outcomes (0 or 1) based on the probability p.

2.1.5 How to Use It in Max

1. Create an `[alea.bernoulli]` object
2. Send it a bang to generate a new random number
3. Optionally set the probability value through the second inlet
4. Receive the output as an integer (0 or 1)

2.1.6 Key Points

- Models binary outcomes with a specified probability of success
- Can set the probability dynamically
- Ensures reproducibility through seed setting

2.1.7 Creative Uses

- Simulating coin flips or other binary decision processes
- Generating random sequences for algorithmic music compositions

2.1.8 Original Math Formula

The Bernoulli distribution can be represented as:

$$P(X = 1) = p$$

$$P(X = 0) = 1 - p$$

3 alea.beta

3.1 Distribution: Beta Distribution

3.1.1 Brief History

The beta distribution was first studied by Euler and Legendre in the 18th century. It's a versatile distribution defined on the interval [0, 1] and is often used in Bayesian statistics.

3.1.2 General Uses

- Statistics: Modeling probabilities and proportions
- Project Management: Estimating task completion times (PERT technique)
- Reliability Analysis: Modeling failure rates and system reliability
- Bayesian Inference: As a conjugate prior for binomial and Bernoulli distributions

3.1.3 How the Object Works in Max

Arguments:

- a (optional, default 0.5): First shape parameter
- b (optional, default 0.5): Second shape parameter

Inlets:

- Main inlet: Trigger generation (bang)
- a inlet: Set a parameter (float)
- b inlet: Set b parameter (float)

Outlet:

- Random number output (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information

3.1.4 What the Object Outputs

The object generates random numbers using the beta distribution, which is shaped by the a and b parameters.

3.1.5 How to Use It in Max

1. Create an `[alea.beta]` object
2. Send it a bang to generate a new random number
3. Optionally set the a and b values through the respective inlets
4. Receive the output as a float

3.1.6 Key Points

- Shape parameters a and b control the form of the distribution
- Useful in various statistical and Bayesian applications
- Ensures reproducibility through seed setting

3.1.7 Creative Uses

- Modeling probabilities in interactive systems
- Generating random values for game mechanics or simulations

3.1.8 Original Math Formula

The beta distribution can be represented as:

$$f(x; a, b) = \frac{x^{a-1}(1-x)^{b-1}}{B(a, b)}$$

where $B(a, b)$ is the beta function.

4 alea.brownie

4.1 Distribution: Brownian Motion (with boundaries)

4.1.1 Brief History

Brownian motion, also known as the Wiener process, was discovered by Robert Brown in 1827 while studying pollen grains in water. It was later mathematically modeled by Albert Einstein in 1905, and rigorously constructed as a mathematical object by Norbert Wiener in 1923.

4.1.2 General Uses

- Physics: Modeling particle motion in fluids
- Finance: Modeling stock prices and other financial instruments
- Biology: Describing the movement of molecules in cells
- Signal Processing: Analyzing and generating noise
- Music and Sound Design: Creating natural-sounding random variations in pitch or timbre

4.1.3 How the Object Works in Max

Arguments:

- start (optional): Initial value (default 51.75)
- low (optional): Lower boundary (default 25.75)
- high (optional): Upper boundary (default 115.5)
- bandwidth (optional): Step size parameter (default 2.1)
- seed (optional): Random seed

Inlets:

- Main inlet: Trigger generation (bang)
- Start inlet: Set start value (float/int)
- Low inlet: Set low boundary (float/int)
- High inlet: Set high boundary (float/int)
- Bandwidth inlet: Set bandwidth (float)

Outlet:

- Brownian motion value (float/int)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information
- Boundary reflection: When the generated value exceeds the high or low boundary, it's reflected back into the valid range

4.1.4 What the Object Outputs

The object generates random numbers simulating Brownian motion within specified boundaries. It uses a normal distribution to generate steps, which are then added to the current value. The bandwidth parameter controls the standard deviation of these steps. If the new value exceeds the boundaries, it's reflected back into the valid range.

4.1.5 How to Use It in Max

1. Create an `[alea.brownie]` object with optional arguments
2. Send a bang to the main inlet to generate a new Brownian motion value
3. Optionally set start, low, high, and bandwidth values through the respective inlets
4. Receive the output as either a float or int, depending on the input type to the start inlet

4.1.6 Key Points

- Models random walk with reflection at boundaries
- Generates smooth variations over time
- Reproducibility through seed setting

4.1.7 Creative Uses

- Generating smooth random variations in musical parameters
- Modeling realistic price movements in financial simulations

4.1.8 Original Math Formula

Brownian motion can be described by the stochastic differential equation:

$$dX_t = \mu dt + \sigma dW_t$$

where μ is the drift coefficient, σ is the diffusion coefficient, and W_t is a Wiener process.

5 alea.cauchy

5.1 Distribution: Cauchy Distribution

5.1.1 Brief History

The Cauchy distribution, also known as the Lorentz distribution, is named after Augustin-Louis Cauchy. It was first studied by Cauchy in 1853, although similar distributions were studied earlier by Pierre-Simon Laplace and Simeon Denis Poisson. The distribution gained prominence in physics as the Lorentz distribution, named after Hendrik Lorentz, who used it to model spectral lines.

5.1.2 General Uses

- Physics: Modeling resonance behavior and spectral line shapes
- Statistics: As a pathological example in statistical theory due to its undefined moments
- Engineering: Describing the distribution of mechanical stress in solid materials
- Finance: Modeling financial data with heavy tails
- Signal Processing: Analyzing signals with sharp peaks

5.1.3 How the Object Works in Max

Arguments:

- alpha (optional): Scale parameter (default 1.0)
- seed (optional): Random seed

Inlets:

- Main inlet: Trigger generation (bang)
- Alpha inlet: Set alpha value (float/int)

Outlet:

- Cauchy-distributed random number (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including alpha, seed, last generated value, and count of generations

5.1.4 What the Object Outputs

The object generates random numbers following the Cauchy distribution with a specified scale parameter (alpha). The Cauchy distribution is known for its heavy tails and undefined mean and variance.

5.1.5 How to Use It in Max

1. Create an `[alea.cauchy]` object with an optional alpha argument
2. Send a bang to the main inlet to generate a new Cauchy-distributed random number
3. Optionally set the alpha value through the second inlet
4. Receive the output as a float

5.1.6 Key Points

- The alpha parameter controls the width of the distribution. Smaller values result in a more peaked distribution, while larger values create a more spread-out distribution.
- The Cauchy distribution can produce extreme values more frequently than a normal distribution, which can be useful for modeling certain phenomena but may also require careful handling in some applications.
- Unlike many other distributions, the Cauchy distribution does not have defined moments (mean, variance, etc.), which makes it unique and sometimes challenging to work with in statistical analyses.

5.1.7 Creative Uses

- Modeling phenomena with frequent extreme values
- Generating random effects in audio or visual applications

5.1.8 Original Math Formula

The probability density function (PDF) of the Cauchy distribution is:

$$f(x; \alpha, x_0) = \frac{1}{\pi\alpha \left[1 + \left(\frac{x - x_0}{\alpha}\right)^2\right]}$$

where α is the scale parameter and x_0 is the location parameter.

6 alea.cauchypos

6.1 Distribution: Positive Cauchy Distribution

6.1.1 Brief History

The positive Cauchy distribution is a variation of the standard Cauchy distribution, restricted to positive values. It's derived from the work of Augustin-Louis Cauchy in the 19th century, but modified to only produce positive outcomes.

6.1.2 General Uses

- Physics: Modeling one-sided resonance phenomena
- Engineering: Analyzing lifetime data with heavy right tails
- Finance: Modeling positive financial returns with extreme values
- Reliability Analysis: Describing failure times in certain scenarios
- Optics: Modeling intensity distributions in certain optical systems

6.1.3 How the Object Works in Max

Arguments:

- alpha (optional): Scale parameter (default 1.0)
- seed (optional): Random seed

Inlets:

- Main inlet: Trigger generation (bang)
- Alpha inlet: Set alpha value (float/int)

Outlet:

- Positive Cauchy-distributed random number (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including alpha, seed, last generated value, and count of generations

6.1.4 What the Object Outputs

The object generates random numbers following the positive Cauchy distribution with a specified scale parameter (alpha). Unlike the standard Cauchy distribution, this version only produces positive values.

6.1.5 How to Use It in Max

1. Create an `[alea.cauchypos]` object with an optional alpha argument
2. Send a bang to the main inlet to generate a new positive Cauchy-distributed random number
3. Optionally set the alpha value through the second inlet
4. Receive the output as a float

6.1.6 Key Points

- The alpha parameter controls the scale of the distribution. Larger values of alpha result in a more spread-out distribution.
- This distribution always produces positive values, making it useful for scenarios where negative values don't make sense (e.g., lifetimes, distances).
- Like the standard Cauchy distribution, the positive Cauchy distribution has heavy tails, meaning it can produce extreme values more frequently than many other distributions.
- The distribution is created by using only the positive half of a standard Cauchy distribution, effectively "folding" the negative half onto the positive side.

6.1.7 Creative Uses

- Modeling lifetime data or failure times
- Generating random positive values for simulations or generative art

6.1.8 Original Math Formula

The probability density function (PDF) of the positive Cauchy distribution is:

$$f(x; \alpha, x_0) = \frac{1}{\pi\alpha \left[1 + \left(\frac{x-x_0}{\alpha}\right)^2\right]}, \quad x > 0$$

where α is the scale parameter and x_0 is the location parameter.

7 alea.choice

7.1 Distribution: Weighted Binary Choice

7.1.1 Brief History

While not a traditional statistical distribution, this object implements a weighted random choice between two options, which is a fundamental concept in probability theory and decision making. The idea of weighted random selection has applications in various fields and can be traced back to early probability theory developed by mathematicians like Blaise Pascal and Pierre de Fermat in the 17th century.

7.1.2 General Uses

- Decision Making: Simulating choices based on probabilities
- Game Development: Implementing random events with different likelihoods
- Artificial Intelligence: Basic decision making in AI agents
- Simulations: Modeling scenarios with binary outcomes
- Music and Art: Creating generative content with controlled randomness

7.1.3 How the Object Works in Max

Arguments:

- weight (optional): Initial weight value (default 0.5)
- choice_A (optional): Initial value for choice A
- choice_B (optional): Initial value for choice B

Inlets:

- Main inlet: Trigger generation (bang)
- Weight inlet: Set weight value (float, 0.0 to 1.0)
- Choice A inlet: Set value for choice A (anything)
- Choice B inlet: Set value for choice B (anything)

Outlet:

- Chosen value (anything)

Additional Features:

- Weight attribute: Can be set and queried as an object attribute
- Flexible input: Choices can be numbers, symbols, or lists

7.1.4 What the Object Outputs

While this object uses the same math formula as the Bernoulli object, it allows the choices to be anything (ints, floats, lists, symbols, strings). The object generates a random choice between two options (A and B) based on a weight value. The weight determines the probability of choosing option A.

7.1.5 How to Use It in Max

1. Create an `[alea.choice]` object with optional arguments for weight and initial choices
2. Set choices A and B using the respective inlets (can be numbers, symbols, or lists)
3. Set the weight value if desired (default is 0.5, meaning equal probability)
4. Send a bang to the main inlet to trigger a random choice
5. Receive the chosen value from the outlet

7.1.6 Key Points

- The weight value ranges from 0.0 to 1.0, where 0.0 always selects choice B, 1.0 always selects choice A, and 0.5 gives equal probability to both choices.
- Choices can be single values (numbers or symbols) or lists, allowing for flexible use in various contexts.
- The object uses a uniform random distribution to make the weighted choice, ensuring fair selection based on the specified weight.

7.1.7 Creative Uses

- Creating controlled random decisions in interactive installations
- Generating random binary outcomes in algorithmic compositions

7.1.8 Original Math Formula

The weighted binary choice can be represented as:

$$P(A) = w$$

$$P(B) = 1 - w$$

where w is the weight for choice A.

8 alea.circ

8.1 Distribution: Circular Distribution

8.1.1 Brief History

Circular distributions are used to model periodic phenomena or directional data. They are a broad class of probability distributions defined on the unit circle. These distributions describe random variables whose values are angles or directions. While this specific implementation is a custom variation, circular statistics have been studied since the early 20th century, with significant contributions from Ronald Fisher in the 1950s.

8.1.2 General Uses

- Directional Statistics: Modeling wind directions, compass bearings, or other circular data
- Time-based Cyclical Events: Modeling events that repeat on a daily or yearly basis
- Biology: Analyzing animal migration patterns or circadian rhythms
- Music: Generating cyclical patterns or variations in pitch or rhythm
- Physics: Describing phase angles in wave phenomena

8.1.3 How the Object Works in Max

Arguments:

- mean (optional): Initial mean value (default 0.0)
- arc (optional): Initial arc value (default 1.0)

Inlets:

- Main inlet: Trigger generation (bang)
- Mean inlet: Set mean value (float)
- Arc inlet: Set arc value (float)

Outlet:

- Generated random value (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including mean, arc, and seed values

8.1.4 What the Object Outputs

The object generates random values based on a custom circular distribution. The distribution is controlled by two parameters: mean and arc.

8.1.5 How to Use It in Max

1. Create an `[alea.circ]` object with optional arguments for mean and arc
2. Optionally set the mean and arc values through the respective inlets
3. Send a bang to the main inlet to generate a new random value
4. Receive the output as a float

8.1.6 Key Points

- The mean parameter centers the distribution around a specific value
- The arc parameter controls the spread of the distribution
- The generated values are wrapped around the circle, creating a periodic distribution
- The implementation uses a uniform distribution transformed to create the circular distribution

8.1.7 Creative Uses

- Generating cyclical variations in musical compositions
- Modeling directional data in simulations or visual art

8.1.8 Original Math Formula

The probability density function (PDF) of a custom circular distribution can be complex, but typically involves transformations of the uniform distribution with trigonometric functions.

9 alea.exp

9.1 Distribution: Exponential Distribution

9.1.1 Brief History

The exponential distribution was first introduced by Agner Krarup Erlang in 1909 while studying telephone call patterns. It has since become a fundamental distribution in probability theory and statistics, particularly in modeling time-to-event data.

9.1.2 General Uses

- Reliability Engineering: Modeling the time between failures in systems
- Queueing Theory: Describing inter-arrival times in service systems
- Physics: Modeling radioactive decay processes
- Biology: Analyzing survival times in populations
- Finance: Modeling the time between financial events, such as stock price changes

9.1.3 How the Object Works in Max

Arguments:

- lambda (optional): Rate parameter (default 1.0)

Inlets:

- Main inlet: Trigger generation (bang)
- Lambda inlet: Set lambda value (float)

Outlet:

- Exponentially distributed random number (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including seed, lambda, and last generated value

9.1.4 What the Object Outputs

The object generates random numbers following the exponential distribution with a specified rate parameter (lambda).

9.1.5 How to Use It in Max

1. Create an `[alea.exp]` object with an optional lambda argument
2. Optionally set the lambda value through the second inlet
3. Send a bang to the main inlet to generate a new exponentially distributed random number
4. Receive the output as a float

9.1.6 Key Points

- The lambda parameter is the rate parameter of the exponential distribution. It's the inverse of the mean of the distribution.
- Larger values of lambda result in a distribution more concentrated near zero, while smaller values create a more spread-out distribution.
- The exponential distribution always generates non-negative values.
- If a lambda value of 0 is provided, the object will output an error message and set lambda to the default value of 1.0.

9.1.7 Creative Uses

- Generating random waiting times for events in interactive installations
- Modeling time intervals in simulations

9.1.8 Original Math Formula

The probability density function (PDF) of the exponential distribution is:

$$f(x; \lambda) = \lambda e^{-\lambda x}$$

for $x \geq 0$, where λ is the rate parameter.

10 alea.gamma

10.1 Distribution: Gamma Distribution

10.1.1 Brief History

The gamma distribution was first introduced by Leonard Euler in the 18th century. It gained prominence in the early 20th century through the work of statisticians like Karl Pearson and Ronald Fisher. The gamma distribution generalizes several other distributions and has connections to the chi-squared and exponential distributions.

10.1.2 General Uses

- Reliability Analysis: Modeling lifetimes of devices or systems
- Meteorology: Analyzing rainfall patterns and other weather phenomena
- Finance: Modeling income distributions and insurance claims
- Biology: Studying species abundance and other ecological data
- Physics: Analyzing particle interactions and other physical processes

10.1.3 How the Object Works in Max

Arguments:

- nu (optional): Shape parameter (default 1.0)
- lambda (optional): Rate parameter (default 1.0)

Inlets:

- Main inlet: Trigger generation (bang)
- Nu inlet: Set nu value (float)
- Lambda inlet: Set lambda value (float)

Outlet:

- Gamma-distributed random number (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including seed, nu, lambda, and last generated value

10.1.4 What the Object Outputs

The object generates random numbers following the gamma distribution with specified shape (nu) and rate (lambda) parameters.

10.1.5 How to Use It in Max

1. Create an `[alea.gamma]` object with optional nu and lambda arguments
2. Optionally set the nu and lambda values through the respective inlets
3. Send a bang to the main inlet to generate a new gamma-distributed random number
4. Receive the output as a float

10.1.6 Key Points

- The nu parameter (shape) controls the basic shape of the distribution. When nu = 1, the gamma distribution reduces to an exponential distribution.
- The lambda parameter (rate) is the inverse of the scale parameter. It stretches or compresses the distribution along the x-axis.
- Both nu and lambda must be positive. The object will output an error message if non-positive values are provided.
- The gamma distribution is always non-negative, making it suitable for modeling quantities that cannot be negative (like lifetimes or amounts).

10.1.7 Creative Uses

- Generating random lifetimes for objects in simulations
- Modeling rainfall or other natural phenomena in generative art

10.1.8 Original Math Formula

The probability density function (PDF) of the gamma distribution is:

$$f(x; \nu, \lambda) = \frac{\lambda^\nu x^{\nu-1} e^{-\lambda x}}{\Gamma(\nu)}$$

for $x > 0$, where ν is the shape parameter and λ is the rate parameter.

11 alea.gauss

11.1 Distribution: Gaussian (Normal) Distribution

11.1.1 Brief History

The Gaussian or Normal distribution was introduced by Abraham de Moivre in 1733 and further developed by Carl Friedrich Gauss in the early 19th century. It has become one of the most widely used probability distributions in statistics and natural sciences due to its mathematical properties and its applicability to many real-world phenomena.

11.1.2 General Uses

- Statistics: Modeling measurement errors and random variability
- Physics: Describing particle behavior in quantum mechanics
- Finance: Modeling stock price changes and risk assessment
- Biology: Analyzing biological measurements and population distributions
- Psychology: Studying distribution of IQ scores and other cognitive measures

11.1.3 How the Object Works in Max

Arguments:

- sigma (optional): Standard deviation (default 0.7)
- mu (optional): Mean (default 0.9)

Inlets:

- Main inlet: Trigger generation (bang)
- Sigma inlet: Set sigma value (float)
- Mu inlet: Set mu value (float)

Outlet:

- Gaussian-distributed random number (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including seed, sigma, mu, and last generated value

11.1.4 What the Object Outputs

The object generates random numbers following the Gaussian (normal) distribution with specified standard deviation (sigma) and mean (mu) parameters.

11.1.5 How to Use It in Max

1. Create an `[alea.gauss]` object with optional sigma and mu arguments
2. Optionally set the sigma and mu values through the respective inlets
3. Send a bang to the main inlet to generate a new Gaussian-distributed random number
4. Receive the output as a float

11.1.6 Key Points

- The sigma parameter (standard deviation) controls the spread of the distribution. Larger values of sigma result in a wider, flatter distribution.
- The mu parameter (mean) determines the center of the distribution.
- Sigma must be positive. The object will output an error message and set sigma to the default value (0.7) if a non-positive value is provided.
- The Gaussian distribution is symmetric around its mean and can take on any real value, making it suitable for modeling phenomena that can be positive or negative.
- This object uses the Mersenne Twister algorithm (`std::mt19937`) for random number generation, ensuring high-quality pseudo-random numbers.

11.1.7 Creative Uses

- Generating random values for realistic simulation of measurements
- Creating natural variations in generative music or visual art

11.1.8 Original Math Formula

The probability density function (PDF) of the Gaussian distribution is:

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ is the mean and σ is the standard deviation.

12 alea.hypercos

12.1 Distribution: Hyperbolic Cosine Distribution

12.1.1 Brief History

The hyperbolic cosine distribution is derived from the hyperbolic cosine function, which was first introduced by mathematicians in the 18th century. It has connections to both the logistic distribution and the hyperbolic secant distribution. While less commonly used than some other distributions, it has found applications in various fields due to its unique properties.

12.1.2 General Uses

- Physics: Modeling certain types of particle behavior
- Signal Processing: Analyzing specific waveforms
- Finance: Describing certain types of financial data
- Engineering: Modeling specific types of mechanical stress
- Biology: Analyzing certain biological processes with symmetric, heavy-tailed distributions

12.1.3 How the Object Works in Max

Arguments:

- a (optional): Scale parameter (default 0.5)
- b (optional): Location parameter (default 0.5)

Inlets:

- Main inlet: Trigger generation (bang)
- 'a' inlet: Set 'a' value (float)
- 'b' inlet: Set 'b' value (float)

Outlet:

- Hyperbolic cosine distributed random number (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including seed, 'a', and 'b' values

12.1.4 What the Object Outputs

The object generates random numbers following a distribution based on the hyperbolic cosine function with specified scale ('a') and location ('b') parameters.

12.1.5 How to Use It in Max

1. Create an `[alea.hypercos]` object with optional 'a' and 'b' arguments
2. Optionally set the 'a' and 'b' values through the respective inlets
3. Send a bang to the main inlet to generate a new random number
4. Receive the output as a float

12.1.6 Key Points

- The 'a' parameter scales the distribution, affecting its spread
- The 'b' parameter shifts the distribution along the x-axis
- The distribution is symmetric around its location parameter 'b'
- This distribution has heavier tails than a normal distribution, making it useful for modeling data with more extreme values
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation

12.1.7 Creative Uses

- Sound design: Use the heavy-tailed nature of the distribution to create interesting textures or granular effects with occasional extreme values
- Generative music: Apply the distribution to pitch selection for melodies with mostly close intervals but occasional large jumps
- Visual art: Map the output to parameters like size or position in generative visuals to create compositions with subtle variations and occasional dramatic elements
- Interactive installations: Use the distribution to control timing of events, creating a mostly regular but occasionally surprising rhythm of interactions
- Data sonification: Apply this distribution when sonifying data sets that have symmetric but heavy-tailed characteristics, providing an auditory representation of the data's nature

12.1.8 Original Math Formula

The probability density function (PDF) of the hyperbolic cosine distribution is based on the hyperbolic cosine function:

$$f(x; a, b) = \frac{\exp(-|x - b|/a)}{2a}$$

where a is the scale parameter and b is the location parameter.

13 alea.interval

13.1 Distribution: Uniform Distribution within a Specified Interval

13.1.1 Brief History

The concept of uniform distribution dates back to the early development of probability theory in the 17th and 18th centuries. It's one of the simplest continuous probability distributions and has been widely used in various fields due to its straightforward interpretation and implementation.

13.1.2 General Uses

- Simulations: Modeling random events with equal likelihood within a range
- Game Design: Generating fair random outcomes
- Cryptography: Creating random keys and initialization vectors
- Statistical Sampling: Selecting random samples from a population
- Computer Graphics: Generating random positions or colors

13.1.3 How the Object Works in Max

Arguments:

- interval_a (optional): Start value (default 60)
- interval_b (optional): Interval value (default 2)

Inlets:

- Main inlet: Trigger generation (bang)
- Start inlet: Set start value (float/int)
- Interval inlet: Set interval value (float/int)

Outlet:

- Random value within the specified interval (float or int)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including seed, start value, and interval value
- Type switching: Automatically switches between float and int output based on the type of the start value

13.1.4 What the Object Outputs

The object generates random numbers uniformly distributed within the interval [a, a+b], where 'a' is the start value and 'b' is the interval value.

13.1.5 How to Use It in Max

1. Create an `[alea.interval]` object with optional start and interval arguments
2. Optionally set the start and interval values through the respective inlets
3. Send a bang to the main inlet to generate a new random number within the specified interval
4. Receive the output as a float or int, depending on the type of the start value

13.1.6 Key Points

- The start value (a) sets the lower bound of the interval
- The interval value (b) determines the range of the distribution
- The object can output either float or int values, automatically determined by the type of the start value
- When outputting integers, the object uses custom rounding to ensure fair distribution
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation

13.1.7 Creative Uses

- Musical composition: Generate random pitches within a specific scale or range
- Rhythmic variation: Create random timing offsets within a defined window for more organic rhythms
- Generative art: Produce random positions, sizes, or colors within constrained ranges for visual elements
- Algorithmic decision making: Use the output to make weighted choices in interactive systems
- Sound design: Generate random parameters for synthesis or effects within musically useful ranges
- Probabilistic sequencing: Create sequences with controlled randomness for evolving patterns
- Stochastic processes: Model simple random walks or other stochastic processes within bounds

13.1.8 Original Math Formula

The uniform distribution within a specified interval [a, b] can be represented as:

$$f(x) = \frac{1}{b - a}$$

for $a \leq x \leq b$.

14 alea.laplace

14.1 Distribution: Laplace Distribution

14.1.1 Brief History

The Laplace distribution, also known as the double exponential distribution, is a continuous probability distribution that describes data that is sharply peaked at the mean and has heavier tails than the normal distribution. It was introduced by Pierre-Simon Laplace in 1774.

14.1.2 General Uses

- Signal Processing: Modeling noise in communication systems.
- Finance: Modeling price changes in financial markets
- Biology: Analyzing genetic data and protein folding
- Machine Learning: Used in some robust regression techniques

14.1.3 How the Object Works in Max

Arguments:

- lambda (optional, default 1.0): Rate parameter
- mu (optional, default 0.0): Location parameter

Inlets:

- Main inlet: Trigger generation (bang)
- Lambda inlet: Set lambda value (float)
- Mu inlet: Set mu value (float)

Outlet:

- Random number output (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information

14.1.4 What the Object Outputs

The object generates random numbers using the Laplace distribution, which is centered around mu and shaped by lambda.

14.1.5 How to Use It in Max

1. Create an `[alea.laplace]` object
2. Send it a bang to generate a new random number
3. Optionally set the lambda and mu values through the respective inlets
4. Receive the output as a float

14.1.6 Key Points

- Models distributions with both positive and negative values
- Shape controlled by λ: The parameter λ (scale parameter) controls the spread or dispersion of the distribution.
- Centered around μ: The parameter μ (location parameter) represents the peak or center of the distribution.
- Ensures reproducibility through seed setting
- The Laplace distribution's shape is controlled by λ, which dictates the steepness of the peak and the thickness of the tails. A smaller λ results in a sharper peak and thinner tails, while a larger λ results in a flatter peak and thicker tails.

14.1.7 Creative Uses

- Generating noise with both positive and negative deviations
- Modeling price changes or other bidirectional processes

14.1.8 Original Math Formula

The bilateral exponential distribution can be represented as:

$$f(x; \lambda, \mu) = \frac{1}{2\lambda} \exp\left(-\frac{|x - \mu|}{\lambda}\right)$$

15 alea.lin

15.1 Distribution: Linear Distribution

15.1.1 Brief History

The linear distribution is a simple probability distribution with a linearly decreasing probability density function. While not as commonly used as some other distributions, it can be useful in certain modeling scenarios and educational contexts due to its simplicity and intuitive nature.

15.1.2 General Uses

- Simple Modeling: Representing scenarios with linearly decreasing probabilities
- Education: Demonstrating basic concepts in probability and statistics
- Simulation: Generating random events with a bias towards lower values
- Game Design: Creating weighted random outcomes with a linear bias
- Resource Allocation: Modeling diminishing resources or returns

15.1.3 How the Object Works in Max

Arguments:

- lambda (optional): Scale parameter (default 10.0)

Inlets:

- Main inlet: Trigger generation (bang)
- Lambda inlet: Set lambda value (float/int)

Outlet:

- Linearly distributed random number (float or int)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including seed and lambda value
- Type switching: Automatically switches between float and int output based on the type of the lambda value

15.1.4 What the Object Outputs

The object generates random numbers following a linear distribution with the specified lambda parameter.

15.1.5 How to Use It in Max

1. Create an `[alea.lin]` object with an optional lambda argument
2. Optionally set the lambda value through the second inlet
3. Send a bang to the main inlet to generate a new linearly distributed random number
4. Receive the output as a float or int, depending on the type of the lambda value

15.1.6 Key Points

- The lambda parameter controls the scale of the distribution
- The probability density function decreases linearly from its maximum at 0 to 0 at lambda
- The object can output either float or int values, automatically determined by the type of the lambda value
- When outputting integers, the object uses rounding to ensure appropriate distribution
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation

15.1.7 Creative Uses

- Generative music: Create melodies with a tendency towards lower notes but occasional higher ones
- Sound design: Generate filter cutoff frequencies with a bias towards lower frequencies for darker textures
- Visual art: Produce particle systems with density decreasing linearly from a central point
- Algorithmic composition: Create rhythmic patterns with decreasing likelihood of events over time
- Interactive installations: Model user engagement that decreases linearly over time
- Data sonification: Represent data sets with linear trends in an auditory form
- Probabilistic sequencing: Create evolving patterns with linearly changing probabilities

15.1.8 Original Math Formula

The probability density function (PDF) of a linear distribution is:

$$f(x; \lambda) = \frac{2(\lambda - x)}{\lambda^2}$$

for $0 \leq x \leq \lambda$.

16 alea.log

16.1 Distribution: Logistic Distribution

16.1.1 Brief History

The logistic distribution was introduced by Pierre François Verhulst in 1844 in his study of population growth. It gained prominence in the mid-20th century through its applications in various fields, including biology, economics, and psychology. The logistic distribution is closely related to the logistic function, which is fundamental in logistic regression and neural networks.

16.1.2 General Uses

- Biology: Modeling population growth and species distribution
- Economics: Analyzing economic growth and market penetration
- Psychology: Studying reaction times and decision-making processes
- Medicine: Analyzing survival data and dose-response relationships
- Machine Learning: As the basis for logistic regression in classification problems

16.1.3 How the Object Works in Max

Arguments:

- alpha (optional): Scale parameter (default 0.5)
- beta (optional): Location parameter (default 0.5)

Inlets:

- Main inlet: Trigger generation (bang)
- Alpha inlet: Set alpha value (float/int)
- Beta inlet: Set beta value (float/int)

Outlet:

- Logistically distributed random number (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including seed, alpha, and beta values

16.1.4 What the Object Outputs

The object generates random numbers following the logistic distribution with specified scale (alpha) and location (beta) parameters.

16.1.5 How to Use It in Max

1. Create an `[alea.log]` object with optional alpha and beta arguments
2. Optionally set the alpha and beta values through the respective inlets
3. Send a bang to the main inlet to generate a new logistically distributed random number
4. Receive the output as a float

16.1.6 Key Points

- The alpha parameter controls the scale (dispersion) of the distribution
- The beta parameter determines the mean (location) of the distribution
- The logistic distribution is symmetric around its mean
- This distribution has heavier tails than a normal distribution but lighter than a Cauchy distribution
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation

16.1.7 Creative Uses

- Generative music: Create melodies or harmonies with a tendency towards central values but occasional extremes
- Sound design: Generate filter parameters or modulation depths with a balanced distribution around a central tendency
- Visual art: Produce particle systems or generative shapes with a logistic distribution of sizes or positions
- Interactive installations: Model user behavior or response times in interactive systems
- Data sonification: Represent logistically distributed data sets in an auditory form
- Algorithmic composition: Create evolving textures or densities with logistically distributed parameters
- Probabilistic sequencing: Design probability tables for event triggering with a logistic distribution

16.1.8 Original Math Formula

The probability density function (PDF) of the logistic distribution is:

$$f(x; \alpha, \beta) = \frac{e^{-(x-\beta)/\alpha}}{\alpha(1 + e^{-(x-\beta)/\alpha})^2}$$

where α is the scale parameter and β is the location parameter.

17 alea.markov

17.1 Distribution: Markov Chain

17.1.1 Brief History

Markov chains, named after Russian mathematician Andrey Markov, were first introduced in 1906. They represent a mathematical system that transitions between states according to certain probabilistic rules. Markov chains have found widespread applications in various fields due to their ability to model complex systems with memory-less state transitions.

17.1.2 General Uses

- Natural Language Processing: Modeling text generation and speech recognition
- Finance: Analyzing stock market trends and credit ratings
- Biology: Studying DNA sequences and protein structures
- Music Composition: Generating melodies and chord progressions
- Weather Forecasting: Predicting weather patterns
- Page Ranking: Powering search engine algorithms
- Game AI: Creating decision-making processes for game characters

17.1.3 How the Object Works in Max

Arguments:

- num_states (optional): Number of states in the Markov chain (default 3)

Inlets:

- Main inlet: Trigger generation of next state (bang)
- State inlet: Set initial state (int)

Outlet:

- Current state (int)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state, seed, and transition matrix
- Size message: Set the size of the transition matrix
- Set_matrix message: Set probabilities for a specific row of the transition matrix
- Reset message: Reset the transition matrix to equal probabilities

17.1.4 What the Object Outputs

The object generates states based on a Markov chain with a user-definable transition matrix.

17.1.5 How to Use It in Max

1. Create an `[alea.markov]` object with an optional number of states argument
2. Use the 'size' message to set or change the number of states
3. Use the 'set_matrix' message to define transition probabilities
4. Optionally set the initial state through the second inlet
5. Send a bang to the main inlet to generate the next state
6. Receive the current state as an integer output

17.1.6 Key Points

- States are represented by integers from 1 to num_states
- The transition matrix defines the probability of moving from one state to another
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation
- The transition matrix is automatically normalized if probabilities don't sum to 1
- The 'info' message provides a comprehensive view of the current Markov chain configuration

17.1.7 Creative Uses

- Algorithmic composition: Generate note sequences or chord progressions based on musical rules
- Interactive storytelling: Create narrative structures with probabilistic plot developments
- Generative sound design: Produce evolving textures by mapping states to synthesis parameters
- Rhythmic pattern generation: Create drum patterns or rhythmic variations with controlled randomness
- Interactive installations: Model user behavior or system responses in multi-state scenarios
- Visual art: Generate sequences of shapes, colors, or movements for generative visuals
- Game design: Create probabilistic event sequences or NPC behavior patterns

17.1.8 Original Math Formula

The state transition in a Markov chain is typically represented as:

$$P(X_{n+1} = j | X_n = i) = p_{ij}$$

where p_{ij} is the probability of transitioning from state i to state j.

18 alea.markov2

18.1 Distribution: Second-Order Markov Chain

18.1.1 Brief History

Second-order Markov chains are an extension of the Markov chain concept, introduced by Andrey Markov in the early 20th century. They provide a more sophisticated model by considering not just the current state, but also the previous state when determining the next state. This added complexity allows for more nuanced modeling of systems with longer-term dependencies.

18.1.2 General Uses

- Natural Language Processing: More accurate text generation and language modeling
- Music Composition: Creating more complex and context-aware melodic and harmonic progressions
- Weather Forecasting: Improved prediction of weather patterns considering longer-term trends
- Financial Modeling: Analyzing market trends with consideration of recent history
- Bioinformatics: Modeling DNA and protein sequences with higher-order dependencies
- Speech Recognition: Enhancing accuracy by considering phoneme sequences
- Game AI: Creating more sophisticated and context-aware decision-making processes

18.1.3 How the Object Works in Max

Arguments:

- num_states (optional): Number of states in the Markov chain (default 3)

Inlets:

- Main inlet: Trigger generation of next state (bang) or set both previous and current states (int)
- State inlet: Set current state (int)

Outlet:

- Current state (int)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current states, seed, and transition matrix
- Size message: Set the size of the transition matrix
- Set_matrix message: Set probabilities for a specific slice of the transition matrix
- Reset message: Reset the transition matrix to equal probabilities
- Set_states message: Set both previous and current states simultaneously

18.1.4 What the Object Outputs

The object generates states based on a second-order Markov chain with a user-definable 3D transition matrix.

18.1.5 How to Use It in Max

1. Create a `[markov2]` object with an optional number of states argument
2. Use the 'size' message to set or change the number of states
3. Use the 'set_matrix' message to define transition probabilities
4. Set the initial states using the 'set_states' message or through the inlets
5. Send a bang to the main inlet to generate the next state
6. Receive the current state as an integer output

18.1.6 Key Points

- States are represented by integers from 1 to num_states
- The transition matrix is three-dimensional, considering both the previous and current states
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation
- The transition matrix is automatically normalized if probabilities don't sum to 1
- The 'info' message provides a comprehensive view of the current Markov chain configuration

18.1.7 Creative Uses

- Advanced algorithmic composition: Generate more coherent and stylistically consistent musical phrases
- Complex narrative generation: Create storytelling systems with deeper context awareness
- Sophisticated sound design: Produce evolving textures and timbres with longer-term structure
- Advanced rhythmic pattern generation: Create drum patterns or polyrhythms with higher-order dependencies
- Interactive installations: Model more complex user behavior or system responses in multi-state scenarios
- Generative visual art: Create sequences of shapes, colors, or movements with more sophisticated rules
- AI-driven game mechanics: Implement more nuanced and context-aware game events or NPC behaviors

18.1.8 Original Math Formula

The state transition in a second-order Markov chain is typically represented as:

$$P(X_{n+1} = k | X_n = j, X_{n-1} = i) = p_{ijk}$$

where p_{ijk} is the probability of transitioning to state k given the previous states i and j.

19 alea.mchoice

19.1 Distribution: Discrete Choice (Multinomial Distribution)

19.1.1 Brief History

The concept of making weighted random choices has its roots in probability theory and statistics, dating back to the 18th century. This type of selection process is closely related to the multinomial distribution, first described by James Bernoulli in 1713. It has since become a fundamental tool in various fields for modeling scenarios involving multiple possible outcomes with different probabilities.

19.1.2 General Uses

- Game Design: Creating randomized events or loot drops with different probabilities
- Artificial Intelligence: Decision-making in AI agents
- Simulations: Modeling complex systems with multiple possible states or outcomes
- Music Composition: Generating musical elements with weighted probabilities
- Natural Language Processing: Word prediction and text generation
- Economics: Modeling consumer choice behavior
- Operations Research: Optimizing resource allocation with probabilistic constraints

19.1.3 How the Object Works in Max

Arguments:

- num_choices (optional): Number of choices (default 2, min 2, max 64)

Inlets:

- Main inlet: Trigger random choice (bang)
- Choice inlets: Set choice values (anything)

Outlet:

- Chosen value or list (anything)

Additional Features:

- Weights message: Set weights for each choice
- Reset_warning message: Reset the empty choice warning
- info message: Output current state including choices and weights
- Seed message: Set seed value for reproducibility

19.1.4 What the Object Outputs

The object generates a random choice among multiple options, with optional weighting for each choice.

19.1.5 How to Use It in Max

1. Create an `[alea.mchoice]` object with an optional number of choices
2. Set the possible choices by sending values to the choice inlets
3. Optionally set weights using the 'weights' message
4. Send a bang to the main inlet to trigger a random choice
5. Receive the chosen value or list from the outlet

19.1.6 Key Points

- The number of choices can be set between 2 and 64
- Each choice can be a single value (int or float) or a list of values
- Weights can be set to create non-uniform probability distributions
- Empty choices are allowed but will trigger a warning and output 0
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation

19.1.7 Creative Uses

- Algorithmic composition: Generate melodies, rhythms, or chord progressions with weighted probabilities for different musical elements
- Interactive sound installations: Create evolving soundscapes by randomly selecting and mixing audio samples with different likelihoods
- Generative visual art: Produce dynamic visuals by randomly selecting shapes, colors, or patterns with controlled probabilities
- Game development: Implement randomized events, character behaviors, or level generation with customizable probabilities
- Live performance tools: Create performance systems that introduce controlled randomness in effects processing, mixing, or sequencing
- Data sonification: Represent probabilistic data sets through sound by mapping data to weighted choices of audio parameters
- AI-driven music systems: Develop more sophisticated algorithmic decision-making for computer-generated music by combining multiple weighted choices

19.1.8 Original Math Formula

The multinomial distribution can be represented as:

$$P(X_1 = x_1, \ldots, X_k = x_k) = \frac{n!}{x_1!x_2!\cdots x_k!} p_1^{x_1} p_2^{x_2} \cdots p_k^{x_k}$$

where n is the number of trials, x_i is the number of successes for each choice, and p_i is the probability of each choice.

20 alea.oneoverf

20.1 Distribution: 1/f Noise (Pink Noise)

20.1.1 Brief History

1/f noise, also known as pink noise, was first described by W. Schottky in 1918 in the context of electron tubes. It gained prominence in various fields throughout the 20th century. The algorithm used in this object is based on the work of Clarke and Voss, who developed an efficient method for generating 1/f noise in 1978.

20.1.2 General Uses

- Audio Synthesis: Creating natural-sounding background noise or textures
- Music Composition: Generating organic rhythms or pitch sequences
- Environmental Modeling: Simulating natural phenomena like wind or water flow
- Financial Analysis: Modeling certain types of market fluctuations
- Biological Systems: Representing various biological processes and rhythms
- Signal Processing: Testing and calibrating audio equipment
- Psychology: Studying human perception of noise and its effects

20.1.3 How the Object Works in Max

Arguments:

- mode (optional): Output mode (0 for float, 1 for int, default 0)

Inlets:

- Main inlet: Trigger generation of 1/f noise (bang)
- Start value inlet: Set start value (float)
- NOP inlet: Set number of possible results (float)

Outlet:

- 1/f noise output (float or int, depending on mode)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including mode, last result, and seed
- Reset message: Reset to default start and NOP values
- Mode message: Set the output mode (float or int)

20.1.4 What the Object Outputs

The object generates 1/f noise using a simple algorithm based on Clarke and Voss's method.

20.1.5 How to Use It in Max

1. Create an `[alea.oneoverf]` object with an optional mode argument
2. Optionally set the start value and number of possible results (NOP) through the respective inlets
3. Send a bang to the main inlet to generate a new 1/f noise value
4. Receive the output as a float or int, depending on the mode setting

20.1.6 Key Points

- The start value sets the initial point for the noise generation
- The NOP value determines the range and resolution of the generated noise
- The mode setting allows for either floating-point or integer output
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation
- The generated noise follows a 1/f power spectrum, characteristic of pink noise

20.1.7 Creative Uses

- Generative music: Create evolving textures or background atmospheres with natural-sounding fluctuations
- Algorithmic composition: Generate pitch or rhythm sequences with organic variations
- Sound design: Produce natural-sounding wind, water, or ambient noise effects
- Visual art: Drive parameters for generative visuals with organic, flowing changes
- Interactive installations: Create responsive environments with naturally evolving parameters
- Data sonification: Represent data sets with 1/f characteristics through sound
- Experimental synthesis: Use as a modulation source for various synthesis parameters to create complex, evolving timbres

20.1.8 Original Math Formula

The 1/f noise, or pink noise, follows a power law where the power spectral density (PSD) is inversely proportional to the frequency:

$$S(f) \propto \frac{1}{f}$$

21 alea.pareto

21.1 Distribution: Pareto Distribution

21.1.1 Brief History

The Pareto distribution, named after Italian economist Vilfredo Pareto, was originally used to describe the allocation of wealth among individuals in a society, observing that a large portion of wealth is held by a small percentage of the population. Since its introduction in the late 19th century, it has found applications in various fields beyond economics, including physics, biology, and computer science.

21.1.2 General Uses

- Economics: Modeling income distribution and wealth inequality
- Finance: Analyzing stock market returns and price movements
- Computer Science: Describing file sizes in file systems and packet sizes in network traffic
- Physics: Studying the distribution of particle sizes in aggregates
- Geology: Modeling the size distribution of oil fields
- Actuarial Science: Analyzing extreme insurance losses
- Social Sciences: Studying the distribution of city populations

21.1.3 How the Object Works in Max

Arguments:

- shape (optional): Shape parameter (alpha) (default 1.0)
- scale (optional): Scale parameter (beta) (default 1.0)

Inlets:

- Main inlet: Trigger generation (bang)
- Shape inlet: Set shape (alpha) value (float/int)
- Scale inlet: Set scale (beta) value (float/int)

Outlet:

- Pareto-distributed random number (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state information including shape, scale, and seed values

21.1.4 What the Object Outputs

The object generates random numbers following the Pareto distribution with specified shape (alpha) and scale (beta) parameters.

21.1.5 How to Use It in Max

1. Create an `[alea.pareto]` object with optional shape and scale arguments
2. Optionally set the shape and scale values through the respective inlets
3. Send a bang to the main inlet to generate a new Pareto-distributed random number
4. Receive the output as a float

21.1.6 Key Points

- The shape parameter (alpha) controls the tail heaviness of the distribution
- The scale parameter (beta) sets the minimum possible value of the distribution
- Both shape and scale must be positive
- The Pareto distribution is known for its "long tail" property, making it suitable for modeling extreme events or highly skewed data
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation

21.1.7 Creative Uses

- Generative music: Create dramatic dynamic changes or pitch variations with occasional extreme values
- Sound design: Generate particle systems for granular synthesis with size distributions mimicking natural phenomena
- Algorithmic composition: Produce note durations or intervals with occasional very long notes or large jumps
- Visual art: Create size distributions for generative visual elements, mimicking natural patterns like city sizes or galaxy distributions
- Interactive installations: Model user engagement times or interaction intensities with a realistic distribution of occasional highly engaged users
- Data sonification: Represent Pareto-distributed data sets through sound parameters
- Experimental synthesis: Use as a control source for synthesis parameters to create evolving timbres with occasional extreme modulations

21.1.8 Original Math Formula

The probability density function (PDF) of the Pareto distribution is:

$$f(x; \alpha, \beta) = \alpha \beta^{\alpha} x^{-(\alpha+1)}$$

for $x \geq \beta$, where α is the shape parameter and β is the scale parameter.

22 alea.poisson

22.1 Distribution: Poisson Distribution

22.1.1 Brief History

The Poisson distribution, named after French mathematician Siméon Denis Poisson, was introduced in 1838. It models the probability of a given number of events occurring in a fixed interval of time or space, assuming these events occur with a known average rate and independently of the time since the last event.

22.1.2 General Uses

- Queue Theory: Modeling arrival rates of customers or calls
- Quality Control: Analyzing defects in manufacturing processes
- Biology: Studying mutation rates in DNA or cell division
- Finance: Modeling the number of trades or defaults in a given period
- Insurance: Estimating the number of claims in a time interval
- Particle Physics: Analyzing radioactive decay events
- Telecommunications: Modeling network traffic and data packet arrivals

22.1.3 How the Object Works in Max

Arguments:

- lambda (optional): Set the lambda value for the Poisson distribution (default 1.0)

Inlets:

- Main inlet: Trigger generation of random value (bang)
- Lambda inlet: Set lambda value (float/int)

Outlet:

- Random value output (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current lambda and seed values
- Type setting: Choose between integer (1) or float (0) output

22.1.4 What the Object Outputs

The object generates random numbers following the Poisson distribution with a specified lambda parameter, which represents the average rate of events in a given interval.

22.1.5 How to Use It in Max

1. Create an `[alea.poisson]` object with an optional lambda argument
2. Optionally set the lambda value through the second inlet
3. Send a bang to the main inlet to generate a new Poisson-distributed random number
4. Receive the output as a float or int, depending on the mode setting

22.1.6 Key Points

- The lambda parameter represents the average rate of occurrence for events
- Larger values of lambda result in distributions that can produce higher values more frequently
- The object can output either float or int values, automatically determined by the type of the lambda value
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation

22.1.7 Creative Uses

- Generative music: Create rhythmic patterns with event densities based on a Poisson process
- Sound design: Generate random event triggers, such as for granular synthesis or sound effects
- Visual art: Produce particle systems or visual elements with densities following a Poisson distribution
- Algorithmic composition: Generate note onsets or durations with realistic distributions
- Interactive installations: Model user interactions or engagement times with a realistic event distribution
- Data sonification: Represent Poisson-distributed data sets through sound
- Experimental synthesis: Use as a control source for synthesis parameters to create evolving textures with random events

22.1.8 Original Math Formula

The probability mass function (PMF) of the Poisson distribution is:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

for $k = 0, 1, 2, \ldots$, where λ is the average rate of occurrence.

23 alea.ran

23.1 Distribution: Uniform (with multiple modes)

23.1.1 Brief History

The Mersenne Twister is a pseudorandom number generator (PRNG) developed by Makoto Matsumoto and Takuji Nishimura in 1997. It's named after the Mersenne prime number $2^{19937} - 1$, which is used in the algorithm. The Mersenne Twister is widely used due to its long period, high-quality randomness, and efficient implementation.

23.1.2 General Uses

- Monte Carlo simulations
- Cryptography (with caution)
- Scientific modeling and research
- Computer graphics and games
- Statistical sampling
- Financial modeling
- Machine learning and artificial intelligence

23.1.3 How the Object Works in Max

Arguments:

- mode (optional): Set the output mode (1-6, default 5)
- seed (optional): Set the random seed

Inlets:

- Main inlet: Trigger generation of random number (bang)

Outlet:

- Random number output (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- mode message: Choose between different output modes (1-6)
- info message: Print current mode, count, seed, and last generated value

23.1.4 What the Object Outputs

The object generates random numbers using the Mersenne Twister algorithm with various output modes.

23.1.5 How to Use It in Max

1. Create an `[alea.ran]` object with optional mode and seed arguments
2. Use the 'mode' message to set or change the output mode
3. Use the 'seed' message to set a specific seed for reproducibility
4. Send a bang to the main inlet to generate a random number
5. Receive the generated value as a float from the outlet

23.1.6 Key Points

- The object uses both the standard library's `std::mt19937` and an original implementation of MT19937
- Six different output modes are available, offering various ranges and precisions
- The 'seed' message allows for reproducible random sequences
- The 'info' message provides comprehensive information about the current state

23.1.7 Output Modes

- `[0, 1]` uniform
- `[0, 1)` uniform
- `(0, 1)` uniform
- Original MT19937 implementation `[0, 1)`
- Default: `(0, 1)` uniform using `std::mt19937`
- `[0, 1)` with 53-bit resolution using `std::mt19937`

23.1.8 Creative Uses

- Generative Music: Create stochastic compositions with precise control over randomness
- Algorithmic Sound Design: Generate complex, evolving sound textures
- Visual Art: Produce generative visuals with controllable randomness
- Interactive Installations: Create unpredictable yet reproducible behaviors
- Data Sonification: Map data to sound parameters with high-resolution randomness
- Game Design: Implement sophisticated procedural generation techniques
- Probability-based Effects: Design audio effects with fine-grained random modulation

24 alea.rancd

24.1 Distribution: Uniform (scaled between C and D)

24.1.1 Brief History

The concept of scaling random numbers to a specific range is a fundamental technique in computer science and statistics. This object combines the high-quality Mersenne Twister random number generator with range scaling, providing a versatile tool for generating random numbers within specified bounds.

24.1.2 General Uses

- Simulations with custom ranges
- Game development for randomized events
- Generative art and music
- Statistical sampling within specific intervals
- Noise generation for audio and visual effects
- Randomized parameter selection in algorithms
- Educational tools for probability and statistics

24.1.3 How the Object Works in Max

Arguments:

- C (optional): Lower limit of the range (default 0)
- D (optional): Upper limit of the range (default 1)
- seed (optional): Set the random seed

Inlets:

- Main inlet: Trigger generation of random number (bang)
- Low limit inlet: Set the lower limit C (float/int)
- High limit inlet: Set the upper limit D (float/int)

Outlet:

- Random number output (float/int)

Additional Features:

- seed setting: Set the random seed for reproducibility
- mode message: Choose between float (0) or int (1) output
- info message: Print current mode, count, seed, last value, and range limits

24.1.4 What the Object Outputs

The object generates random numbers using the Mersenne Twister algorithm, scaled between user-defined limits C and D.

24.1.5 How to Use It in Max

1. Create an `[alea.rancd]` object with optional C, D, and seed arguments
2. Use the float or int messages to set or change the C and D limits
3. Use the 'mode' message to set float or int output
4. Send a bang to the main inlet to generate a random number
5. Receive the generated value as a float or int from the outlet

24.1.6 Key Points

- The object uses `std::mt19937` for high-quality random number generation
- Output can be set to float or int mode
- The 'seed' message allows for reproducible random sequences
- The 'info' message provides comprehensive information about the current state
- The range is inclusive of C but exclusive of D (i.e., [C, D))

24.1.7 Creative Uses

- Musical Pitch Generation: Map random values to specific pitch ranges
- Parametric Sound Design: Generate random parameter values within defined limits
- Generative Visual Art: Create randomized positions, sizes, or colors within constraints
- Algorithmic Composition: Generate note durations or velocities within musical bounds
- Interactive Installations: Produce bounded random values for various interactive elements
- Game Design: Create balanced random events or item properties
- Data Sonification: Map data ranges to specific sound parameters

24.1.8 Original Math Formula

The scaled uniform distribution can be represented as:

$$X = C + (D - C) \cdot U$$

where U is a uniformly distributed random variable in the interval [0, 1).

25 alea.tri

25.1 Distribution: Triangular Distribution

25.1.1 Brief History

The triangular distribution is a continuous probability distribution with a lower limit, upper limit, and a mode. It was first described by Simpson in 1755. This distribution is often used in business decision making and project management when limited sample data is available, but minimum, maximum, and most likely values can be estimated.

25.1.2 General Uses

- Project Management: Estimating task durations
- Risk Analysis: Modeling uncertainties in financial models
- Inventory Management: Predicting demand
- Quality Control: Representing manufacturing tolerances
- Decision Theory: Modeling subjective probabilities
- Hydrology: Estimating rainfall or flood levels
- Economics: Modeling consumer behavior

25.1.3 How the Object Works in Max

Arguments:

- a (optional): Minimum value (default 0.0)
- b (optional): Maximum value (default 1.0)
- c (optional): Mode value (default 0.5)

Inlets:

- Main inlet: Trigger generation of random value (bang)
- Minimum inlet: Set minimum value 'a' (float/int)
- Maximum inlet: Set maximum value 'b' (float/int)
- Mode inlet: Set mode value 'c' (float/int)

Outlet:

- Random value output (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current minimum, maximum, mode, and seed values
- Parameter validation: Ensures mode is between minimum and maximum

25.1.4 What the Object Outputs

The object generates random values based on the triangular distribution with user-definable minimum (a), maximum (b), and mode (c) parameters.

25.1.5 How to Use It in Max

1. Create an `[alea.tri]` object with optional a, b, and c arguments
2. Use float or int messages to set or change the a, b, and c values
3. Send a bang to the main inlet to generate a random value
4. Receive the generated value as a float from the outlet

25.1.6 Key Points

- The triangular distribution is defined by three parameters: minimum (a), maximum (b), and mode (c)
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation
- The 'seed' message allows for reproducible random sequences
- The 'info' message provides the current parameter values and seed
- Parameter validation ensures the mode is always between the minimum and maximum

25.1.7 Creative Uses

- Generative Music: Create melodies with controlled randomness
- Sound Design: Generate filter cutoff frequencies with a preferred range
- Visual Art: Produce shapes or color values with a most likely outcome
- Algorithmic Composition: Control note densities or dynamics
- Interactive Installations: Model user behavior with estimated boundaries
- Game Design: Create balanced random events with a most likely outcome
- Data Sonification: Represent asymmetric data distributions in sound

25.1.8 Original Math Formula

The probability density function (PDF) of the triangular distribution is:

$$f(x; a, b, c) = \begin{cases} \frac{2(x-a)}{(b-a)(c-a)} & \text{for } a \leq x < c, \\ \frac{2(b-x)}{(b-a)(b-c)} & \text{for } c \leq x \leq b. \end{cases}$$

26 alea.vonmises

26.1 Distribution: von Mises Distribution

26.1.1 Brief History

The von Mises distribution is a specific type of circular distribution. It's often called the circular normal distribution because it is analogous to the normal distribution but for circular data. It was introduced by Richard von Mises in 1918. It's a continuous probability distribution on the circle, analogous to the normal distribution on a line. This distribution is particularly useful for modeling directional or angular data.

26.1.2 General Uses

- Directional Statistics: Analyzing circular data in various fields
- Biology: Modeling animal movement patterns or migration directions
- Meteorology: Analyzing wind directions
- Geology: Studying the orientation of rock formations
- Oceanography: Analyzing ocean currents
- Physics: Describing phase angles in signal processing
- Music: Generating circular rhythms or pitch sequences

26.1.3 How the Object Works in Max

Arguments:

- mu (optional): Mean direction (default π)
- kappa (optional): Concentration parameter (default 2.0)

Inlets:

- Main inlet: Trigger generation of random value (bang)
- Mu inlet: Set mean direction 'mu' (float)
- Kappa inlet: Set concentration parameter 'kappa' (float)

Outlet:

- Random value output (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current mu, kappa, and seed values

26.1.4 What the Object Outputs

The object generates random values based on the von Mises distribution with user-definable mean direction (mu) and concentration (kappa) parameters.

26.1.5 How to Use It in Max

1. Create an `[alea.vonmises]` object with optional mu and kappa arguments
2. Use float or int messages to set or change the mu and kappa values
3. Send a bang to the main inlet to generate a random value
4. Receive the generated value as a float from the outlet

26.1.6 Key Points

- The von Mises distribution is defined on the interval [0, 2π)
- Mu (μ) represents the mean direction (in radians)
- Kappa (κ) controls the concentration of the distribution around mu
- When kappa is 0, the distribution becomes uniform on the circle
- As kappa increases, the distribution becomes more concentrated around mu
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation
- The 'seed' message allows for reproducible random sequences

26.1.7 Creative Uses

- Generative Music: Create circular melodies or rhythms with a preferred direction
- Sound Spatialization: Generate random angles for sound source positioning
- Visual Art: Produce circular patterns or spirals with controlled randomness
- Algorithmic Composition: Control phase relationships in additive synthesis
- Interactive Installations: Model circular or rotational user interactions
- Game Design: Create randomized but directionally biased events or movements
- Data Sonification: Represent directional data in sound

26.1.8 Original Math Formula

The probability density function (PDF) of the von Mises distribution is:

$$f(x; \mu, \kappa) = \frac{e^{\kappa \cos(x-\mu)}}{2\pi I_0(\kappa)}$$

where μ is the mean direction, κ is the concentration parameter, and I_0 is the modified Bessel function of order 0.

27 alea.walker

27.1 Distribution: Biased and Bounded Random Walk

27.1.1 Brief History

Random walks have been studied since the early 1900s, with significant contributions from mathematicians like Karl Pearson and Albert Einstein. The concept is closely related to Brownian motion, first observed by Robert Brown in 1827. Random walks have applications in various fields, from physics to economics, and form the basis for many stochastic processes.

27.1.2 General Uses

- Finance: Modeling stock prices and market trends
- Physics: Describing particle diffusion and thermal motion
- Biology: Analyzing animal foraging patterns and cell movement
- Computer Science: Developing algorithms for search and optimization
- Ecology: Modeling population dispersal and gene flow
- Psychology: Studying decision-making processes
- Music and Art: Generating evolving patterns and compositions

27.1.3 How the Object Works in Max

Arguments:

- start (optional): Initial position of the walker (default 60.0)
- maxstep (optional): Maximum step size (default 2.0)
- low (optional): Lower boundary (default 24.0)
- high (optional): Upper boundary (default 100.0)
- weight (optional): Directional bias (default 0.5)

Inlets:

- Main inlet: Trigger generation of next step (bang)
- Start inlet: Set start value (float)
- Maxstep inlet: Set maximum step size (float)
- Low inlet: Set lower boundary (float)
- High inlet: Set upper boundary (float)
- Weight inlet: Set directional bias (float)

Outlet:

- Current position of the walker (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state, including all parameters and seed

27.1.4 What the Object Outputs

The object generates a sequence of values based on a biased and bounded random walk.

27.1.5 How to Use It in Max

1. Create an `[alea.walker]` object with optional arguments
2. Use float messages to set or change the start, maxstep, low, high, and weight values
3. Send a bang to the main inlet to generate the next step
4. Receive the current position as a float from the outlet

27.1.6 Key Points

- The walker starts at the 'start' position and moves randomly within the bounds
- Each step is randomly generated, with a maximum size of 'maxstep'
- The 'weight' parameter biases the walk direction (0.5 is unbiased, ¿0.5 biases upward, ¡0.5 biases downward)
- If the walker hits a boundary, it "reflects" off it
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation

27.1.7 Creative Uses

- Generative Melodies: Create evolving melodic patterns within a specific range
- Sound Design: Control synthesis parameters for organic, evolving textures
- Algorithmic Composition: Generate pitch, rhythm, or dynamics that wander within constraints
- Interactive Installations: Model natural-feeling movements or progressions
- Visual Art: Create animations with controlled randomness
- Granular Synthesis: Control grain parameters for evolving textures
- Generative Harmony: Walk through chord progressions or harmonic spaces

27.1.8 Original Math Formula

The biased and bounded random walk can be described by the following steps:

$$X_{n+1} = X_n + \delta$$

where δ is a random step size within the bounds, influenced by the weight parameter. The walk is reflected off the boundaries if X_{n+1} exceeds the range defined by 'low' and 'high'.

28 alea.weibull

28.1 Distribution: Weibull Distribution

28.1.1 Brief History

The Weibull distribution, named after Swedish engineer Waloddi Weibull, was introduced in 1939. It is a versatile distribution that can model various types of data, including life data, reliability, and survival analysis. Its flexibility has made it popular in many fields, from engineering to medicine.

28.1.2 General Uses

- Reliability Engineering: Modeling the life of products and systems
- Survival Analysis: Studying time-to-event data in medical research
- Hydrology: Analyzing flood and drought data
- Materials Science: Modeling the strength of materials
- Meteorology: Analyzing wind speed distributions
- Finance: Modeling the time between financial transactions
- Actuarial Science: Estimating insurance risks and losses

28.1.3 How the Object Works in Max

Arguments:

- shape (optional): Shape parameter (default 1.0)
- scale (optional): Scale parameter (default 1.0)

Inlets:

- Main inlet: Trigger generation of random value (bang)
- Shape inlet: Set shape value (float/int)
- Scale inlet: Set scale value (float/int)

Outlet:

- Weibull-distributed random number (float)

Additional Features:

- seed setting: Set the random seed for reproducibility
- info message: Print current state including shape, scale, and seed values

28.1.4 What the Object Outputs

The object generates random numbers following the Weibull distribution with specified shape and scale parameters.

28.1.5 How to Use It in Max

1. Create an `[alea.weibull]` object with optional shape and scale arguments
2. Optionally set the shape and scale values through the respective inlets
3. Send a bang to the main inlet to generate a new Weibull-distributed random number
4. Receive the output as a float

28.1.6 Key Points

- The shape parameter controls the form of the distribution, with different shapes for different types of data
- The scale parameter stretches or compresses the distribution along the x-axis
- The Weibull distribution can model increasing, constant, or decreasing failure rates
- The object uses the Mersenne Twister algorithm (`std::mt19937`) for high-quality pseudo-random number generation

28.1.7 Creative Uses

- Generative music: Create note durations, velocities, or intervals with a realistic distribution.
- Sound design: Generate parameters for synthesis or effects with a distribution reflecting physical phenomena.
- Visual art: Produce particle systems or visual elements with a distribution resembling natural processes.
- Algorithmic composition: Generate rhythmic patterns or note sequences with a realistic distribution of event sizes.
- Interactive installations: Model user engagement or interaction intensities with a realistic event distribution.
- Data sonification: Represent Weibull-distributed data sets through sound.
- Experimental synthesis: Use as a control source for synthesis parameters to create evolving textures with realistic modulations.

Original Math Formula The probability density function (PDF) of the Weibull distribution is:

$$f(x; k, \lambda) = \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$$

for $x \geq 0$, where k is the shape parameter and λ is the scale parameter.

29 Max.alea Utilities

30 alea.ana

30.1 Description

Analyzes input for Markov chain transitions.

Usage

1. Create an `[alea.ana]` object in Max.
2. Send data (numbers, symbols, or lists) to the input.
3. Use a bang to output the transition matrix.
4. Use the 'get_size' message to get the current matrix size.
5. Use the 'dumpout' message to output the correlation matrix to a coll object.

How it Works

Builds a Markov chain model by analyzing sequences of input data. It tracks transitions between items and constructs a probability matrix.

Scenarios

- **Musical Composition:** Analyze existing melodies to create new ones with similar patterns.
- **Text Generation:** Model language structures for generating similar text sequences.

31 alea.bendover

31.1 Description

Folds values over limits set by the user.

Usage

1. Create an `[alea.bendover]` object with optional low and high limit arguments.
2. Send values to the first inlet to be folded.
3. Set low and high limits via the second and third inlets.

How it Works

Folds values that exceed the limits back into the range, reflecting off the boundaries to create a continuous, cyclical mapping of values.

Scenarios

- **Sound Synthesis:** Create continuous parameter changes that wrap around limits, generating interesting cyclical effects.

32 alea.border

32.1 Description

Folds over values outside limits to the limits themselves.

Usage

1. Create an `[alea.border]` object with optional low and high limit arguments.
2. Send values to the first inlet to be constrained.

3. Set low and high limits via the second and third inlets.

How it Works

Constrains values to the specified range by setting any value below the low limit to the low limit, and any value above the high limit to the high limit.

Scenarios

- **Generative Art:** Keep coordinates within a canvas, creating bouncing or reflecting effects at the edges.

33 alea.mapper

33.1 Description

Maps input values from one range to another.

Usage

1. Create an `[alea.mapper]` object with optional input range and output range arguments.
2. Send values to be mapped to the first inlet.
3. Set input and output ranges via the other inlets.
4. Use the 'scalemode' message to toggle between normal and scale-only modes.

How it Works

Linearly maps values from the input range to the output range. In normal mode, it also outputs values that fall outside the input range to separate outlets.

Scenarios

- **Music:** Map sensor data to MIDI note values.
- **Visual Art:** Map audio input to color values.

34 js alea.ana2.js

34.1 Description

JavaScript implementation of second-order Markov chain analysis and generation.

Usage

1. Create a `[js alea.ana2.js]` object in Max.
2. Send data to the object to be analyzed.
3. Use a bang to output the transition matrix.
4. Use the 'get_size' message to get the current matrix size.
5. Use the 'dumpout' message to output the correlation matrix in coll (oulet 1) and dict (outlet 2) formats.

How it Works

Builds a Markov chain model based on input sequences, capable of handling more complex state transitions (up to 2nd order Markov chains).

Scenarios

- **Text Generation:** Create evolving text sequences.
- **Musical Patterns:** Generate evolving musical patterns.
- **Interactive Installations:** Model complex state transitions.

35 Summary

These tools collectively provide a robust set of utilities for working with random processes, data mapping, and statistical modeling within the Max environment. They enable users to create complex, probabilistic, and adaptive systems for a wide range of creative applications in music, visual arts, and interactive media.

35.1 General Tips for All Utilities

- Use the 'info' message (where available) to get current settings and state.
- For objects with multiple inlets, the leftmost inlet typically triggers the main action, while others set parameters.
- These utilities can be combined in Max patches to create complex, probabilistic, and adaptive systems.

These utilities provide a flexible toolkit for handling random processes, data mapping, and statistical modeling in Max, enabling a wide range of creative applications in algorithmic composition, generative art, and interactive media.