The DISTRIBUTIONS Reference Manual

Statistical distributions and related functions, version 1.0.0

Steven Nunez <steve@symbolics.tech>

This manual was generated automatically by Declt 4.0b2.

Table of Contents

Copying

This program is distributed under the terms of the Microsoft Public License.

1 Systems

The main system appears first, followed by any subsystem dependency.

1.1 distributions

Random numbers and distributions

Long Name
Statistical distributions and related functions

Author Steven Nunez <steve@symbolics.tech>

Home Page
https://lisp-stat.dev/docs/manuals/distributions/

Source Control
(GIT https://github.com/Lisp-Stat/distributions.git)

Bug Tracker
https://github.com/Lisp-Stat/distributions/issues

License MS-PL

Long Description
The Distributions package provides a collection of probabilistic distributions and related functions including:

- Sampling from distributions
- Moments (e.g mean, variance, skewness, and kurtosis), entropy, and other properties
- Probability density/mass functions (pdf) and their logarithm (logpdf)
- Moment-generating functions and characteristic functions
- Maximum likelihood estimation
- Distribution composition and derived distributions

Version 1.0.0

Dependencies
- alexandria (system).
- anaphora (system).
- array-operations (system).
- cephes (system).
- num-utils (system).
- special-functions (system).
- let-plus (system).
- float-features (system).

Source [distributions.asd], page 5.

Child Components
- [packages.lisp], page 5 (file).
- [internals.lisp], page 5 (file).
- [generator.lisp], page 5 (file).
- [simple-multiplicative-congruential-generators.lisp], page 6 (file).
- [defs.lisp], page 7 (file).

2 Files

Files are sorted by type and then listed depth-first from the systems components trees.

2.1 Lisp

2.1.1 distributions/distributions.asd

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

ASDF Systems
[distributions], page 3.

2.1.2 distributions/packages.lisp

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Packages

- [distributions.internals], page 17.
- [distributions], page 17.

2.1.3 distributions/internals.lisp

Dependency
[packages.lisp], page 5 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface

- [as-float], page 23 (function).
- [as-float-probabilities], page 23 (function).
- [as-float-vector], page 23 (function).
- [float-vector], page 44 (type).
- [internal-float], page 44 (type).
- [maybe-ignore-constant], page 23 (macro).
- [try], page 23 (macro).
- [with-floats], page 23 (macro).

2.1.4 distributions/generator.lisp

Dependency
[internals.lisp], page 5 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface

- [generator], page 42 (class).
- [initialize-instance], page 34 (method).
- [make-generator], page 25 (function).
- [next], page 25 (function).

Internals

- [%next-double-float], page 45 (function).
- [%next-integer], page 45 (function).
- [%next-single-float], page 45 (function).
- [*default-generator-type*], page 44 (special variable).
- [chunk-length], page 56 (reader method).
- [copy-state], page 57 (generic function).
- [default-seed], page 57 (reader method).
- [generate-seed], page 47 (function).
- [generate-state], page 57 (generic function).
- [state], page 59 (reader method).
- [(setf state)], page 59 (writer method).

2.1.5 distributions/simple-multiplicative-congruential-generators.lisp

Dependency

[generator.lisp], page 5 (file).

Source [distributions.asd], page 5.

Parent Component

[distributions], page 3 (system).

Public Interface

- [borosh13], page 42 (class).
- [initialize-instance], page 34 (method).
- [randu], page 43 (class).
- [transputer], page 43 (class).
- [waterman14], page 44 (class).

Internals

- [a], page 56 (reader method).
- [clone], page 56 (method).
- [generate-state], page 57 (method).
- [m], page 57 (reader method).
- [next-chunk], page 58 (method).
- [next-real], page 58 (method).
- [simple-multiplicative-congruential], page 61 (class).

2.1.6 distributions/defs.lisp

Dependency
[simple-multiplicative-congruential-generators.lisp], page 6 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface
- [cdf], page 28 (generic function).
- [draw], page 28 (generic function).
- [generator], page 29 (generic function).
- [log-pdf], page 30 (generic function).
- [mean], page 30 (generic function).
- [pdf], page 25 (function).
- [variance], page 33 (generic function).

Internals
- [check-probability], page 46 (function).
- [define-rv], page 44 (macro).

2.1.7 distributions/generics.lisp

Dependency
[defs.lisp], page 7 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface
- [nu], page 31 (generic function).
- [standard-deviation], page 33 (generic function).

Internals
- [copy-r-univariate], page 47 (function).
- [make-r-univariate], page 50 (function).
- [r-univariate], page 60 (structure).
- [r-univariate-p], page 55 (function).
- [s^2], page 58 (generic function).

2.1.8 distributions/discrete.lisp

Dependency
[generics.lisp], page 7 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface
- [cdf], page 28 (method).

- [distinct-random-integers], page 23 (function).
- [distinct-random-integers-dense], page 24 (function).
- [draw], page 29 (method).
- [log-pdf], page 30 (method).
- [mean], page 31 (method).
- [probabilities], page 32 (method).
- [r-discrete], page 26 (function).
- [r-discrete], page 36 (structure).
- [variance], page 34 (method).

Internals

- [copy-r-discrete], page 46 (function).
- [make-r-discrete], page 49 (function).
- [r-discrete-alias], page 51 (reader).
- [(setf r-discrete-alias)], page 51 (writer).
- [r-discrete-n-float], page 52 (reader).
- [(setf r-discrete-n-float)], page 52 (writer).
- [r-discrete-p], page 52 (function).
- [r-discrete-prob], page 52 (reader).
- [(setf r-discrete-prob)], page 52 (writer).
- [r-discrete-probabilities], page 52 (reader).
- [(setf r-discrete-probabilities)], page 52 (writer).

2.1.9 distributions/uniform.lisp

Dependency
[discrete.lisp], page 7 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface

- [cdf], page 28 (method).
- [draw], page 29 (method).
- [draw-uniform], page 25 (function).
- [left], page 30 (method).
- [log-pdf], page 30 (method).
- [mean], page 31 (method).
- [quantile], page 32 (method).
- [r-uniform], page 27 (function).
- [r-uniform], page 41 (structure).
- [right], page 32 (method).
- [variance], page 34 (method).

Internals

- [copy-r-uniform], page 47 (function).

- [make-r-uniform], page 50 (function).
- [r-uniform-left], page 55 (reader).
- [(setf r-uniform-left)], page 55 (writer).
- [r-uniform-p], page 55 (function).
- [r-uniform-right], page 55 (reader).
- [(setf r-uniform-right)], page 55 (writer).
- [r-uniform-width], page 55 (reader).
- [(setf r-uniform-width)], page 55 (writer).

2.1.10 distributions/exponential.lisp

Dependency
[uniform.lisp], page 8 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface
- [cdf], page 28 (method).
- [draw], page 29 (method).
- [draw-exponential], page 24 (function).
- [draw-standard-exponential], page 24 (function).
- [log-pdf], page 30 (method).
- [mean], page 31 (method).
- [quantile], page 32 (method).
- [r-exponential], page 26 (function).
- [r-exponential], page 36 (structure).
- [rate], page 32 (method).
- [variance], page 34 (method).

Internals
- [copy-r-exponential], page 46 (function).
- [make-r-exponential], page 49 (function).
- [r-exponential-p], page 52 (function).
- [r-exponential-rate], page 52 (reader).
- [(setf r-exponential-rate)], page 52 (writer).

2.1.11 distributions/normal.lisp

Dependency
[exponential.lisp], page 9 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface
- [cdf], page 28 (method).
- [draw], page 29 (method).

- [draw-standard-normal], page 25 (function).
- [from-standard-normal], page 25 (function).
- [log-pdf], page 30 (method).
- [mean], page 31 (method).
- [quantile], page 32 (method).
- [r-normal], page 26 (function).
- [r-normal], page 39 (structure).
- [to-standard-normal], page 27 (function).
- [variance], page 34 (method).

Internals

- [+normal-log-pdf-constant+], page 44 (constant).
- [cdf-normal%], page 46 (function).
- [copy-r-normal], page 47 (function).
- [make-r-normal], page 49 (function).
- [pdf-normal%], page 50 (function).
- [quantile-normal%], page 50 (function).
- [r-normal-mean], page 54 (reader).
- [(setf r-normal-mean)], page 54 (writer).
- [r-normal-p], page 54 (function).
- [r-normal-sd], page 54 (reader).
- [(setf r-normal-sd)], page 54 (writer).
- [sd], page 58 (method).

2.1.12 distributions/log-normal.lisp

Dependency

[normal.lisp], page 9 (file).

Source [distributions.asd], page 5.

Parent Component

[distributions], page 3 (system).

Public Interface

- [cdf], page 28 (method).
- [draw], page 29 (method).
- [log-pdf], page 30 (method).
- [mean], page 31 (method).
- [quantile], page 32 (method).
- [r-log-normal], page 26 (function).
- [r-log-normal], page 39 (structure).
- [variance], page 34 (method).

Internals

- [copy-r-log-normal], page 47 (function).
- [make-r-log-normal], page 49 (function).
- [r-log-normal-log-mean], page 53 (reader).

- [(setf r-log-normal-log-mean)], page 53 (writer).
- [r-log-normal-log-sd], page 53 (reader).
- [(setf r-log-normal-log-sd)], page 53 (writer).
- [r-log-normal-p], page 54 (function).

2.1.13 distributions/truncated-normal.lisp

Dependency
[log-normal.lisp], page 10 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface

- [cdf], page 28 (method).
- [draw], page 29 (method).
- [log-pdf], page 30 (method).
- [mean], page 31 (method).
- [quantile], page 32 (method).
- [r-truncated-normal], page 27 (function).
- [variance], page 33 (method).

Internals

- [copy-left-truncated-normal], page 46 (function).
- [draw-left-truncated-standard-normal], page 47 (function).
- [left-truncated-normal], page 48 (function).
- [left-truncated-normal], page 59 (structure).
- [left-truncated-normal-alpha], page 48 (reader).
- [(setf left-truncated-normal-alpha)], page 48 (writer).
- [left-truncated-normal-left], page 48 (reader).
- [(setf left-truncated-normal-left)], page 48 (writer).
- [left-truncated-normal-left-standardized], page 48 (reader).
- [(setf left-truncated-normal-left-standardized)], page 48 (writer).
- [left-truncated-normal-m0], page 48 (reader).
- [(setf left-truncated-normal-m0)], page 48 (writer).
- [left-truncated-normal-mu], page 48 (reader).
- [(setf left-truncated-normal-mu)], page 48 (writer).
- [left-truncated-normal-p], page 48 (function).
- [left-truncated-normal-sigma], page 48 (reader).
- [(setf left-truncated-normal-sigma)], page 48 (writer).
- [make-left-truncated-normal], page 49 (function).
- [truncated-normal-moments%], page 56 (function).
- [truncated-normal-optimal-alpha], page 56 (function).

2.1.14 distributions/t-distribution.lisp

Dependency
[truncated-normal.lisp], page 11 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface

- [draw], page 29 (method).
- [draw-standard-t], page 25 (function).
- [mean], page 31 (method).
- [nu], page 32 (method).
- [r-t], page 27 (function).
- [r-t], page 40 (structure).
- [scale], page 33 (method).
- [t-scale-to-variance-coefficient], page 27 (function).
- [variance], page 33 (method).

Internals

- [copy-r-t], page 47 (function).
- [make-r-t], page 50 (function).
- [r-t-mean], page 54 (reader).
- [(setf r-t-mean)], page 54 (writer).
- [r-t-nu], page 54 (reader).
- [(setf r-t-nu)], page 54 (writer).
- [r-t-p], page 55 (function).
- [r-t-scale], page 55 (reader).
- [(setf r-t-scale)], page 55 (writer).

2.1.15 distributions/gamma.lisp

Dependency
[t-distribution.lisp], page 12 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface

- [alpha], page 27 (method).
- [alpha], page 27 (method).
- [beta], page 28 (method).
- [beta], page 28 (method).
- [cdf], page 28 (method).
- [draw], page 29 (method).
- [draw], page 29 (method).
- [log-pdf], page 30 (method).

- [log-pdf], page 30 (method).
- [mean], page 31 (method).
- [mean], page 31 (method).
- [num=], page 34 (method).
- [quantile], page 32 (method).
- [r-gamma], page 26 (function).
- [r-gamma], page 37 (structure).
- [r-inverse-gamma], page 26 (function).
- [r-inverse-gamma], page 38 (structure).
- [variance], page 33 (method).
- [variance], page 33 (method).

Internals

- [cdf-gamma%], page 45 (function).
- [cdf-gamma%+], page 45 (function).
- [copy-r-gamma], page 46 (function).
- [copy-r-inverse-gamma], page 46 (function).
- [draw-standard-gamma1], page 47 (function).
- [make-r-gamma], page 49 (function).
- [make-r-inverse-gamma], page 49 (function).
- [pdf-gamma], page 50 (function).
- [pdf-gamma%], page 50 (function).
- [pdf-gamma*], page 50 (function).
- [pdf-gamma+], page 50 (function).
- [r-gamma-alpha], page 52 (reader).
- [(setf r-gamma-alpha)], page 52 (writer).
- [r-gamma-beta], page 52 (reader).
- [(setf r-gamma-beta)], page 52 (writer).
- [r-gamma-p], page 53 (function).
- [r-inverse-gamma-alpha], page 53 (reader).
- [(setf r-inverse-gamma-alpha)], page 53 (writer).
- [r-inverse-gamma-beta], page 53 (reader).
- [(setf r-inverse-gamma-beta)], page 53 (writer).
- [r-inverse-gamma-p], page 53 (function).
- [standard-gamma1-d-c], page 55 (function).

2.1.16 distributions/chi-square.lisp

Dependency

[gamma.lisp], page 12 (file).

Source [distributions.asd], page 5.

Parent Component

[distributions], page 3 (system).

Public Interface

- [nu], page 31 (method).

- [nu], page 32 (method).
- [r-chi-square], page 26 (function).
- [r-inverse-chi-square], page 26 (function).

Internals [s^2], page 58 (method).

2.1.17 distributions/beta.lisp

Dependency
[chi-square.lisp], page 13 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface
- [alpha], page 27 (method).
- [beta], page 27 (method).
- [draw], page 29 (method).
- [mean], page 31 (method).
- [quantile], page 32 (method).
- [r-beta], page 26 (function).
- [r-beta], page 35 (structure).
- [variance], page 33 (method).

Internals
- [copy-r-beta], page 46 (function).
- [make-r-beta], page 49 (function).
- [r-beta-alpha], page 51 (reader).
- [(setf r-beta-alpha)], page 51 (writer).
- [r-beta-beta], page 51 (reader).
- [(setf r-beta-beta)], page 51 (writer).
- [r-beta-p], page 51 (function).

2.1.18 distributions/rayleigh.lisp

Dependency
[beta.lisp], page 14 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface
- [cdf], page 28 (method).
- [draw], page 29 (method).
- [draw-rayleigh], page 24 (function).
- [mean], page 31 (method).
- [r-rayleigh], page 27 (function).
- [r-rayleigh], page 40 (structure).
- [scale], page 33 (method).

- [variance], page 33 (method).

Internals

- [copy-r-rayleigh], page 47 (function).
- [make-r-rayleigh], page 49 (function).
- [r-rayleigh-p], page 54 (function).
- [r-rayleigh-scale], page 54 (reader).
- [(setf r-rayleigh-scale)], page 54 (writer).

2.1.19 distributions/bernoulli.lisp

Dependency
[rayleigh.lisp], page 14 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface

- [cdf], page 28 (method).
- [draw], page 29 (method).
- [draw-bernoulli], page 24 (function).
- [mean], page 31 (method).
- [r-bernoulli], page 25 (function).
- [r-bernoulli], page 34 (structure).
- [variance], page 33 (method).

Internals

- [copy-r-bernoulli], page 46 (function).
- [draw-bernoulli-bit], page 47 (function).
- [make-r-bernoulli], page 49 (function).
- [pr], page 58 (method).
- [r-bernoulli-p], page 50 (function).
- [r-bernoulli-pr], page 51 (reader).
- [(setf r-bernoulli-pr)], page 51 (writer).

2.1.20 distributions/binomial.lisp

Dependency
[bernoulli.lisp], page 15 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface

- [draw], page 28 (method).
- [draw-binomial], page 24 (function).
- [mean], page 31 (method).
- [r-binomial], page 26 (function).
- [r-binomial], page 35 (structure).

- [variance], page 33 (method).

Internals

- [copy-r-binomial], page 46 (function).
- [make-r-binomial], page 49 (function).
- [n], page 58 (method).
- [pr], page 58 (method).
- [r-binomial-n], page 51 (reader).
- [(setf r-binomial-n)], page 51 (writer).
- [r-binomial-p], page 51 (function).
- [r-binomial-pr], page 51 (reader).
- [(setf r-binomial-pr)], page 51 (writer).

2.1.21 distributions/geometric.lisp

Dependency
[binomial.lisp], page 15 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface

- [draw], page 28 (method).
- [draw-geometric], page 24 (function).
- [mean], page 31 (method).
- [r-geometric], page 26 (function).
- [r-geometric], page 38 (structure).
- [variance], page 33 (method).

Internals

- [copy-r-geometric], page 46 (function).
- [make-r-geometric], page 49 (function).
- [pr], page 58 (method).
- [r-geometric-p], page 53 (function).
- [r-geometric-pr], page 53 (reader).
- [(setf r-geometric-pr)], page 53 (writer).

2.1.22 distributions/poisson.lisp

Dependency
[geometric.lisp], page 16 (file).

Source [distributions.asd], page 5.

Parent Component
[distributions], page 3 (system).

Public Interface
[draw-poisson], page 24 (function).

3 Packages

Packages are listed by definition order.

3.1 distributions.internals

Source [packages.lisp], page 5.

Use List

- alexandria.
- common-lisp.
- let-plus.

Used By List

[distributions], page 17.

Public Interface

- [as-float], page 23 (function).
- [as-float-probabilities], page 23 (function).
- [as-float-vector], page 23 (function).
- [float-vector], page 44 (type).
- [internal-float], page 44 (type).
- [maybe-ignore-constant], page 23 (macro).
- [try], page 23 (macro).
- [with-floats], page 23 (macro).

3.2 distributions

Source [packages.lisp], page 5.

Use List

- alexandria.
- anaphora.
- common-lisp.
- [distributions.internals], page 17.
- let-plus.
- num-utils.elementwise.
- num-utils.matrix.
- num-utils.num=.
- special-functions.

Public Interface

- [alpha], page 27 (generic function).
- [beta], page 27 (generic function).
- [borosh13], page 42 (class).
- [cdf], page 28 (generic function).
- [distinct-random-integers], page 23 (function).
- [distinct-random-integers-dense], page 24 (function).
- [draw], page 28 (generic function).

Internals

4 Definitions

Definitions are sorted by export status, category, package, and then by lexicographic order.

4.1 Public Interface

4.1.1 Macros

`maybe-ignore-constant` (*ignore-constant? value constant*) [Macro]
Handle a constant that is calculated only when IGNORE-CONSTANT? is NIL and VALUE is not negative infinity (represented by NIL).

Package [`distributions.internals`], page 17.

Source [`internals.lisp`], page 5.

`try` ((**&rest** *bindings*) *condition value*) [Macro]
Evaluate bindings (expanding into LET+, so all features can be used) until condition is satisfied, then return value.

Package [`distributions.internals`], page 17.

Source [`internals.lisp`], page 5.

`with-floats` ((**&rest** *variables*) **&body** *body*) [Macro]
Rebind each variable, coerced to the internal float type used by DISTRIBUTIONS.

Package [`distributions.internals`], page 17.

Source [`internals.lisp`], page 5.

4.1.2 Ordinary functions

`as-float` (*x*) [Function]
Return the argument coerced to the DISTRIBUTIONS library's internal float type.

Package [`distributions.internals`], page 17.

Source [`internals.lisp`], page 5.

`as-float-probabilities` (*vector*) [Function]
Normalize vector as probabilities, assert that all are positive, return them as a VECTOR-DOUBLE-FLOAT. Vector is always copied.

Package [`distributions.internals`], page 17.

Source [`internals.lisp`], page 5.

`as-float-vector` (*vector* **&key** *copy?*) [Function]
Return VECTOR converted to another vector with elements converted to INTERNAL-FLOAT if necessary. When COPY?, the vector is always copied.

Package [`distributions.internals`], page 17.

Source [`internals.lisp`], page 5.

`distinct-random-integers` (*count limit* **&key** *rng*) [Function]
Return a vector of COUNT distinct random integers, in increasing order, drawn from the uniform discrete distribution on {0 , ..., limit-1}.

Package [`distributions`], page 17.

Source [`discrete.lisp`], page 7.

`distinct-random-integers-dense` (*count limit* **&key** *rng*) [Function]
Implementation of DISTINCT-RANDOM-INTEGERS when count/limit is (relatively) high. Implements algorithm S from @cite{taocp3}, p 142.

Package [`distributions`], page 17.

Source [`discrete.lisp`], page 7.

`draw-bernoulli` (*p* **&key** *rng*) [Function]
Return T with probability p, otherwise NIL. Rationals are handled exactly.

Package [`distributions`], page 17.

Source [`bernoulli.lisp`], page 15.

`draw-binomial` (*p n* **&key** *rng*) [Function]
Return the number of successes out of N Bernoulli trials with probability of success P.

Package [`distributions`], page 17.

Source [`binomial.lisp`], page 15.

`draw-exponential` (*rate* **&key** *rng*) [Function]
Return a random variable from the Exponential(rate) distribution which has density rate*exp(-rate*x) for x>=0 and 0 for x<0. rate > 0.

Package [`distributions`], page 17.

Source [`exponential.lisp`], page 9.

`draw-geometric` (*p* **&key** *rng*) [Function]
Return the number of Bernoulli trials, with probability of success P, that were needed to reach the first success. This is >= 1.

Package [`distributions`], page 17.

Source [`geometric.lisp`], page 16.

`draw-poisson` (*lamda* **&key** *rng*) [Function]
Return the number of events that occur with probability LAMDA. The algorithm is from Donald E. Knuth (1969). Seminumerical Algorithms. The Art of Computer Programming, Volume 2. Addison Wesley. WARNING: It's simple but only linear in the return value K and is numerically unstable for large LAMDA.

Package [`distributions`], page 17.

Source [`poisson.lisp`], page 16.

`draw-rayleigh` (*scale* **&key** *rng*) [Function]
Return a random variable from the Rayleigh(scale) distribution, where scale > 0 and density x * exp(-x^2 / (2 scale^2)) / scale^2 for x>=0 and 0 for x<0.

Package [`distributions`], page 17.

Source [`rayleigh.lisp`], page 14.

`draw-standard-exponential` (**&key** *rng*) [Function]
Return a random variable from the Exponential(1) distribution, which has density exp(-x) for x>=0 and 0 for x<0.

Package [`distributions`], page 17.

Source [`exponential.lisp`], page 9.

`draw-standard-normal` (**&key** *rng*) [Function]
Draw a random number from N(0,1).

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

`draw-standard-t` (*nu* **&key** *rng*) [Function]
Draw a standard T random variate, with NU degrees of freedom.

Package [`distributions`], page 17.

Source [`t-distribution.lisp`], page 12.

`draw-uniform` (*left right* **&key** *rng*) [Function]
Return a random variable from the uniform distribution between LEFT and RIGHT. It's type is the same as that of (- LEFT RIGHT).

Package [`distributions`], page 17.

Source [`uniform.lisp`], page 8.

`from-standard-normal` (*x mu sigma*) [Function]
Scale x from standard normal.

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

`make-generator` (**&key** *seed type*) [Function]
Make a random number generator object. SEED can be any of NIL, T, an other generator, an integer, or any type of seed that a generator of type TYPE supports: - NIL: the generator's STD-SEED is used;
- T: a random seed is used;
- a generator: a clone is returned;
- otherwise: SEED is used as depends on the generator.

Package [`distributions`], page 17.

Source [`generator.lisp`], page 5.

`next` (*limit* **&optional** *rng*) [Function]
Generates a uniformly distributed pseudo-random number greater than or equal to zero and less than LIMIT. RNG, if supplied, is the random generator to use.

Package [`distributions`], page 17.

Source [`generator.lisp`], page 5.

`pdf` (*rv x* **&optional** *ignore-constant?*) [Function]
Probability distribution function of RANDOM-VARIABLE at X. See LOG-PDF for the semantics of IGNORE-CONSTANT?.

Package [`distributions`], page 17.

Source [`defs.lisp`], page 7.

`r-bernoulli` (*pr*) [Function]

Package [`distributions`], page 17.

Source [`bernoulli.lisp`], page 15.

`r-beta` (*alpha beta*) [Function]

Package [`distributions`], page 17.

Source [`beta.lisp`], page 14.

`r-binomial` (*pr n*) [Function]

Package [`distributions`], page 17.

Source [`binomial.lisp`], page 15.

`r-chi-square` (*nu*) [Function]

Chi-square distribution with NU degrees of freedom.

Package [`distributions`], page 17.

Source [`chi-square.lisp`], page 13.

`r-discrete` (*probabilities*) [Function]

Package [`distributions`], page 17.

Source [`discrete.lisp`], page 7.

`r-exponential` (*rate*) [Function]

Package [`distributions`], page 17.

Source [`exponential.lisp`], page 9.

`r-gamma` (*alpha beta*) [Function]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

`r-geometric` (*pr*) [Function]

Package [`distributions`], page 17.

Source [`geometric.lisp`], page 16.

`r-inverse-chi-square` (*nu* **&optional** *s^2*) [Function]

Generalized inverse chi-square distribution. Reparametrized to INVERSE-GAMMA.

Package [`distributions`], page 17.

Source [`chi-square.lisp`], page 13.

`r-inverse-gamma` (*alpha beta*) [Function]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

`r-log-normal` (*log-mean log-sd*) [Function]

Package [`distributions`], page 17.

Source [`log-normal.lisp`], page 10.

`r-normal` (**&optional** *mean variance*) [Function]

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

`r-rayleigh` (*scale*) [Function]

Package [`distributions`], page 17.

Source [`rayleigh.lisp`], page 14.

`r-t` (*mean scale nu*) [Function]

Package [`distributions`], page 17.

Source [`t-distribution.lisp`], page 12.

`r-truncated-normal` (*left right* **&optional** *mu sigma*) [Function]

Truncated normal distribution. If LEFT or RIGHT is NIL, it corresponds to -/+ infinity.

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

`r-uniform` (*left right*) [Function]

Package [`distributions`], page 17.

Source [`uniform.lisp`], page 8.

`t-scale-to-variance-coefficient` (*nu*) [Function]

Return the coefficient that multiplies the Sigma matrix or the squared scale to get the variance of a (multivariate) Student-T distribution. Also checks that nu > 2, ie the variance is defined.

Package [`distributions`], page 17.

Source [`t-distribution.lisp`], page 12.

`to-standard-normal` (*x mu sigma*) [Function]

Scale x to standard normal.

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

4.1.3 Generic functions

`alpha` (*r-gamma0*) [Generic Function]

Package [`distributions`], page 17.

Methods

`alpha` ((*r-beta0 [`r-beta`], page 35*)) [Method]

Source [`beta.lisp`], page 14.

`alpha` ((*r-inverse-gamma0 [`r-inverse-gamma`], page 38*)) [Method]

Source [`gamma.lisp`], page 12.

`alpha` ((*r-gamma0 [`r-gamma`], page 37*)) [Method]

Source [`gamma.lisp`], page 12.

`beta` (*r-gamma0*) [Generic Function]

Package [`distributions`], page 17.

Methods

`beta` ((*r-beta0 [`r-beta`], page 35*)) [Method]

Source [`beta.lisp`], page 14.

`beta` ((*r-inverse-gamma0 [*`r-inverse-gamma`*], page 38*)) [Method]

Source [`gamma.lisp`], page 12.

`beta` ((*r-gamma0 [*`r-gamma`*], page 37*)) [Method]

Source [`gamma.lisp`], page 12.

`cdf` (*random-variable x*) [Generic Function]

Cumulative distribution function of RANDOM-VARIABLE at X.

Package [`distributions`], page 17.

Source [`defs.lisp`], page 7.

Methods

`cdf` ((*r-bernoulli0 [*`r-bernoulli`*], page 34*) *x*) [Method]

Source [`bernoulli.lisp`], page 15.

`cdf` ((*r-rayleigh0 [*`r-rayleigh`*], page 40*) *x*) [Method]

Source [`rayleigh.lisp`], page 14.

`cdf` ((*r-gamma0 [*`r-gamma`*], page 37*) *x*) [Method]

Source [`gamma.lisp`], page 12.

`cdf` ((*left-truncated-normal0 [*`left-truncated-normal`*], page 59*) *x*) [Method]

Source [`truncated-normal.lisp`], page 11.

`cdf` ((*r-log-normal0 [*`r-log-normal`*], page 39*) *x*) [Method]

Source [`log-normal.lisp`], page 10.

`cdf` ((*r-normal0 [*`r-normal`*], page 39*) *x*) [Method]

Source [`normal.lisp`], page 9.

`cdf` ((*r-exponential0 [*`r-exponential`*], page 36*) *x*) [Method]

Source [`exponential.lisp`], page 9.

`cdf` ((*r-uniform0 [*`r-uniform`*], page 41*) *x*) [Method]

Source [`uniform.lisp`], page 8.

`cdf` ((*instance [*`r-discrete`*], page 36*) *i*) [Method]

Source [`discrete.lisp`], page 7.

`draw` (*random-variable* **&key** *rng* **&allow-other-keys**) [Generic Function]

Draw random variates. Can also be used on generators.

Package [`distributions`], page 17.

Source [`defs.lisp`], page 7.

Methods

`draw` ((*r-geometric0 [*`r-geometric`*], page 38*) **&key** *rng*) [Method]

Source [`geometric.lisp`], page 16.

`draw` ((*r-binomial0 [*`r-binomial`*], page 35*) **&key** *rng*) [Method]

Source [`binomial.lisp`], page 15.

`draw` ((*r-bernoulli0 [`r-bernoulli`], page 34*) **&key** *rng*) [Method]

Source [`bernoulli.lisp`], page 15.

`draw` ((*r-rayleigh0 [`r-rayleigh`], page 40*) **&key** *rng*) [Method]

Source [`rayleigh.lisp`], page 14.

`draw` ((*r-beta0 [`r-beta`], page 35*) **&key** *rng*) [Method]

Source [`beta.lisp`], page 14.

`draw` ((*r-inverse-gamma0 [`r-inverse-gamma`], page 38*) **&key** *rng*) [Method]

Source [`gamma.lisp`], page 12.

`draw` ((*r-gamma0 [`r-gamma`], page 37*) **&key** *rng*) [Method]

Source [`gamma.lisp`], page 12.

`draw` ((*r-t0 [`r-t`], page 40*) **&key** *rng*) [Method]

Source [`t-distribution.lisp`], page 12.

`draw` ((*left-truncated-normal0 [`left-truncated-normal`], page 59*) **&key** *rng*) [Method]

Source [`truncated-normal.lisp`], page 11.

`draw` ((*r-log-normal0 [`r-log-normal`], page 39*) **&key** *rng*) [Method]

Source [`log-normal.lisp`], page 10.

`draw` ((*r-normal0 [`r-normal`], page 39*) **&key** *rng*) [Method]

Source [`normal.lisp`], page 9.

`draw` ((*r-exponential0 [`r-exponential`], page 36*) **&key** *rng*) [Method]

Source [`exponential.lisp`], page 9.

`draw` ((*r-uniform0 [`r-uniform`], page 41*) **&key** *rng*) [Method]

Source [`uniform.lisp`], page 8.

`draw` ((*instance [`r-discrete`], page 36*) **&key** *rng*) [Method]

Source [`discrete.lisp`], page 7.

`draw` ((*function* `function`) **&key** *rng*) [Method]

`generator` (*random-variable* **&key** *rng*) [Generic Function]

Return a closure that returns random draws.

Package [`distributions`], page 17.

Source [`defs.lisp`], page 7.

Methods

`generator` (*random-variable* **&key** *rng*) [Method]

`left` (*r-uniform0*) [Generic Function]

Package [`distributions`], page 17.

Methods

`left` ((*r-uniform0 [*`r-uniform`*], page 41*)) [Method]

Source [`uniform.lisp`], page 8.

`log-pdf` (*random-variable x* **&optional** *ignore-constant?*) [Generic Function]

Log of probability distribution function of RANDOM-VARIABLE at X. NIL corresponds to log(-infinity). When IGNORE-CONSTANT?, the result may be shifted by an arbitrary real constant that does not change between calls of the same RANDOM-VARIABLE. This may save computation, and is useful for MCMC methods, etc.

Package [`distributions`], page 17.

Source [`defs.lisp`], page 7.

Methods

`log-pdf` ((*r-inverse-gamma0 [*`r-inverse-gamma`*], page 38*) *x* **&optional** *ignore-constant?*) [Method]

Source [`gamma.lisp`], page 12.

`log-pdf` ((*r-gamma0 [*`r-gamma`*], page 37*) *x* **&optional** *ignore-constant?*) [Method]

Source [`gamma.lisp`], page 12.

`log-pdf` ((*left-truncated-normal0 [*`left-truncated-normal`*], page 59*) *x* **&optional** *ignore-constant?*) [Method]

Source [`truncated-normal.lisp`], page 11.

`log-pdf` ((*r-log-normal0 [*`r-log-normal`*], page 39*) *x* **&optional** *ignore-constant?*) [Method]

Source [`log-normal.lisp`], page 10.

`log-pdf` ((*r-normal0 [*`r-normal`*], page 39*) *x* **&optional** *ignore-constant?*) [Method]

Source [`normal.lisp`], page 9.

`log-pdf` ((*r-exponential0 [*`r-exponential`*], page 36*) *x* **&optional** *ignore-constant?*) [Method]

Source [`exponential.lisp`], page 9.

`log-pdf` ((*r-uniform0 [*`r-uniform`*], page 41*) *x* **&optional** *ignore-constant?*) [Method]

Source [`uniform.lisp`], page 8.

`log-pdf` ((*instance [*`r-discrete`*], page 36*) *i* **&optional** *ignore-constant?*) [Method]

Source [`discrete.lisp`], page 7.

`mean` (*random-variable*) [Generic Function]

Mean of random variable.

Package [`distributions`], page 17.

Source [`defs.lisp`], page 7.

Methods

`mean` ((*r-geometric0 [*`r-geometric`*], page 38*)) [Method]
Source [`geometric.lisp`], page 16.

`mean` ((*r-binomial0 [*`r-binomial`*], page 35*)) [Method]
Source [`binomial.lisp`], page 15.

`mean` ((*r-bernoulli0 [*`r-bernoulli`*], page 34*)) [Method]
Source [`bernoulli.lisp`], page 15.

`mean` ((*r-rayleigh0 [*`r-rayleigh`*], page 40*)) [Method]
Source [`rayleigh.lisp`], page 14.

`mean` ((*r-beta0 [*`r-beta`*], page 35*)) [Method]
Source [`beta.lisp`], page 14.

`mean` ((*r-inverse-gamma0 [*`r-inverse-gamma`*], page 38*)) [Method]
Source [`gamma.lisp`], page 12.

`mean` ((*r-gamma0 [*`r-gamma`*], page 37*)) [Method]
Source [`gamma.lisp`], page 12.

`mean` ((*r-t0 [*`r-t`*], page 40*)) [Method]
Source [`t-distribution.lisp`], page 12.

`mean` ((*left-truncated-normal0 [*`left-truncated-normal`*], page 59*)) [Method]
Source [`truncated-normal.lisp`], page 11.

`mean` ((*r-log-normal0 [*`r-log-normal`*], page 39*)) [Method]
Source [`log-normal.lisp`], page 10.

`mean` ((*r-normal0 [*`r-normal`*], page 39*)) [Method]
Source [`normal.lisp`], page 9.

`mean` ((*r-exponential0 [*`r-exponential`*], page 36*)) [Method]
Source [`exponential.lisp`], page 9.

`mean` ((*r-uniform0 [*`r-uniform`*], page 41*)) [Method]
Source [`uniform.lisp`], page 8.

`mean` ((*instance [*`r-discrete`*], page 36*)) [Method]
Source [`discrete.lisp`], page 7.

`nu` (*distribution*) [Generic Function]
Return the degrees of freedom when applicable.

Package [`distributions`], page 17.

Source [`generics.lisp`], page 7.

Methods

`nu` ((*r-inverse-gamma [*`r-inverse-gamma`*], page 38*)) [Method]
Source [`chi-square.lisp`], page 13.

`nu` ((*r-gamma [`r-gamma`], page 37*)) [Method]
Source [`chi-square.lisp`], page 13.

`nu` ((*r-t0 [`r-t`], page 40*)) [Method]
Source [`t-distribution.lisp`], page 12.

`probabilities` (*instance*) [Generic Function]
Package [`distributions`], page 17.
Methods

`probabilities` ((*instance [`r-discrete`], page 36*)) [Method]
Source [`discrete.lisp`], page 7.

`quantile` (*r-uniform0 p*) [Generic Function]
Package [`distributions`], page 17.
Methods

`quantile` ((*r-beta0 [`r-beta`], page 35*) *q*) [Method]
Source [`beta.lisp`], page 14.

`quantile` ((*r-gamma0 [`r-gamma`], page 37*) *q*) [Method]
Source [`gamma.lisp`], page 12.

`quantile` ((*left-truncated-normal0 [`left-truncated-normal`], page 59*) *q*) [Method]
Source [`truncated-normal.lisp`], page 11.

`quantile` ((*r-log-normal0 [`r-log-normal`], page 39*) *q*) [Method]
Source [`log-normal.lisp`], page 10.

`quantile` ((*r-normal0 [`r-normal`], page 39*) *q*) [Method]
Source [`normal.lisp`], page 9.

`quantile` ((*r-exponential0 [`r-exponential`], page 36*) *p*) [Method]
Source [`exponential.lisp`], page 9.

`quantile` ((*r-uniform0 [`r-uniform`], page 41*) *p*) [Method]
Source [`uniform.lisp`], page 8.

`rate` (*r-exponential0*) [Generic Function]
Package [`distributions`], page 17.
Methods

`rate` ((*r-exponential0 [`r-exponential`], page 36*)) [Method]
Source [`exponential.lisp`], page 9.

`right` (*r-uniform0*) [Generic Function]
Package [`distributions`], page 17.
Methods

`right` ((*r-uniform0 [`r-uniform`], page 41*)) [Method]
Source [`uniform.lisp`], page 8.

`scale` (*r-t0*) [Generic Function]

Package [`distributions`], page 17.

Methods

`scale` ((*r-rayleigh0 [`r-rayleigh`], page 40*)) [Method]

Source [`rayleigh.lisp`], page 14.

`scale` ((*r-t0 [`r-t`], page 40*)) [Method]

Source [`t-distribution.lisp`], page 12.

`standard-deviation` (*random-variable*) [Generic Function]

Standard deviation of random variable.

Package [`distributions`], page 17.

Source [`generics.lisp`], page 7.

Methods

`standard-deviation` ((*random-variable [`r-univariate`], page 60*)) [Method]

`variance` (*random-variable*) [Generic Function]

Variance of random variable.

Package [`distributions`], page 17.

Source [`defs.lisp`], page 7.

Methods

`variance` ((*r-geometric0 [`r-geometric`], page 38*)) [Method]

Source [`geometric.lisp`], page 16.

`variance` ((*r-binomial0 [`r-binomial`], page 35*)) [Method]

Source [`binomial.lisp`], page 15.

`variance` ((*r-bernoulli0 [`r-bernoulli`], page 34*)) [Method]

Source [`bernoulli.lisp`], page 15.

`variance` ((*r-rayleigh0 [`r-rayleigh`], page 40*)) [Method]

Source [`rayleigh.lisp`], page 14.

`variance` ((*r-beta0 [`r-beta`], page 35*)) [Method]

Source [`beta.lisp`], page 14.

`variance` ((*r-inverse-gamma0 [`r-inverse-gamma`], page 38*)) [Method]

Source [`gamma.lisp`], page 12.

`variance` ((*r-gamma0 [`r-gamma`], page 37*)) [Method]

Source [`gamma.lisp`], page 12.

`variance` ((*r-t0 [`r-t`], page 40*)) [Method]

Source [`t-distribution.lisp`], page 12.

`variance` ((*left-truncated-normal0 [`left-truncated-normal`], page 59*)) [Method]

Source [`truncated-normal.lisp`], page 11.

`variance` ((*r-log-normal0 [*`r-log-normal`*], page 39*)) [Method]

Source [`log-normal.lisp`], page 10.

`variance` ((*r-normal0 [*`r-normal`*], page 39*)) [Method]

Source [`normal.lisp`], page 9.

`variance` ((*r-exponential0 [*`r-exponential`*], page 36*)) [Method]

Source [`exponential.lisp`], page 9.

`variance` ((*r-uniform0 [*`r-uniform`*], page 41*)) [Method]

Source [`uniform.lisp`], page 8.

`variance` ((*instance [*`r-discrete`*], page 36*)) [Method]

Source [`discrete.lisp`], page 7.

4.1.4 Standalone methods

`initialize-instance` *:after* ((*rng [*`generator`*], page 42*) **&key** *seed* **&allow-other-keys**) [Method]

Source [`generator.lisp`], page 5.

`initialize-instance` *:after* ((*self [*`simple-multiplicative-congruential`*], page 61*) **&key &allow-other-keys**) [Method]

Source [`simple-multiplicative-congruential-generators.lisp`], page 6.

`num=` ((*a1 [*`r-inverse-gamma`*], page 38*) (*b2 [*`r-inverse-gamma`*], page 38*) **&optional** *tolerance3*) [Method]

Package `num-utils.num=`.

Source [`gamma.lisp`], page 12.

4.1.5 Structures

`r-bernoulli` [Structure]

Bernoulli(pr) distribution, with probability PR for success and 1-PR for failure.

Package [`distributions`], page 17.

Source [`bernoulli.lisp`], page 15.

Direct superclasses

[`r-univariate`], page 60.

Direct methods

- [`cdf`], page 28.
- [`draw`], page 29.
- [`mean`], page 31.
- [`pr`], page 58.
- [`variance`], page 33.

Direct slots

`pr` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-bernoulli-pr`], page 51.

Writers [`(setf r-bernoulli-pr)`], page 51.

`r-beta` [Structure]

Beta(alpha,beta) distribution, with density proportional to x^(alpha-1)*(1-x)^(beta-1).

Package [`distributions`], page 17.

Source [`beta.lisp`], page 14.

Direct superclasses

[`r-univariate`], page 60.

Direct methods

- [`alpha`], page 27.
- [`beta`], page 27.
- [`draw`], page 29.
- [`mean`], page 31.
- [`quantile`], page 32.
- [`variance`], page 33.

Direct slots

`alpha` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-beta-alpha`], page 51.

Writers [`(setf r-beta-alpha)`], page 51.

`beta` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-beta-beta`], page 51.

Writers [`(setf r-beta-beta)`], page 51.

`r-binomial` [Structure]

Binomial(pr,n) distribution, with N Bernoulli trials with probability PR for success.

Package [`distributions`], page 17.

Source [`binomial.lisp`], page 15.

Direct superclasses

[`r-univariate`], page 60.

Direct methods

- [`draw`], page 28.
- [`mean`], page 31.
- [`n`], page 58.
- [`pr`], page 58.
- [`variance`], page 33.

Direct slots

`pr` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-binomial-pr`], page 51.

Writers [`(setf r-binomial-pr)`], page 51.

`n` [Slot]

Type `integer`

Readers [`r-binomial-n`], page 51.

Writers [`(setf r-binomial-n)`], page 51.

`r-discrete` [Structure]

Discrete probabilities.

Package [`distributions`], page 17.

Source [`discrete.lisp`], page 7.

Direct superclasses

[`r-univariate`], page 60.

Direct methods

- [`cdf`], page 28.
- [`draw`], page 29.
- [`log-pdf`], page 30.
- [`mean`], page 31.
- [`probabilities`], page 32.
- [`variance`], page 34.

Direct slots

`probabilities` [Slot]

Type `distributions.internals:float-vector`

Readers [`r-discrete-probabilities`], page 52.

Writers [`(setf r-discrete-probabilities)`], page 52.

`prob` [Slot]

Type `distributions.internals:float-vector`

Readers [`r-discrete-prob`], page 52.

Writers [`(setf r-discrete-prob)`], page 52.

`alias` [Slot]

Type `(simple-array fixnum (*))`

Readers [`r-discrete-alias`], page 51.

Writers [`(setf r-discrete-alias)`], page 51.

`n-float` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-discrete-n-float`], page 52.

Writers [`(setf r-discrete-n-float)`], page 52.

`r-exponential` [Structure]

Exponential(rate) distribution, with density rate*exp(-rate*x) for x>=0 and 0 for x<0. rate > 0.

Package [`distributions`], page 17.

Source [exponential.lisp], page 9.

Direct superclasses
[r-univariate], page 60.

Direct methods
- [cdf], page 28.
- [draw], page 29.
- [log-pdf], page 30.
- [mean], page 31.
- [quantile], page 32.
- [rate], page 32.
- [variance], page 34.

Direct slots

`rate` [Slot]

Type `distributions.internals:internal-float`

Readers [r-exponential-rate], page 52.

Writers [(setf r-exponential-rate)], page 52.

`r-gamma` [Structure]

Gamma(alpha,beta) distribution, with density proportional to x^(alpha-1) exp(-x*beta). Alpha and beta are known as shape and inverse scale (or rate) parameters, respectively.

Package [distributions], page 17.

Source [gamma.lisp], page 12.

Direct superclasses
[r-univariate], page 60.

Direct methods
- [alpha], page 27.
- [beta], page 28.
- [cdf], page 28.
- [draw], page 29.
- [log-pdf], page 30.
- [mean], page 31.
- [nu], page 32.
- [quantile], page 32.
- [variance], page 33.

Direct slots

`alpha` [Slot]

Type `distributions.internals:internal-float`

Readers [r-gamma-alpha], page 52.

Writers [(setf r-gamma-alpha)], page 52.

`beta` [Slot]

Type `distributions.internals:internal-float`

Readers [r-gamma-beta], page 52.

Writers [(setf r-gamma-beta)], page 52.

`r-geometric` [Structure]

Geometric(pr) distribution.

Package [`distributions`], page 17.

Source [`geometric.lisp`], page 16.

Direct superclasses

[`r-univariate`], page 60.

Direct methods

- [`draw`], page 28.
- [`mean`], page 31.
- [`pr`], page 58.
- [`variance`], page 33.

Direct slots

`pr` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-geometric-pr`], page 53.

Writers [`(setf r-geometric-pr)`], page 53.

`r-inverse-gamma` [Structure]

Inverse-Gamma(alpha,beta) distribution, with density p(x) proportional to x^(-alpha+1) exp(-beta/x)

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

Direct superclasses

[`r-univariate`], page 60.

Direct methods

- [`alpha`], page 27.
- [`beta`], page 28.
- [`draw`], page 29.
- [`log-pdf`], page 30.
- [`mean`], page 31.
- [`nu`], page 31.
- [`num=`], page 34.
- [`s^2`], page 58.
- [`variance`], page 33.

Direct slots

`alpha` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-inverse-gamma-alpha`], page 53.

Writers [`(setf r-inverse-gamma-alpha)`], page 53.

`beta` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-inverse-gamma-beta`], page 53.

Writers [`(setf r-inverse-gamma-beta)`], page 53.

`r-log-normal` [Structure]

Log-normal distribution with location log-mean and scale log-sd.

Package [`distributions`], page 17.

Source [`log-normal.lisp`], page 10.

Direct superclasses

[`r-univariate`], page 60.

Direct methods

- [`cdf`], page 28.
- [`draw`], page 29.
- [`log-pdf`], page 30.
- [`mean`], page 31.
- [`quantile`], page 32.
- [`variance`], page 34.

Direct slots

`log-mean` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-log-normal-log-mean`], page 53.

Writers [`(setf r-log-normal-log-mean)`], page 53.

`log-sd` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-log-normal-log-sd`], page 53.

Writers [`(setf r-log-normal-log-sd)`], page 53.

`r-normal` [Structure]

Normal(mean,variance) distribution.

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

Direct superclasses

[`r-univariate`], page 60.

Direct methods

- [`cdf`], page 28.
- [`draw`], page 29.
- [`log-pdf`], page 30.
- [`mean`], page 31.
- [`quantile`], page 32.
- [`sd`], page 58.
- [`variance`], page 34.

Direct slots

`mean` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-normal-mean`], page 54.

Writers [`(setf r-normal-mean)`], page 54.

`sd` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-normal-sd`], page 54.

Writers [`(setf r-normal-sd)`], page 54.

`r-rayleigh` [Structure]

Rayleigh(scale) distribution with scale > 0 and density x * exp(-x^2 / (2 scale^2)) / scale^2 for x>=0 and 0 for x<0.

Package [`distributions`], page 17.

Source [`rayleigh.lisp`], page 14.

Direct superclasses

[`r-univariate`], page 60.

Direct methods

- [`cdf`], page 28.
- [`draw`], page 29.
- [`mean`], page 31.
- [`scale`], page 33.
- [`variance`], page 33.

Direct slots

`scale` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-rayleigh-scale`], page 54.

Writers [`(setf r-rayleigh-scale)`], page 54.

`r-t` [Structure]

T(mean,scale,nu) random variate.

Package [`distributions`], page 17.

Source [`t-distribution.lisp`], page 12.

Direct superclasses

[`r-univariate`], page 60.

Direct methods

- [`draw`], page 29.
- [`mean`], page 31.
- [`nu`], page 32.
- [`scale`], page 33.
- [`variance`], page 33.

Direct slots

`mean` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-t-mean`], page 54.

Writers [`(setf r-t-mean)`], page 54.

`scale` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-t-scale`], page 55.

Writers [`(setf r-t-scale)`], page 55.

`nu` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-t-nu`], page 54.

Writers [`(setf r-t-nu)`], page 54.

`r-uniform` [Structure]

Uniform(left,right) distribution.

Package [`distributions`], page 17.

Source [`uniform.lisp`], page 8.

Direct superclasses

[`r-univariate`], page 60.

Direct methods

- [`cdf`], page 28.
- [`draw`], page 29.
- [`left`], page 30.
- [`log-pdf`], page 30.
- [`mean`], page 31.
- [`quantile`], page 32.
- [`right`], page 32.
- [`variance`], page 34.

Direct slots

`left` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-uniform-left`], page 55.

Writers [`(setf r-uniform-left)`], page 55.

`right` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-uniform-right`], page 55.

Writers [`(setf r-uniform-right)`], page 55.

`width` [Slot]

Type `distributions.internals:internal-float`

Readers [`r-uniform-width`], page 55.

Writers [`(setf r-uniform-width)`], page 55.

4.1.6 Classes

`borosh13` [Class]

Donald E. Knuth's Borosh-Niederreiter, The Art of Computer Programming, Volume 2, Third Edition, Addison-Wesley, pp 106-108.

Package [`distributions`], page 17.

Source [`simple-multiplicative-congruential-generators.lisp`], page 6.

Direct superclasses

[`simple-multiplicative-congruential`], page 61.

Direct slots

`a` [Slot]

Initform `1812433253`

`chunk-length` [Slot]

Initform `32`

`generator` [Class]

Base class for random number generators.

Package [`distributions`], page 17.

Source [`generator.lisp`], page 5.

Direct subclasses

[`simple-multiplicative-congruential`], page 61.

Direct methods

- [`chunk-length`], page 56.
- [`copy-state`], page 57.
- [`default-seed`], page 57.
- [`generate-state`], page 57.
- [`initialize-instance`], page 34.
- [`(setf state)`], page 59.
- [`state`], page 59.

Direct slots

`state` [Slot]

All information needed by the generator to create the next chunk of random bits. This state is modified after each call to NEXT-CHUNK.

Initargs `:state`

Readers [`state`], page 59.

Writers [`(setf state)`], page 59.

`min` [Slot]

The minimum value return by NEXT-CHUNK.

Package `common-lisp`.

Initargs `:min`

`max` [Slot]
The maximum value return by NEXT-CHUNK.

Package `common-lisp`.

Initargs `:max`

`chunk-length` [Slot]
The length in bits of the integer returned by NEXT-CHUNK.

Readers [`chunk-length`], page 56.

Writers *This slot is read-only.*

`default-seed` [Slot]
The seed used by default, when the seed is NIL.

Initform `0`

Readers [`default-seed`], page 57.

Writers *This slot is read-only.*

`randu` [Class]
The poor IBM randu generator. Park and Miller, Random Number Generators: Good ones are hard to find, Communications of the ACM, October 1988, Volume 31, No 10, pp 1192-1201.

Package [`distributions`], page 17.

Source [`simple-multiplicative-congruential-generators.lisp`], page 6.

Direct superclasses
[`simple-multiplicative-congruential`], page 61.

Direct slots

`a` [Slot]
Initform `65539`

`chunk-length` [Slot]
Initform `31`

`transputer` [Class]
INMOS Transputer Development System generator.

Package [`distributions`], page 17.

Source [`simple-multiplicative-congruential-generators.lisp`], page 6.

Direct superclasses
[`simple-multiplicative-congruential`], page 61.

Direct slots

`a` [Slot]
Initform `1664525`

`chunk-length` [Slot]
Initform `32`

`waterman14` [Class]
Donald E. Knuth's Waterman, The Art of Computer Programming, Volume 2, Third Edition, Addison-Wesley, pp 106-108.

Package [`distributions`], page 17.

Source [`simple-multiplicative-congruential-generators.lisp`], page 6.

Direct superclasses
[`simple-multiplicative-congruential`], page 61.

Direct slots

`a` [Slot]
Initform `1566083941`

`chunk-length` [Slot]
Initform `32`

4.1.7 Types

`float-vector` (*&optional n*) [Type]

Package [`distributions.internals`], page 17.

Source [`internals.lisp`], page 5.

`internal-float` (*&optional lower-limit upper-limit*) [Type]
Type used for internal representation of floats in the DISTRIBUTIONS library.

Package [`distributions.internals`], page 17.

Source [`internals.lisp`], page 5.

4.2 Internals

4.2.1 Constants

`+normal-log-pdf-constant+` [Constant]
Normalizing constant for a standard normal PDF.

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

4.2.2 Special variables

`*default-generator-type*` [Special Variable]

Package [`distributions`], page 17.

Source [`generator.lisp`], page 5.

4.2.3 Macros

`define-rv` (*name constructor-lambda-list options slots constructor-form* **&rest** *methods*) [Macro]
Define a random variable, abstracting from the representation. Syntax:

NAME is a symbol, and will name the class and the creation function.

CONSTRUCTOR-LAMBDA-LIST will be used to wrap the CONSTRUCTOR-FORM, which can use the locally define macro (MAKE :slot-name value1 ...) to initialize slots.

SLOTS is a list of (slot-name &key type read-only reader) slot specifications. When READER is T, SLOT-NAME is used instead, otherwise a method is defined using the given symbol.

OPTIONS is (&key documentation instance), the default instance is a gensym.

METHODS are (function-name lambda-list &body body), with (INSTANCE NAME) prepended to the lambda-list, ie the instance is accessible using INSTANCE. Also, within BODY, slots are accessible by their names.

Package [distributions], page 17.

Source [defs.lisp], page 7.

4.2.4 Ordinary functions

`%next-double-float` (*limit rng*) [Function]

Handle the single or double float case of RANDOM. We generate a float in [0d0, 1d0) by clobbering the mantissa of 1d0 with random bits (52 bits); this yields a number in [1d0, 2d0). Then 1d0 is subtracted.

Package [distributions], page 17.

Source [generator.lisp], page 5.

`%next-integer` (*limit rng*) [Function]

Generates an integer greater than or equal to zero and less than LIMIT. Successive chunks are concatenated without overlap to construct integers larger than a single chunk. The return value has this property: If two integers are generated from the same RNG with LIMIT equal to 2^m and 2^n, respectively, then bit k is the same in both integers for 0 <= k < min(m,n). Each call to %NEXT-INTEGER consumes at least one chunk; bits left over from previous chunks are not re-used.

Package [distributions], page 17.

Source [generator.lisp], page 5.

`%next-single-float` (*limit rng*) [Function]

Handle the single or double float case of RANDOM. We generate a float in [0f0, 1f0) by clobbering the mantissa of 1f0 with random bits (23 bits); this yields a number in [1f0, 2f0). Then 1f0 is subtracted.

Package [distributions], page 17.

Source [generator.lisp], page 5.

`cdf-gamma%` (*x shape* **&key** *rate scale upper-tail log*) [Function]

CDF of Gamma with parameterisation like that of R pgamma

Package [distributions], page 17.

Source [gamma.lisp], page 12.

`cdf-gamma%+` (*x k θ*) [Function]

Return the cumulative gamma distribution function, shape k>0, scale θ>0

Package [distributions], page 17.

Source [gamma.lisp], page 12.

`cdf-normal%` (*x mu sigma*) [Function]
Internal function for normal CDF.

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

`check-probability` (*p* **&optional** *open*) [Function]
Assert that P is a probability (ie a real number between 0 and 1). When OPEN is given, it is checked that p is not 0 (:LEFT), 1 (:RIGHT), or 0/1 (:BOTH).

Package [`distributions`], page 17.

Source [`defs.lisp`], page 7.

`copy-left-truncated-normal` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

`copy-r-bernoulli` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`bernoulli.lisp`], page 15.

`copy-r-beta` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`beta.lisp`], page 14.

`copy-r-binomial` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`binomial.lisp`], page 15.

`copy-r-discrete` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`discrete.lisp`], page 7.

`copy-r-exponential` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`exponential.lisp`], page 9.

`copy-r-gamma` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

`copy-r-geometric` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`geometric.lisp`], page 16.

`copy-r-inverse-gamma` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

`copy-r-log-normal` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`log-normal.lisp`], page 10.

`copy-r-normal` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

`copy-r-rayleigh` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`rayleigh.lisp`], page 14.

`copy-r-t` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`t-distribution.lisp`], page 12.

`copy-r-uniform` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`uniform.lisp`], page 8.

`copy-r-univariate` (*instance*) [Function]

Package [`distributions`], page 17.

Source [`generics.lisp`], page 7.

`draw-bernoulli-bit` (*p* **&key** *rng*) [Function]

Package [`distributions`], page 17.

Source [`bernoulli.lisp`], page 15.

`draw-left-truncated-standard-normal` (*left alpha* **&key** *rng*) [Function]

Draw a left truncated standard normal, using an Exp(alpha,left) distribution. LEFT is the standardized boundary, ALPHA should be calculated with TRUNCATED-NORMAL-OPTIMAL-ALPHA.

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

`draw-standard-gamma1` (*alpha d c* **&key** *rng*) [Function]

Return a standard gamma variate (beta=1) with shape parameter alpha >= 1. See Marsaglia and Tsang (2004). You should precalculate d and c using the utility function above.

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

`generate-seed` () [Function]

Return a 64-bit random seed, based on current time.

Package [`distributions`], page 17.

Source [`generator.lisp`], page 5.

`left-truncated-normal` (*mu sigma left*) [Function]

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

`left-truncated-normal-alpha` (*instance*) [Reader]
`(setf left-truncated-normal-alpha)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

Target Slot
[`alpha`], page 60.

`left-truncated-normal-left` (*instance*) [Reader]
`(setf left-truncated-normal-left)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

Target Slot
[`left`], page 59.

`left-truncated-normal-left-standardized` (*instance*) [Reader]
`(setf left-truncated-normal-left-standardized)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

Target Slot
[`left-standardized`], page 60.

`left-truncated-normal-m0` (*instance*) [Reader]
`(setf left-truncated-normal-m0)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

Target Slot
[`m0`], page 60.

`left-truncated-normal-mu` (*instance*) [Reader]
`(setf left-truncated-normal-mu)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

Target Slot
[`mu`], page 59.

`left-truncated-normal-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

`left-truncated-normal-sigma` (*instance*) [Reader]
`(setf left-truncated-normal-sigma)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

Target Slot
[`sigma`], page 59.

`make-left-truncated-normal` (**&key** *mu sigma left left-standardized m0 alpha*) [Function]

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

`make-r-bernoulli` (**&key** *pr*) [Function]

Package [`distributions`], page 17.

Source [`bernoulli.lisp`], page 15.

`make-r-beta` (**&key** *alpha beta*) [Function]

Package [`distributions`], page 17.

Source [`beta.lisp`], page 14.

`make-r-binomial` (**&key** *pr n*) [Function]

Package [`distributions`], page 17.

Source [`binomial.lisp`], page 15.

`make-r-discrete` (**&key** *probabilities prob alias n-float*) [Function]

Package [`distributions`], page 17.

Source [`discrete.lisp`], page 7.

`make-r-exponential` (**&key** *rate*) [Function]

Package [`distributions`], page 17.

Source [`exponential.lisp`], page 9.

`make-r-gamma` (**&key** *alpha beta*) [Function]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

`make-r-geometric` (**&key** *pr*) [Function]

Package [`distributions`], page 17.

Source [`geometric.lisp`], page 16.

`make-r-inverse-gamma` (**&key** *alpha beta*) [Function]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

`make-r-log-normal` (**&key** *log-mean log-sd*) [Function]

Package [`distributions`], page 17.

Source [`log-normal.lisp`], page 10.

`make-r-normal` (**&key** *mean sd*) [Function]

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

`make-r-rayleigh` (**&key** *scale*) [Function]

Package [`distributions`], page 17.

Source [`rayleigh.lisp`], page 14.

`make-r-t` (**&key** *mean scale nu*) [Function]

Package [distributions], page 17.

Source [t-distribution.lisp], page 12.

`make-r-uniform` (**&key** *left right width*) [Function]

Package [distributions], page 17.

Source [uniform.lisp], page 8.

`make-r-univariate` (**&key**) [Function]

Package [distributions], page 17.

Source [generics.lisp], page 7.

`pdf-gamma` (*x a b*) [Function]

Return the probability density function of a gamma distribution with shape a>0, scale b>0
Returns: x^(a-1)*exp(-x/b)/gamma(a)/b^a, x>0

Package [distributions], page 17.

Source [gamma.lisp], page 12.

`pdf-gamma%` (*x shape scale*) [Function]

Package [distributions], page 17.

Source [gamma.lisp], page 12.

`pdf-gamma*` (*x shape scale*) [Function]

Package [distributions], page 17.

Source [gamma.lisp], page 12.

`pdf-gamma+` (*x k* θ) [Function]

Return the probability density function where:

K is the shape of the distribution
θ (theta) is the scale
X is the random variate

Package [distributions], page 17.

Source [gamma.lisp], page 12.

`pdf-normal%` (*x* **&key** *mu sigma*) [Function]

Direct calculation of the Probability Density of the normal distribution.

Package [distributions], page 17.

Source [normal.lisp], page 9.

`quantile-normal%` (*q mu sigma*) [Function]

Internal function for normal quantile.

Package [distributions], page 17.

Source [normal.lisp], page 9.

`r-bernoulli-p` (*object*) [Function]

Package [distributions], page 17.

Source [bernoulli.lisp], page 15.

`r-bernoulli-pr` (*instance*) [Reader]
`(setf r-bernoulli-pr)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`bernoulli.lisp`], page 15.

Target Slot
[`pr`], page 34.

`r-beta-alpha` (*instance*) [Reader]
`(setf r-beta-alpha)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`beta.lisp`], page 14.

Target Slot
[`alpha`], page 35.

`r-beta-beta` (*instance*) [Reader]
`(setf r-beta-beta)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`beta.lisp`], page 14.

Target Slot
[`beta`], page 35.

`r-beta-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`beta.lisp`], page 14.

`r-binomial-n` (*instance*) [Reader]
`(setf r-binomial-n)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`binomial.lisp`], page 15.

Target Slot
[`n`], page 36.

`r-binomial-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`binomial.lisp`], page 15.

`r-binomial-pr` (*instance*) [Reader]
`(setf r-binomial-pr)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`binomial.lisp`], page 15.

Target Slot
[`pr`], page 35.

`r-discrete-alias` (*instance*) [Reader]
`(setf r-discrete-alias)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`discrete.lisp`], page 7.

Target Slot
[`alias`], page 36.

`r-discrete-n-float` (*instance*) [Reader]
`(setf r-discrete-n-float)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`discrete.lisp`], page 7.

Target Slot
[`n-float`], page 36.

`r-discrete-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`discrete.lisp`], page 7.

`r-discrete-prob` (*instance*) [Reader]
`(setf r-discrete-prob)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`discrete.lisp`], page 7.

Target Slot
[`prob`], page 36.

`r-discrete-probabilities` (*instance*) [Reader]
`(setf r-discrete-probabilities)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`discrete.lisp`], page 7.

Target Slot
[`probabilities`], page 36.

`r-exponential-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`exponential.lisp`], page 9.

`r-exponential-rate` (*instance*) [Reader]
`(setf r-exponential-rate)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`exponential.lisp`], page 9.

Target Slot
[`rate`], page 37.

`r-gamma-alpha` (*instance*) [Reader]
`(setf r-gamma-alpha)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

Target Slot
[`alpha`], page 37.

`r-gamma-beta` (*instance*) [Reader]
`(setf r-gamma-beta)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

Target Slot
[`beta`], page 37.

`r-gamma-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

`r-geometric-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`geometric.lisp`], page 16.

`r-geometric-pr` (*instance*) [Reader]
`(setf r-geometric-pr)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`geometric.lisp`], page 16.

Target Slot
[`pr`], page 38.

`r-inverse-gamma-alpha` (*instance*) [Reader]
`(setf r-inverse-gamma-alpha)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

Target Slot
[`alpha`], page 38.

`r-inverse-gamma-beta` (*instance*) [Reader]
`(setf r-inverse-gamma-beta)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

Target Slot
[`beta`], page 38.

`r-inverse-gamma-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

`r-log-normal-log-mean` (*instance*) [Reader]
`(setf r-log-normal-log-mean)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`log-normal.lisp`], page 10.

Target Slot
[`log-mean`], page 39.

`r-log-normal-log-sd` (*instance*) [Reader]
`(setf r-log-normal-log-sd)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`log-normal.lisp`], page 10.

Target Slot
[`log-sd`], page 39.

`r-log-normal-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`log-normal.lisp`], page 10.

`r-normal-mean` (*instance*) [Reader]
`(setf r-normal-mean)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

Target Slot
[`mean`], page 39.

`r-normal-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

`r-normal-sd` (*instance*) [Reader]
`(setf r-normal-sd)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`normal.lisp`], page 9.

Target Slot
[`sd`], page 40.

`r-rayleigh-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`rayleigh.lisp`], page 14.

`r-rayleigh-scale` (*instance*) [Reader]
`(setf r-rayleigh-scale)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`rayleigh.lisp`], page 14.

Target Slot
[`scale`], page 40.

`r-t-mean` (*instance*) [Reader]
`(setf r-t-mean)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`t-distribution.lisp`], page 12.

Target Slot
[`mean`], page 40.

`r-t-nu` (*instance*) [Reader]
`(setf r-t-nu)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`t-distribution.lisp`], page 12.

Target Slot
[`nu`], page 41.

`r-t-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`t-distribution.lisp`], page 12.

`r-t-scale` (*instance*) [Reader]
`(setf r-t-scale)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`t-distribution.lisp`], page 12.

Target Slot
[`scale`], page 41.

`r-uniform-left` (*instance*) [Reader]
`(setf r-uniform-left)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`uniform.lisp`], page 8.

Target Slot
[`left`], page 41.

`r-uniform-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`uniform.lisp`], page 8.

`r-uniform-right` (*instance*) [Reader]
`(setf r-uniform-right)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`uniform.lisp`], page 8.

Target Slot
[`right`], page 41.

`r-uniform-width` (*instance*) [Reader]
`(setf r-uniform-width)` (*instance*) [Writer]

Package [`distributions`], page 17.

Source [`uniform.lisp`], page 8.

Target Slot
[`width`], page 41.

`r-univariate-p` (*object*) [Function]

Package [`distributions`], page 17.

Source [`generics.lisp`], page 7.

`standard-gamma1-d-c` (*alpha*) [Function]
Return precalculated constants (values d c), useful for drawing from a gamma distribution.

Package [`distributions`], page 17.

Source [`gamma.lisp`], page 12.

`truncated-normal-moments%` (*n mu sigma left right* **&optional** *m0*) [Function]
N=0 gives the total mass of the truncated normal, used for normalization,
N=1 the mean, and N=2 the variance. where p(x) is the normal density. When LEFT or RIGHT are NIL, they are taken to be - or + infinity, respectively. M0 may be provided for efficiency if would be calculated multiple times. The formulas are from Jawitz (2004).

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

`truncated-normal-optimal-alpha` (*left*) [Function]
Calculate optimal exponential parameter for left-truncated normals. LEFT is the standardized boundary.

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

4.2.5 Generic functions

`a` (*object*) [Generic Reader]

Package [`distributions`], page 17.

Methods

`a` ((*simple-multiplicative-congruential* *[`simple-multiplicative-congruential`], page 61*)) [Reader Method]
The multiplier of the sequence x(n+1) = A * x(n) mod M.

Source [`simple-multiplicative-congruential-generators.lisp`], page 6.

Target Slot
[`a`], page 61.

`chunk-length` (*object*) [Generic Reader]

Package [`distributions`], page 17.

Methods

`chunk-length` ((*generator* *[`generator`], page 42*)) [Reader Method]
The length in bits of the integer returned by NEXT-CHUNK.

Source [`generator.lisp`], page 5.

Target Slot
[`chunk-length`], page 43.

`clone` (*self*) [Generic Function]

Package [`distributions`], page 17.

Methods

`clone` ((*self* *[`simple-multiplicative-congruential`], page 61*)) [Method]

Source [`simple-multiplicative-congruential-generators.lisp`], page 6.

`copy-state` (*rng*) [Generic Function]
Return a deep copy of RNG. The stream of random numbers drawn from RNG and its clone should be the same (given you draw according to the same distributions).

Package [`distributions`], page 17.

Source [`generator.lisp`], page 5.

Methods

`copy-state` ((*rng* [`generator`], *page 42*)) [Method]

`copy-state` ((*rng* `random-state`)) [Method]

`default-seed` (*object*) [Generic Reader]

Package [`distributions`], page 17.

Methods

`default-seed` ((*generator* [`generator`], *page 42*)) [Reader Method]
The seed used by default, when the seed is NIL.

Source [`generator.lisp`], page 5.

Target Slot
[`default-seed`], page 43.

`generate-state` (*rng seed*) [Generic Function]
Return a state for a generator of RNG's type using seed.

Package [`distributions`], page 17.

Source [`generator.lisp`], page 5.

Methods

`generate-state` ((*self* [`simple-multiplicative-congruential`], *page 61*) *seed*) [Method]

Source [`simple-multiplicative-congruential-generators.lisp`], page 6.

`generate-state` ((*rng* [`generator`], *page 42*) *seed*) [Method]

`m` (*object*) [Generic Reader]

Package [`distributions`], page 17.

Methods

`m` ((*simple-multiplicative-congruential* [`simple-multiplicative-congruential`], *page 61*)) [Reader Method]
The modulo of the sequence x(n+1) = A * x(n) mod M.

Source [`simple-multiplicative-congruential-generators.lisp`], page 6.

Target Slot
[`m`], page 61.

`n` (*r-binomial0*) [Generic Function]

Package [`distributions`], page 17.

Methods

`n` ((*r-binomial0 [r-binomial], page 35*)) [Method]
Source [binomial.lisp], page 15.

`next-chunk` (*self*) [Generic Function]
Package [distributions], page 17.
Methods

`next-chunk` ((*self [simple-multiplicative-congruential], page 61*)) [Method]
Source [simple-multiplicative-congruential-generators.lisp], page 6.

`next-real` (*self*) [Generic Function]
Package [distributions], page 17.
Methods

`next-real` ((*self [simple-multiplicative-congruential], page 61*)) [Method]
Source [simple-multiplicative-congruential-generators.lisp], page 6.

`pr` (*r-bernoulli0*) [Generic Function]
Package [distributions], page 17.
Methods

`pr` ((*r-geometric0 [r-geometric], page 38*)) [Method]
Source [geometric.lisp], page 16.

`pr` ((*r-binomial0 [r-binomial], page 35*)) [Method]
Source [binomial.lisp], page 15.

`pr` ((*r-bernoulli0 [r-bernoulli], page 34*)) [Method]
Source [bernoulli.lisp], page 15.

`s^2` (*distribution*) [Generic Function]
Return the scale when applicable.
Package [distributions], page 17.
Source [generics.lisp], page 7.
Methods

`s^2` ((*r-inverse-gamma [r-inverse-gamma], page 38*)) [Method]
Source [chi-square.lisp], page 13.

`sd` (*r-normal0*) [Generic Function]
Package [distributions], page 17.
Methods

`sd` ((*r-normal0 [r-normal], page 39*)) [Method]
Source [normal.lisp], page 9.

`state` (*object*) [Generic Reader]
`(setf state)` (*object*) [Generic Writer]

Package [`distributions`], page 17.

Methods

`state` ((*generator [*`generator`*], page 42*)) [Reader Method]
`(setf state)` ((*generator [*`generator`*], page 42*)) [Writer Method]
All information needed by the generator to create the next chunk of random bits. This state is modified after each call to NEXT-CHUNK.

Source [`generator.lisp`], page 5.

Target Slot
[`state`], page 42.

4.2.6 Structures

`left-truncated-normal` [Structure]
Truncated normal distribution with given mu and sigma (corresponds to the mean and standard deviation in the untruncated case, respectively), on the interval [left, infinity).

Package [`distributions`], page 17.

Source [`truncated-normal.lisp`], page 11.

Direct superclasses
[`r-univariate`], page 60.

Direct methods
- [`cdf`], page 28.
- [`draw`], page 29.
- [`log-pdf`], page 30.
- [`mean`], page 31.
- [`quantile`], page 32.
- [`variance`], page 33.

Direct slots

`mu` [Slot]
Type `distributions.internals:internal-float`
Readers [`left-truncated-normal-mu`], page 48.
Writers [`(setf left-truncated-normal-mu)`], page 48.

`sigma` [Slot]
Type `distributions.internals:internal-float`
Readers [`left-truncated-normal-sigma`], page 48.
Writers [`(setf left-truncated-normal-sigma)`], page 48.

`left` [Slot]
Type `distributions.internals:internal-float`
Readers [`left-truncated-normal-left`], page 48.
Writers [`(setf left-truncated-normal-left)`], page 48.

`left-standardized` [Slot]

Type `distributions.internals:internal-float`

Readers [`left-truncated-normal-left-standardized`], page 48.

Writers [`(setf left-truncated-normal-left-standardized)`], page 48.

`m0` [Slot]

Type `distributions.internals:internal-float`

Readers [`left-truncated-normal-m0`], page 48.

Writers [`(setf left-truncated-normal-m0)`], page 48.

`alpha` [Slot]

Type `distributions.internals:internal-float`

Readers [`left-truncated-normal-alpha`], page 48.

Writers [`(setf left-truncated-normal-alpha)`], page 48.

`r-univariate` [Structure]

Univariate distribution.

Package [`distributions`], page 17.

Source [`generics.lisp`], page 7.

Direct superclasses

`structure-object`.

Direct subclasses

- [`left-truncated-normal`], page 59.
- [`r-bernoulli`], page 34.
- [`r-beta`], page 35.
- [`r-binomial`], page 35.
- [`r-discrete`], page 36.
- [`r-exponential`], page 36.
- [`r-gamma`], page 37.
- [`r-geometric`], page 38.
- [`r-inverse-gamma`], page 38.
- [`r-log-normal`], page 39.
- [`r-normal`], page 39.
- [`r-rayleigh`], page 40.
- [`r-t`], page 40.
- [`r-uniform`], page 41.

Direct methods

[`standard-deviation`], page 33.

4.2.7 Classes

`simple-multiplicative-congruential` [Class]

A multiplicative congruential generator generates the sequence x(n+1) = A * x(n) mod M and uses the seed as x(1). A simple multiplicative congruential generator is a multiplicative congruential generator with M a power of 2. This allows to implement the modulo operation as a bitwise and operation of M-1, which is also the maximum value of a random chunk.

Package [`distributions`], page 17.

Source [`simple-multiplicative-congruential-generators.lisp`], page 6.

Direct superclasses

[`generator`], page 42.

Direct subclasses

- [`borosh13`], page 42.
- [`randu`], page 43.
- [`transputer`], page 43.
- [`waterman14`], page 44.

Direct methods

- [`a`], page 56.
- [`clone`], page 56.
- [`generate-state`], page 57.
- [`initialize-instance`], page 34.
- [`m`], page 57.
- [`next-chunk`], page 58.
- [`next-real`], page 58.

Direct slots

`default-seed` [Slot]

Initform `1`

`a` [Slot]

The multiplier of the sequence x(n+1) = A * x(n) mod M.

Readers [`a`], page 56.

Writers *This slot is read-only.*

`m` [Slot]

The modulo of the sequence x(n+1) = A * x(n) mod M.

Readers [`m`], page 57.

Writers *This slot is read-only.*

Appendix A Indexes

A.1 Concepts

(Index is nonexistent)

A.2 Functions

%

(

A

B

C

D

F

G

I

L

M

N

P

Q

R

S

T

V

W

A.3 Variables

*

+

A

B

C

D

L

M

N

P

R

S

W

A.4 Data types

B

C

D

E

F

G

I

L

N

P

R

S

T

U

W