# Probability

***Release 10.2***

**The Sage Development Team**

**Dec 06, 2023**

# CONTENTS

1 Probability Distributions 1

2 Random variables and probability spaces 13

3 Indices and Tables 17

Python Module Index 19

Index 21

CHAPTER

ONE

# PROBABILITY DISTRIBUTIONS

This module provides three types of probability distributions:

- *RealDistribution*: various real-valued probability distributions.
- *SphericalDistribution*: uniformly distributed points on the surface of an $n-1$ sphere in $n$ dimensional euclidean space.
- *GeneralDiscreteDistribution*: user-defined discrete distributions.

AUTHORS:

- Josh Kantor (2007-02): first version
- William Stein (2007-02): rewrite of docs, conventions, etc.
- Carlo Hamalainen (2008-08): full doctest coverage, more documentation, GeneralDiscreteDistribution, misc fixes.
- Kwankyu Lee (2010-05-29): F-distribution support.

REFERENCES:

GNU gsl library, General discrete distributions http://www.gnu.org/software/gsl/manual/html_node/General-Discrete-Distributions.html

GNU gsl library, Random number distributions http://www.gnu.org/software/gsl/manual/html_node/Random-Number-Distributions.html

**class** sage.probability.probability_distribution.**GeneralDiscreteDistribution**

Bases: *ProbabilityDistribution*

Create a discrete probability distribution.

INPUT:

- `P` – list of probabilities. The list will automatically be normalised if `sum(P)` is not equal to 1.
- `rng` – (optional) random number generator to use. May be one of `'default'`, `'luxury'`, or `'taus'`.
- `seed` – (optional) seed to use with the random number generator.

OUTPUT:

- a probability distribution where the probability of selecting `x` is `P[x]`.

EXAMPLES:

Construct a `GeneralDiscreteDistribution` with the probability distribution $P$ where $P(0) = 0.3$, $P(1) = 0.4$, $P(2) = 0.3$:

```
sage: P = [0.3, 0.4, 0.3]
sage: X = GeneralDiscreteDistribution(P)
sage: X.get_random_element() in (0, 1, 2)
True
```

Checking the distribution of samples:

```
sage: P = [0.3, 0.4, 0.3]
sage: counts = [0] * len(P)
sage: X = GeneralDiscreteDistribution(P)
sage: nr_samples = 10000
sage: for _ in range(nr_samples):
....:     counts[X.get_random_element()] += 1
sage: [1.0*x/nr_samples for x in counts]  # abs tol 3e-2
[0.3, 0.4, 0.3]
```

The distribution probabilities will automatically be normalised:

```
sage: P = [0.1, 0.3]
sage: X = GeneralDiscreteDistribution(P, seed=0)
sage: counts = [0, 0]
sage: for _ in range(10000):
....:     counts[X.get_random_element()] += 1
sage: float(counts[1]/counts[0])
3.042037186742118
```

**get_random_element**()

Get a random sample from the probability distribution.

EXAMPLES:

```
sage: P = [0.3, 0.4, 0.3]
sage: X = GeneralDiscreteDistribution(P)
sage: all(X.get_random_element() in (0,1,2) for _ in range(10))
True
sage: isinstance(X.get_random_element(), sage.rings.integer.Integer)
True
```

**reset_distribution**()

This method resets the distribution.

EXAMPLES:

```
sage: T = GeneralDiscreteDistribution([0.1, 0.3, 0.6])
sage: T.set_seed(0)
sage: [T.get_random_element() for _ in range(10)]
[2, 2, 2, 2, 2, 1, 2, 2, 1, 2]
sage: T.reset_distribution()
sage: [T.get_random_element() for _ in range(10)]
[2, 2, 2, 2, 2, 1, 2, 2, 1, 2]
```

**set_random_number_generator**(*rng='default'*)

Set the random number generator to be used by gsl.

EXAMPLES:

```
sage: X = GeneralDiscreteDistribution([0.3, 0.4, 0.3])
sage: X.set_random_number_generator('taus')
```

**set_seed**(*seed*)

Set the seed to be used by the random number generator.

EXAMPLES:

```
sage: X = GeneralDiscreteDistribution([0.3, 0.4, 0.3])
sage: X.set_seed(1)
sage: X.get_random_element()
1
```

**class** sage.probability.probability_distribution.**ProbabilityDistribution**

Bases: `object`

Concrete probability distributions should be derived from this abstract class.

**generate_histogram_data**(*num_samples=1000*, *bins=50*)

Compute a histogram of the probability distribution.

INPUT:

- `num_samples` – (optional) number of times to sample from the probability distribution
- `bins` – (optional) number of bins to divide the samples into.

OUTPUT:

- a tuple. The first element of the tuple is a list of length `bins`, consisting of the normalised histogram of the random samples. The second list is the bins.

EXAMPLES:

```
sage: set_random_seed(0)
sage: from sage.probability.probability_distribution import␣
↪GeneralDiscreteDistribution
sage: P = [0.3, 0.4, 0.3]
sage: X = GeneralDiscreteDistribution(P)
sage: h, b = X.generate_histogram_data(bins=10)                    #␣
↪needs sage.plot
sage: h  # rel tol 1e-08                                           #␣
↪needs sage.plot
[1.6299999999999999,
 0.0,
 0.0,
 0.0,
 0.0,
 1.9049999999999985,
 0.0,
 0.0,
 0.0,
 1.4650000000000003]
sage: b                                                            #␣
↪needs sage.plot
[0.0,
 0.2,
```

(continues on next page)

(continued from previous page)

```
 0.4,
 0.6000000000000001,
 0.8,
 1.0,
 1.2000000000000002,
 1.4000000000000001,
 1.6,
 1.8,
 2.0]
```

**generate_histogram_plot**(*name*, *num_samples=1000*, *bins=50*)

Save the histogram from `generate_histogram_data()` to a file.

INPUT:

- `name` – file to save the histogram plot (as a PNG).
- `num_samples` – (optional) number of times to sample from the probability distribution
- `bins` – (optional) number of bins to divide the samples into.

EXAMPLES:

This saves the histogram plot to a temporary file:

```
sage: from sage.probability.probability_distribution import␣
↪GeneralDiscreteDistribution
sage: import tempfile
sage: P = [0.3, 0.4, 0.3]
sage: X = GeneralDiscreteDistribution(P)
sage: with tempfile.NamedTemporaryFile() as f:                    #␣
↪needs sage.plot
....:     X.generate_histogram_plot(f.name)
```

**get_random_element**()

To be implemented by a derived class:

```
sage: P = sage.probability.probability_distribution.ProbabilityDistribution()
sage: P.get_random_element()
Traceback (most recent call last):
...
NotImplementedError: implement in derived class
```

**class** sage.probability.probability_distribution.**RealDistribution**

Bases: *ProbabilityDistribution*

The *RealDistribution* class provides a number of routines for sampling from and analyzing and visualizing probability distributions. For precise definitions of the distributions and their parameters see the gsl reference manuals chapter on random number generators and probability distributions.

EXAMPLES:

Uniform distribution on the interval `[a, b]`:

```
sage: a = 0
sage: b = 2
```

(continues on next page)

(continued from previous page)

```
sage: T = RealDistribution('uniform', [a, b])
sage: a <= T.get_random_element() <= b
True
sage: T.distribution_function(0)
0.5
sage: T.cum_distribution_function(1)
0.5
sage: T.cum_distribution_function_inv(.5)
1.0
```

The gaussian distribution takes 1 parameter `sigma`. The standard gaussian distribution has `sigma = 1`:

```
sage: sigma = 1
sage: T = RealDistribution('gaussian', sigma)
sage: s = T.get_random_element()
sage: s.parent()
Real Double Field
sage: T.distribution_function(0)
0.3989422804014327
sage: T.cum_distribution_function(1)
0.8413447460685429
sage: T.cum_distribution_function_inv(.5)
0.0
```

The rayleigh distribution has 1 parameter `sigma`:

```
sage: sigma = 3
sage: T = RealDistribution('rayleigh', sigma)
sage: s = T.get_random_element()
sage: s >= 0
True
sage: s.parent()
Real Double Field
sage: T.distribution_function(0)
0.0
sage: T.cum_distribution_function(1)
0.054040531093234534
sage: T.cum_distribution_function_inv(.5)
3.532230067546424...
```

The lognormal distribution has two parameters `sigma` and `zeta`:

```
sage: zeta = 0
sage: sigma = 1
sage: T = RealDistribution('lognormal', [zeta, sigma])
sage: s = T.get_random_element()
sage: s >= 0
True
sage: s.parent()
Real Double Field
sage: T.distribution_function(0)
0.0
sage: T.cum_distribution_function(1)
```

(continues on next page)

(continued from previous page)

```
0.5
sage: T.cum_distribution_function_inv(.5)
1.0
```

The pareto distribution has two parameters a, and b:

```
sage: a = 1
sage: b = 1
sage: T = RealDistribution('pareto', [a, b])
sage: s = T.get_random_element()
sage: s >= b
True
sage: s.parent()
Real Double Field
sage: T.distribution_function(0)
0.0
sage: T.cum_distribution_function(1)
0.0
sage: T.cum_distribution_function_inv(.5)
2.0
```

The t-distribution has one parameter nu:

```
sage: nu = 1
sage: T = RealDistribution('t', nu)
sage: s = T.get_random_element()
sage: s.parent()
Real Double Field
sage: T.distribution_function(0)      # rel tol 1e-15
0.3183098861837906
sage: T.cum_distribution_function(1)  # rel tol 1e-15
0.75
sage: T.cum_distribution_function_inv(.5)
0.0
```

The F-distribution has two parameters nu1 and nu2:

```
sage: nu1 = 9; nu2 = 17
sage: F = RealDistribution('F', [nu1,nu2])
sage: s = F.get_random_element()
sage: s >= 0
True
sage: s.parent()
Real Double Field
sage: F.distribution_function(1)  # rel tol 1e-14
0.6695025505192798
sage: F.cum_distribution_function(3.68)  # rel tol 1e-14
0.9899717772300652
sage: F.cum_distribution_function_inv(0.99)  # rel tol 1e-14
3.682241524045864
```

The chi-squared distribution has one parameter nu:

```
sage: nu = 1
sage: T = RealDistribution('chisquared', nu)
sage: s = T.get_random_element()
sage: s >= 0
True
sage: s.parent()
Real Double Field
sage: T.distribution_function(0)
+infinity
sage: T.cum_distribution_function(1)  # rel tol 1e-14
0.6826894921370856
sage: T.cum_distribution_function_inv(.5)  # rel tol 1e-14
0.45493642311957305
```

The exponential power distribution has two parameters a and b:

```
sage: a = 1
sage: b = 2.5
sage: T = RealDistribution('exppow', [a, b])
sage: s = T.get_random_element()
sage: s.parent()
Real Double Field
sage: T.distribution_function(0)  # rel tol 1e-14
0.5635302489930136
sage: T.cum_distribution_function(1)  # rel tol 1e-14
0.940263052542855
```

The beta distribution has two parameters a and b:

```
sage: a = 2
sage: b = 2
sage: T = RealDistribution('beta', [a, b])
sage: s = T.get_random_element()
sage: 0 <= s <= 1
True
sage: s.parent()
Real Double Field
sage: T.distribution_function(0)
0.0
sage: T.cum_distribution_function(1)
1.0
```

The exponential distribution has one parameter mu:

```
sage: mu = 2
sage: T = RealDistribution('exponential', mu)
sage: s = T.get_random_element()
sage: 0 <= s
True
sage: s.parent()
Real Double Field
sage: T.distribution_function(0)
0.5
```

The gamma distribution has two parameters a and b:

```
sage: a = 2
sage: b = 2
sage: T = RealDistribution('gamma', [a, b])
sage: s = T.get_random_element()
sage: 0 <= s
True
sage: s.parent()
Real Double Field
sage: T.distribution_function(0)
0.0
```

The weibull distribution has two parameters a and b:

```
sage: a = 1
sage: b = 1
sage: T = RealDistribution('weibull', [a, b])
sage: s = T.get_random_element()
sage: s >= 0
True
sage: s.parent()
Real Double Field
sage: T.distribution_function(0)
1.0
sage: T.cum_distribution_function(1)
0.6321205588285577
sage: T.cum_distribution_function_inv(.5)
0.6931471805599453
```

It is possible to select which random number generator drives the sampling as well as the seed. The default is the Mersenne twister. Also available are the RANDLXS algorithm and the Tausworthe generator (see the gsl reference manual for more details). These are all supposed to be simulation quality generators. For RANDLXS use `rng='luxury'` and for tausworth use `rng='taus'`:

```
sage: T = RealDistribution('gaussian', 1, rng='luxury', seed=10)
```

To change the seed at a later time use `set_seed`:

```
sage: T.set_seed(100)
```

**cum_distribution_function**($x$)

Evaluate the cumulative distribution function of the probability distribution at x.

EXAMPLES:

```
sage: T = RealDistribution('uniform', [0, 2])
sage: T.cum_distribution_function(1)
0.5
```

**cum_distribution_function_inv**($x$)

Evaluate the inverse of the cumulative distribution distribution function of the probability distribution at x.

EXAMPLES:

```
sage: T = RealDistribution('uniform', [0, 2])
sage: T.cum_distribution_function_inv(.5)
1.0
```

**distribution_function**(*x*)

Evaluate the distribution function of the probability distribution at `x`.

EXAMPLES:

```
sage: T = RealDistribution('uniform', [0, 2])
sage: T.distribution_function(0)
0.5
sage: T.distribution_function(1)
0.5
sage: T.distribution_function(1.5)
0.5
sage: T.distribution_function(2)
0.0
```

**get_random_element**()

Get a random sample from the probability distribution.

EXAMPLES:

```
sage: T = RealDistribution('gaussian', 1, seed=0)
sage: T.get_random_element()  # rel tol 4e-16
0.13391860811867587
```

**plot**(**args*, ***kwds*)

Plot the distribution function for the probability distribution. Parameters to `sage.plot.plot.plot()` can be passed through `*args` and `**kwds`.

EXAMPLES:

```
sage: T = RealDistribution('uniform', [0, 2])
sage: P = T.plot()                                                          #␣
↪needs sage.plot
```

**reset_distribution**()

Reset the distribution.

EXAMPLES:

```
sage: T = RealDistribution('gaussian', 1, seed=10)
sage: [T.get_random_element() for _ in range(10)]  # rel tol 4e-16
[-0.7460999595745819, -0.004644606626413462, -0.8720538317207641, 0.
↪691625992166037, 2.67668674666043, 0.6325002813661014, -0.7974263521959355, -
↪0.5284976893366636, 1.1353119849528792, 0.9912505673230749]
sage: T.reset_distribution()
sage: [T.get_random_element() for _ in range(10)]  # rel tol 4e-16
[-0.7460999595745819, -0.004644606626413462, -0.8720538317207641, 0.
↪691625992166037, 2.67668674666043, 0.6325002813661014, -0.7974263521959355, -
↪0.5284976893366636, 1.1353119849528792, 0.9912505673230749]
```

**set_distribution**(*name='uniform'*, *parameters=None*)

This method can be called to change the current probability distribution.

EXAMPLES:

```
sage: T = RealDistribution('gaussian', 1)
sage: T.set_distribution('gaussian', 1)
sage: T.set_distribution('pareto', [0, 1])
```

**set_random_number_generator**(*rng='default'*)

Set the gsl random number generator to be one of `'default'`, `'luxury'`, or `'taus'`.

EXAMPLES:

```
sage: T = SphericalDistribution()
sage: T.set_random_number_generator('default')
sage: T.set_seed(0)
sage: T.get_random_element()  # rel tol 4e-16
(0.07961564104639995, -0.05237671627581255, 0.9954486572862178)
sage: T.set_random_number_generator('luxury')
sage: T.set_seed(0)
sage: T.get_random_element()  # rel tol 4e-16
(0.07961564104639995, -0.05237671627581255, 0.9954486572862178)
```

**set_seed**(*seed*)

Set the seed for the underlying random number generator.

EXAMPLES:

```
sage: T = RealDistribution('gaussian', 1, rng='luxury', seed=10)
sage: T.set_seed(100)
```

**class** sage.probability.probability_distribution.**SphericalDistribution**

Bases: *ProbabilityDistribution*

This class is capable of producing random points uniformly distributed on the surface of an $(n-1)$-sphere in $n$-dimensional euclidean space. The dimension $n$ is selected via the keyword `dimension`. The random number generator which drives it can be selected using the keyword `rng`. Valid choices are `'default'` which uses the Mersenne-Twister, `'luxury'` which uses RANDLXS, and `'taus'` which uses the tausworth generator. The default dimension is 3.

EXAMPLES:

```
sage: T = SphericalDistribution()
sage: s = T.get_random_element()
sage: s.norm()  # rel tol 1e-14
1.0
sage: len(s)
3
sage: T = SphericalDistribution(dimension=4, rng='luxury')
sage: s = T.get_random_element()
sage: s.norm()  # rel tol 1e-14
1.0
sage: len(s)
4
```

**get_random_element**()

Get a random sample from the probability distribution.

EXAMPLES:

```
sage: T = SphericalDistribution(seed=0)
sage: T.get_random_element()  # rel tol 4e-16
(0.07961564104639995, -0.05237671627581255, 0.9954486572862178)
```

**reset_distribution**()

This method resets the distribution.

EXAMPLES:

```
sage: T = SphericalDistribution(seed=0)
sage: [T.get_random_element() for _ in range(4)]  # rel tol 4e-16
[(0.07961564104639995, -0.05237671627581255, 0.9954486572862178),
 (0.4123599490593727, 0.5606817859360097, -0.7180495855658982),
 (-0.9619860891623148, -0.2726473494040498, -0.015690351211529927),
 (0.5674297579435619, -0.011206783800420301, -0.8233455397322326)]
sage: T.reset_distribution()
sage: [T.get_random_element() for _ in range(4)]  # rel tol 4e-16
[(0.07961564104639995, -0.05237671627581255, 0.9954486572862178),
 (0.4123599490593727, 0.5606817859360097, -0.7180495855658982),
 (-0.9619860891623148, -0.2726473494040498, -0.015690351211529927),
 (0.5674297579435619, -0.011206783800420301, -0.8233455397322326)]
```

**set_random_number_generator**(*rng='default'*)

Set the gsl random number generator to be one of `default`, `luxury`, or `taus`.

EXAMPLES:

```
sage: T = SphericalDistribution()
sage: T.set_random_number_generator('default')
sage: T.set_seed(0)
sage: T.get_random_element()  # rel tol 4e-16
(0.07961564104639995, -0.05237671627581255, 0.9954486572862178)
sage: T.set_random_number_generator('luxury')
sage: T.set_seed(0)
sage: T.get_random_element()  # rel tol 4e-16
(0.07961564104639995, -0.05237671627581255, 0.9954486572862178)
```

**set_seed**(*seed*)

Set the seed for the underlying random number generator.

EXAMPLES:

```
sage: T = SphericalDistribution(seed=0)
sage: T.set_seed(100)
```

CHAPTER

TWO

# RANDOM VARIABLES AND PROBABILITY SPACES

This introduces a class of random variables, with the focus on discrete random variables (i.e. on a discrete probability space). This avoids the problem of defining a measure space and measurable functions.

**class** sage.probability.random_variable.**DiscreteProbabilitySpace**(*X*, *P*, *codomain=None*, *check=False*)

Bases: *ProbabilitySpace_generic*, *DiscreteRandomVariable*

The discrete probability space

**entropy**()

The entropy of the probability space.

**set**()

The set of values of the probability space taking possibly nonzero probability (a subset of the domain).

**class** sage.probability.random_variable.**DiscreteRandomVariable**(*X*, *f*, *codomain=None*, *check=False*)

Bases: *RandomVariable_generic*

A random variable on a discrete probability space.

**correlation**(*other*)

The correlation of the probability space X = self with Y = other.

**covariance**(*other*)

The covariance of the discrete random variable X = self with Y = other.

Let $S$ be the probability space of $X$ = self, with probability function $p$, and $E(X)$ be the expectation of $X$. Then the variance of $X$ is:

$$\text{cov}(X,Y) = E((X - E(X) \cdot (Y - E(Y)) = \sum_{x \in S} p(x)(X(x) - E(X))(Y(x) - E(Y))$$

**expectation**()

The expectation of the discrete random variable, namely $\sum_{x \in S} p(x)X[x]$, where $X$ = self and $S$ is the probability space of $X$.

**function**()

The function defining the random variable.

**standard_deviation**()

The standard deviation of the discrete random variable.

Let $S$ be the probability space of $X$ = self, with probability function $p$, and $E(X)$ be the expectation of $X$. Then the standard deviation of $X$ is defined to be

$$\sigma(X) = \sqrt{\sum_{x \in S} p(x)(X(x) - E(x))^2}$$

**translation_correlation**(*other*, *map*)

The correlation of the probability space X = self with image of Y = other under map.

**translation_covariance**(*other*, *map*)

The covariance of the probability space X = self with image of Y = other under the given map of the probability space.

Let $S$ be the probability space of $X$ = self, with probability function $p$, and $E(X)$ be the expectation of $X$. Then the variance of $X$ is:

$$\text{cov}(X, Y) = E((X - E(X) \cdot (Y - E(Y)) = \sum_{x \in S} p(x)(X(x) - E(X))(Y(x) - E(Y))$$

**translation_expectation**(*map*)

The expectation of the discrete random variable, namely $\sum_{x \in S} p(x)X[e(x)]$, where $X$ = self, $S$ is the probability space of $X$, and $e$ = map.

**translation_standard_deviation**(*map*)

The standard deviation of the translated discrete random variable $X \circ e$, where $X$ = self and $e$ = map.

Let $S$ be the probability space of $X$ = self, with probability function $p$, and $E(X)$ be the expectation of $X$. Then the standard deviation of $X$ is defined to be

$$\sigma(X) = \sqrt{\sum_{x \in S} p(x)(X(x) - E(x))^2}$$

**translation_variance**(*map*)

The variance of the discrete random variable $X \circ e$, where $X$ = self, and $e$ = map.

Let $S$ be the probability space of $X$ = self, with probability function $p$, and $E(X)$ be the expectation of $X$. Then the variance of $X$ is:

$$\text{var}(X) = E((X - E(x))^2) = \sum_{x \in S} p(x)(X(x) - E(x))^2$$

**variance**()

The variance of the discrete random variable.

Let $S$ be the probability space of $X$ = self, with probability function $p$, and $E(X)$ be the expectation of $X$. Then the variance of $X$ is:

$$\text{var}(X) = E((X - E(x))^2) = \sum_{x \in S} p(x)(X(x) - E(x))^2$$

**class** sage.probability.random_variable.**ProbabilitySpace_generic**(*domain*, *RR*)

Bases: *RandomVariable_generic*

A probability space.

**domain**()

**class** sage.probability.random_variable.**RandomVariable_generic**(*X*, *RR*)

Bases: Parent

A random variable.

**codomain**()

**domain**()

**field**()

**probability_space**()

sage.probability.random_variable.**is_DiscreteProbabilitySpace**(*S*)

sage.probability.random_variable.**is_DiscreteRandomVariable**(*X*)

sage.probability.random_variable.**is_ProbabilitySpace**(*S*)

sage.probability.random_variable.**is_RandomVariable**(*X*)

CHAPTER

THREE

# INDICES AND TABLES

- Index
- Module Index
- Search Page

# PYTHON MODULE INDEX

## p

sage.probability.probability_distribution, 1
sage.probability.random_variable, 13

# INDEX


## C

codomain() (*sage.probability.random_variable.RandomVariable_generic method*), 15
correlation() (*sage.probability.random_variable.DiscreteRandomVariable method*), 13
covariance() (*sage.probability.random_variable.DiscreteRandomVariable method*), 13
cum_distribution_function() (*sage.probability.probability_distribution.RealDistribution method*), 8
cum_distribution_function_inv() (*sage.probability.probability_distribution.RealDistribution method*), 8

## D

DiscreteProbabilitySpace (*class in sage.probability.random_variable*), 13
DiscreteRandomVariable (*class in sage.probability.random_variable*), 13
distribution_function() (*sage.probability.probability_distribution.RealDistribution method*), 9
domain() (*sage.probability.random_variable.ProbabilitySpace_generic method*), 14
domain() (*sage.probability.random_variable.RandomVariable_generic method*), 15

## E

entropy() (*sage.probability.random_variable.DiscreteProbabilitySpace method*), 13
expectation() (*sage.probability.random_variable.DiscreteRandomVariable method*), 13

## F

field() (*sage.probability.random_variable.RandomVariable_generic method*), 15
function() (*sage.probability.random_variable.DiscreteRandomVariable method*), 13

## G

GeneralDiscreteDistribution (*class in sage.probability.probability_distribution*), 1
generate_histogram_data() (*sage.probability.probability_distribution.ProbabilityDistribution method*), 3
generate_histogram_plot() (*sage.probability.probability_distribution.ProbabilityDistribution method*), 4
get_random_element() (*sage.probability.probability_distribution.GeneralDiscreteDistrib method*), 2
get_random_element() (*sage.probability.probability_distribution.ProbabilityDistribution method*), 4
get_random_element() (*sage.probability.probability_distribution.RealDistribution method*), 9
get_random_element() (*sage.probability.probability_distribution.SphericalDistribution method*), 10

## I

is_DiscreteProbabilitySpace() (*in module sage.probability.random_variable*), 15
is_DiscreteRandomVariable() (*in module sage.probability.random_variable*), 15
is_ProbabilitySpace() (*in module sage.probability.random_variable*), 15
is_RandomVariable() (*in module sage.probability.random_variable*), 15

## M

module
    sage.probability.probability_distribution, 1
    sage.probability.random_variable, 13

## P

plot() (*sage.probability.probability_distribution.RealDistribution method*), 9
probability_space() (*sage.probability.random_variable.RandomVariable_generic*

*method*), 15
ProbabilityDistribution (*class in sage.probability.probability_distribution*), 3
ProbabilitySpace_generic (*class in sage.probability.random_variable*), 14

## R

RandomVariable_generic (*class in sage.probability.random_variable*), 15
RealDistribution (*class in sage.probability.probability_distribution*), 4
reset_distribution() (*sage.probability.probability_distribution.GeneralDiscreteDistribution method*), 2
reset_distribution() (*sage.probability.probability_distribution.RealDistribution method*), 9
reset_distribution() (*sage.probability.probability_distribution.SphericalDistribution method*), 11

## S

sage.probability.probability_distribution module, 1
sage.probability.random_variable module, 13
set() (*sage.probability.random_variable.DiscreteProbabilitySpace method*), 13
set_distribution() (*sage.probability.probability_distribution.RealDistribution method*), 9
set_random_number_generator() (*sage.probability.probability_distribution.GeneralDiscreteDistribution method*), 2
set_random_number_generator() (*sage.probability.probability_distribution.RealDistribution method*), 10
set_random_number_generator() (*sage.probability.probability_distribution.SphericalDistribution method*), 11
set_seed() (*sage.probability.probability_distribution.GeneralDiscreteDistribution method*), 3
set_seed() (*sage.probability.probability_distribution.RealDistribution method*), 10
set_seed() (*sage.probability.probability_distribution.SphericalDistribution method*), 11
SphericalDistribution (*class in sage.probability.probability_distribution*), 10
standard_deviation() (*sage.probability.random_variable.DiscreteRandomVariable method*), 13

## T

translation_correlation() (*sage.probability.random_variable.DiscreteRandomVariable method*), 14
translation_covariance() (*sage.probability.random_variable.DiscreteRandomVariable method*), 14
translation_expectation() (*sage.probability.random_variable.DiscreteRandomVariable method*), 14
translation_standard_deviation() (*sage.probability.random_variable.DiscreteRandomVariable method*), 14
translation_variance() (*sage.probability.random_variable.DiscreteRandomVariable method*), 14

## V

variance() (*sage.probability.random_variable.DiscreteRandomVariable method*), 14