Sage Reference Manual: Probability

Release 6.3

The Sage Development Team

August 11, 2014

CONTENTS

CHAPTER

ONE

PROBABILITY DISTRIBUTIONS

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.gsl.probability_distribution.`**`GeneralDiscreteDistribution`**

Bases: `sage.gsl.probability_distribution.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:

Constructs 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()  # random
2
```

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]  # random
[0.295400000000000, 0.400200000000000, 0.304400000000000]
```

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
```

TESTS:

Make sure that repeated initializations are randomly seeded (trac ticket #9770):

```
sage: P = [0.001] * 1000
sage: Xs = [GeneralDiscreteDistribution(P).get_random_element() for _ in range(1000)]
sage: len(set(Xs)) > 2^^32
True
```

The distribution probabilities must be non-negative:

```
sage: GeneralDiscreteDistribution([0.1, -0.1])
Traceback (most recent call last):
...
ValueError: The distribution probabilities must be non-negative
```

get_random_element()

Get a random sample from the probability distribution.

EXAMPLE:

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

reset_distribution()

This method resets the distribution.

EXAMPLE:

```
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.

EXAMPLE:

```
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.

EXAMPLE:

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

class sage.gsl.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.

EXAMPLE:

```
sage: from sage.gsl.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)
```

```
sage: h # random
[1.5249999999999995, 0.0, 0.0, 0.0, 0.0, 1.8649999999999995, 0.0, 0.0, 0.0, 1.60999999999999
sage: b # random
[0.0, 0.20000000000000001, 0.40000000000000002, 0.60000000000000009, 0.80000000000000004, 1.
```

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.

EXAMPLE:

This saves the histogram plot to `my_general_distribution_plot.png` in the temporary directory `SAGE_TMP`:

```
sage: from sage.gsl.probability_distribution import GeneralDiscreteDistribution
sage: import os
sage: P = [0.3, 0.4, 0.3]
sage: X = GeneralDiscreteDistribution(P)
sage: file = os.path.join(SAGE_TMP, "my_general_distribution_plot")
sage: X.generate_histogram_plot(file)
```

get_random_element ()

To be implemented by a derived class:

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

class sage.gsl.probability_distribution.**RealDistribution**

Bases: `sage.gsl.probability_distribution.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
sage: T = RealDistribution('uniform', [a, b])
sage: T.get_random_element() # random
0.416921074037
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: T.get_random_element() # random
0.818610064197
sage: T.distribution_function(0)
0.398942280401
sage: T.cum_distribution_function(1)
0.841344746069
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: T.get_random_element() # random
1.65471291529
sage: T.distribution_function(0)
0.0
sage: T.cum_distribution_function(1)
0.0540405310932
sage: T.cum_distribution_function_inv(.5)
3.53223006755
```

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

```
sage: zeta = 0
sage: sigma = 1
sage: T = RealDistribution('lognormal', [zeta, sigma])
sage: T.get_random_element() # random
1.23541716538
sage: T.distribution_function(0)
0.0
sage: T.cum_distribution_function(1)
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: T.get_random_element() # random
1.16429443511
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: T.get_random_element() # random
-0.994514581164
sage: T.distribution_function(0)
0.318309886184
```

```
sage: T.cum_distribution_function(1)
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: F.get_random_element() # random
1.65211335491
sage: F.distribution_function(1)
0.669502550519
sage: F.cum_distribution_function(3.68)
0.98997177723
sage: F.cum_distribution_function_inv(0.99)
3.68224152405
```

The chi-squared distribution has one parameter nu:

```
sage: nu = 1
sage: T = RealDistribution('chisquared', nu)
sage: T.get_random_element() # random
0.103230507883
sage: T.distribution_function(0)
+infinity
sage: T.cum_distribution_function(1)
0.682689492137
sage: T.cum_distribution_function_inv(.5)
0.45493642312
```

The exponential power distribution has two parameters a and b:

```
sage: a = 1
sage: b = 2.5
sage: T = RealDistribution('exppow', [a, b])
sage: T.get_random_element() # random
0.570108609774
sage: T.distribution_function(0)
0.563530248993
sage: T.cum_distribution_function(1)
0.940263052543
```

The beta distribution has two parameters a and b:

```
sage: a = 2
sage: b = 2
sage: T = RealDistribution('beta', [a, b])
sage: T.get_random_element() # random
0.518139435862
sage: T.distribution_function(0)
0.0
sage: T.cum_distribution_function(1)
1.0
```

The weibull distribution has two parameters a and b:

```
sage: a = 1
sage: b = 1
sage: T = RealDistribution('weibull', [a, b])
```

```
sage: T.get_random_element() # random
1.86974582214
sage: T.distribution_function(0)
1.0
sage: T.cum_distribution_function(1)
0.632120558829
sage: T.cum_distribution_function_inv(.5)
0.69314718056
```

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)
```

TESTS:

Make sure that repeated initializations are randomly seeded (trac ticket #9770):

```
sage: Xs = [RealDistribution('gaussian', 1).get_random_element() for _ in range(1000)]
sage: len(set(Xs)) > 2^^32
True
```

cum_distribution_function(*x*)

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

EXAMPLE:

```
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`.

EXAMPLE:

```
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.

EXAMPLE:

```
sage: T = RealDistribution('gaussian', 1, seed = 0)
sage: T.get_random_element()
0.133918608119
```

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

EXAMPLE:

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

reset_distribution()
This method resets the distribution.

EXAMPLE:

```
sage: T = RealDistribution('gaussian', 1, seed = 10)
sage: [T.get_random_element() for _ in range(10)]
[-0.746099959575, -0.00464460662641, -0.872053831721,
 0.691625992167, 2.67668674666, 0.632500281366,
 -0.797426352196, -0.528497689337, 1.13531198495,
 0.991250567323]
sage: T.reset_distribution()
sage: [T.get_random_element() for _ in range(10)]
[-0.746099959575, -0.00464460662641, -0.872053831721,
 0.691625992167, 2.67668674666, 0.632500281366,
 -0.797426352196, -0.528497689337, 1.13531198495,
 0.991250567323]
```

set_distribution(*name='uniform'*, *parameters=[ ]*)
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`.

EXAMPLE:

```
sage: T = SphericalDistribution()
sage: T.set_random_number_generator('default')
sage: T.set_seed(0)
sage: T.get_random_element()
(0.0796156410464, -0.0523767162758, 0.995448657286)
sage: T.set_random_number_generator('luxury')
sage: T.set_seed(0)
sage: T.get_random_element()
(0.0796156410464, -0.0523767162758, 0.995448657286)
```

set_seed(*seed*)
Set the seed for the underlying random number generator.

EXAMPLE:

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

class sage.gsl.probability_distribution.**SphericalDistribution**

Bases: sage.gsl.probability_distribution.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: T.get_random_element() # random
(-0.872578667429, -0.29632873418, -0.388324285164)
sage: T = SphericalDistribution(dimension = 4, rng = 'luxury')
sage: T.get_random_element() # random
(-0.196597969334, -0.536955365418, -0.672242159448, -0.470232552109)
```

TESTS:

Make sure that repeated initializations are randomly seeded (trac ticket #9770):

```
sage: Xs = [tuple(SphericalDistribution(2).get_random_element()) for _ in range(1000)]
sage: len(set(Xs)) > 2^^32
True
```

get_random_element()

Get a random sample from the probability distribution.

EXAMPLE:

```
sage: T = SphericalDistribution(seed = 0)
sage: T.get_random_element()
(0.0796156410464, -0.0523767162758, 0.995448657286)
```

reset_distribution()

This method resets the distribution.

EXAMPLE:

```
sage: T = SphericalDistribution(seed = 0)
sage: [T.get_random_element() for _ in range(4)]
[(0.0796156410464, -0.0523767162758, 0.995448657286), (0.412359949059, 0.560681785936, -0.71
sage: T.reset_distribution()
sage: [T.get_random_element() for _ in range(4)]
[(0.0796156410464, -0.0523767162758, 0.995448657286), (0.412359949059, 0.560681785936, -0.71
```

set_random_number_generator(*rng='default'*)

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

EXAMPLE:

```
sage: T = SphericalDistribution()
sage: T.set_random_number_generator('default')
sage: T.set_seed(0)
sage: T.get_random_element()
(0.0796156410464, -0.0523767162758, 0.995448657286)
sage: T.set_random_number_generator('luxury')
```

```
sage: T.set_seed(0)
sage: T.get_random_element()
(0.0796156410464, -0.0523767162758, 0.995448657286)
```

set_seed(*seed*)

Set the seed for the underlying random number generator.

EXAMPLE:

```
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: `sage.probability.random_variable.ProbabilitySpace_generic`, `sage.probability.random_variable.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: `sage.probability.random_variable.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:

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

`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:

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

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: sage.probability.random_variable.RandomVariable_generic

A probability space.

domain()
x.__init__(...) initializes x; see help(type(x)) for signature

class sage.probability.random_variable.**RandomVariable_generic**(X, RR)
Bases: sage.structure.parent_base.ParentWithBase

A random variable.

codomain()
x.__init__(...) initializes x; see help(type(x)) for signature

domain()
x.__init__(...) initializes x; see help(type(x)) for signature

field()
x.__init__(...) initializes x; see help(type(x)) for signature

probability_space()
x.__init__(...) initializes x; see help(type(x)) for signature

sage.probability.random_variable.**is_DiscreteProbabilitySpace**(S)
x.__init__(...) initializes x; see help(type(x)) for signature

sage.probability.random_variable.**is_DiscreteRandomVariable**(X)
x.__init__(...) initializes x; see help(type(x)) for signature

sage.probability.random_variable.**is_ProbabilitySpace**(S)
x.__init__(...) initializes x; see help(type(x)) for signature

sage.probability.random_variable.**is_RandomVariable**(X)
x.__init__(...) initializes x; see help(type(x)) for signature

CHAPTER

THREE

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

INDEX

C

D

E

F

G

I

P

R

S

T

V