Stats

SymPy statistics module

Introduces a random variable type into the SymPy language.

Random variables may be declared using prebuilt functions such as Normal, Exponential, Coin, Die, etc... or built with functions like FiniteRV.

Queries on random expressions can be made using the functions

Expression	Meaning
`P(condition)`	Probability
`E(expression)`	Expected value
`variance(expression)`	Variance
`density(expression)`	Probability Density Function
`sample(expression)`	Produce a realization
`where(condition)`	Where the condition is true

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import P, E, variance, Die, Normal
>>> from sympy import Eq, simplify
>>> X, Y = Die('X', 6), Die('Y', 6) # Define two six sided dice
>>> Z = Normal('Z', 0, 1) # Declare a Normal random variable with mean 0, std 1
>>> P(X>3) # Probability X is greater than 3
1/2
>>> E(X+Y) # Expectation of the sum of two dice
7
>>> variance(X+Y) # Variance of the sum of two dice
35/6
```

Show SymPy Live Shell

```
>>> simplify(P(Z>1)) # Probability of Z being greater than 1
1/2 - erf(sqrt(2)/2)/2
```

Random Variable Types

Finite Types

sympy.stats. **DiscreteUniform**(*name*, *items*) [source]

Create a Finite Random Variable representing a uniform distribution over the input set.

Returns a RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import DiscreteUniform, density
>>> from sympy import symbols
```

Run code block in SymPy Live

```
>>> X = DiscreteUniform('X', symbols('a b c')) # equally likely over a, b, c
>>> density(X).dict
{a: 1/3, b: 1/3, c: 1/3}
```

Run code block in SymPy Live

```
>>> Y = DiscreteUniform('Y', list(range(5))) # distribution over a range
>>> density(Y).dict
{0: 1/5, 1: 1/5, 2: 1/5, 3: 1/5, 4: 1/5}
```

References

[R622]https://en.wikipedia.org/wiki/Discrete_uniform_distribution

[R623]http://mathworld.wolfram.com/DiscreteUniformDistribution.html

Show SymPy Live Shell

`sympy.stats.` **Die**(*name*, *sides=6*) [source]

Create a Finite Random Variable representing a fair die.

Returns a RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Die, density
```

Run code block in SymPy Live

```
>>> D6 = Die('D6', 6) # Six sided Die
>>> density(D6).dict
{1: 1/6, 2: 1/6, 3: 1/6, 4: 1/6, 5: 1/6, 6: 1/6}
```

Run code block in SymPy Live

```
>>> D4 = Die('D4', 4) # Four sided Die
>>> density(D4).dict
{1: 1/4, 2: 1/4, 3: 1/4, 4: 1/4}
```

`sympy.stats.` **Bernoulli**(*name*, *p*, *succ=1*, *fail=0*) [source]

Create a Finite Random Variable representing a Bernoulli process.

Returns a RandomSymbol

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Bernoulli, density
>>> from sympy import S
```

Run code block in SymPy Live

```
>>> X = Bernoulli('X', S(3)/4) # 1-0 Bernoulli variable, probability = 3/4
>>> density(X).dict
```

Show SymPy Live Shell

```
{0: 1/4, 1: 3/4}
```

Run code block in SymPy Live

```
>>> X = Bernoulli('X', S.Half, 'Heads', 'Tails') # A fair coin toss
>>> density(X).dict
{Heads: 1/2, Tails: 1/2}
```

References

[R624]https://en.wikipedia.org/wiki/Bernoulli_distribution

[R625]http://mathworld.wolfram.com/BernoulliDistribution.html

`sympy.stats.` **Coin**(*name*, *p=1/2*) [source]

Create a Finite Random Variable representing a Coin toss.

Probability p is the chance of gettings “Heads.” Half by default

Returns a RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Coin, density
>>> from sympy import Rational
```

Run code block in SymPy Live

```
>>> C = Coin('C') # A fair coin toss
>>> density(C).dict
{H: 1/2, T: 1/2}
```

Run code block in SymPy Live

```
>>> C2 = Coin('C2', Rational(3, 5)) # An unfair coin
>>> density(C2).dict
{H: 3/5, T: 2/5}
```

Show SymPy Live Shell

See also: `sympy.stats.Binomial`

References

[R626]https://en.wikipedia.org/wiki/Coin_flipping

`sympy.stats.` **`Binomial`**(*name*, *n*, *p*, *succ=1*, *fail=0*) [source]

Create a Finite Random Variable representing a binomial distribution.

Returns a RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Binomial, density
>>> from sympy import S
```

Run code block in SymPy Live

```
>>> X = Binomial('X', 4, S.Half) # Four "coin flips"
>>> density(X).dict
{0: 1/16, 1: 1/4, 2: 3/8, 3: 1/4, 4: 1/16}
```

References

[R627]https://en.wikipedia.org/wiki/Binomial_distribution

[R628]http://mathworld.wolfram.com/BinomialDistribution.html

`sympy.stats.` **`Hypergeometric`**(*name*, *N*, *m*, *n*) [source]

Create a Finite Random Variable representing a hypergeometric distribution.

Returns a RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Hypergeometric, density
>>> from sympy import S
```

Run code block in SymPy Live

```
>>> X = Hypergeometric('X', 10, 5, 3) # 10 marbles, 5 white (success), 3 draws
>>> density(X).dict
{0: 1/12, 1: 5/12, 2: 5/12, 3: 1/12}
```

References

[R629]https://en.wikipedia.org/wiki/Hypergeometric_distribution

[R630]http://mathworld.wolfram.com/HypergeometricDistribution.html

`sympy.stats.` **`FiniteRV`**(*name*, *density*) [source]

Create a Finite Random Variable given a dict representing the density.

Returns a RandomSymbol.

Run code block in SymPy Live

```
>>> from sympy.stats import FiniteRV, P, E
```

Run code block in SymPy Live

```
>>> density = {0: .1, 1: .2, 2: .3, 3: .4}
>>> X = FiniteRV('X', density)
```

Run code block in SymPy Live

```
>>> E(X)
2.00000000000000
>>> P(X >= 2)
0.700000000000000
```

Show SymPy Live Shell

sympy.stats. **Rademacher**(*name*) [source]

Create a Finite Random Variable representing a Rademacher distribution.

Return a RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Rademacher, density
```

Run code block in SymPy Live

```
>>> X = Rademacher('X')
>>> density(X).dict
{-1: 1/2, 1: 1/2}
```

See also: `sympy.stats.Bernoulli`

References

[R631]https://en.wikipedia.org/wiki/Rademacher_distribution

Discrete Types

sympy.stats. **Geometric**(*name*, *p*) [source]

Create a discrete random variable with a Geometric distribution.

The density of the Geometric distribution is given by

$$f(k) := p(1-p)^{k-1}$$

Parameters: p: A probability between 0 and 1
Returns: A RandomSymbol.

Show SymPy Live Shell

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Geometric, density, E, variance
>>> from sympy import Symbol, S
```

Run code block in SymPy Live

```
>>> p = S.One / 5
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Geometric("x", p)
```

Run code block in SymPy Live

```
>>> density(X)(z)
(4/5)**(z - 1)/5
```

Run code block in SymPy Live

```
>>> E(X)
5
```

Run code block in SymPy Live

```
>>> variance(X)
20
```

References

[R632]https://en.wikipedia.org/wiki/Geometric_distribution

[R633]http://mathworld.wolfram.com/GeometricDistribution.html

`sympy.stats.` **`Poisson`**(*name*, *lamda*) [source]

Create a discrete random variable with a Poisson distribution.

The density of the Poisson distribution is given by

Show SymPy Live Shell

$$f(k) := \frac{\lambda^k e^{-\lambda}}{k!}$$

Parameters: lamda: Positive number, a rate

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Poisson, density, E, variance
>>> from sympy import Symbol, simplify
```

Run code block in SymPy Live

```
>>> rate = Symbol("lambda", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Poisson("x", rate)
```

Run code block in SymPy Live

```
>>> density(X)(z)
lambda**z*exp(-lambda)/factorial(z)
```

Run code block in SymPy Live

```
>>> E(X)
lambda
```

Run code block in SymPy Live

```
>>> simplify(variance(X))
lambda
```

References

[R634]https://en.wikipedia.org/wiki/Poisson_distribution

[R635]http://mathworld.wolfram.com/PoissonDistribution.html

sympy.stats. **Logarithmic**(*name*, *p*) [source]

Create a discrete random variable with a Logarithmic distribution.

The density of the Logarithmic distribution is given by

$$f(k) := \frac{-p^k}{k \ln(1-p)}$$

Parameters: p: A value between 0 and 1
Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Logarithmic, density, E, variance
>>> from sympy import Symbol, S
```

Run code block in SymPy Live

```
>>> p = S.One / 5
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Logarithmic("x", p)
```

Run code block in SymPy Live

```
>>> density(X)(z)
-5**(-z)/(z*log(4/5))
```

Run code block in SymPy Live

```
>>> E(X)
-1/(-4*log(5) + 8*log(2))
```

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> variance(X)
-1/((-4*log(5) + 8*log(2))*(-2*log(5) + 4*log(2))) + 1/(-64*log(2)*log(5) + 64*log(2)**
```

References

[R636]https://en.wikipedia.org/wiki/Logarithmic_distribution

[R637]http://mathworld.wolfram.com/LogarithmicDistribution.html

sympy.stats. **NegativeBinomial**(*name*, *r*, *p*) [source]

Create a discrete random variable with a Negative Binomial distribution.

The density of the Negative Binomial distribution is given by

$$f(k) := \binom{k+r-1}{k}(1-p)^r p^k$$

Parameters: r: A positive value
p: A value between 0 and 1

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import NegativeBinomial, density, E, variance
>>> from sympy import Symbol, S
```

Run code block in SymPy Live

```
>>> r = 5
>>> p = S.One / 5
>>> z = Symbol("z")
```

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> X = NegativeBinomial("x", r, p)
```

Run code block in SymPy Live

```
>>> density(X)(z)
1024*5**(-z)*binomial(z + 4, z)/3125
```

Run code block in SymPy Live

```
>>> E(X)
5/4
```

Run code block in SymPy Live

```
>>> variance(X)
25/16
```

References

[R638]https://en.wikipedia.org/wiki/Negative_binomial_distribution

[R639]http://mathworld.wolfram.com/NegativeBinomialDistribution.html

`sympy.stats.` **YuleSimon**(*name*, *rho*) [source]

Create a discrete random variable with a Yule-Simon distribution.

The density of the Yule-Simon distribution is given by

$$f(k) := \rho B(k, \rho + 1)$$

Parameters: rho: A positive value

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import YuleSimon, density, E, variance
>>> from sympy import Symbol, simplify
```

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> p = 5
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = YuleSimon("x", p)
```

Run code block in SymPy Live

```
>>> density(X)(z)
5*beta(z, 6)
```

Run code block in SymPy Live

```
>>> simplify(E(X))
5/4
```

Run code block in SymPy Live

```
>>> simplify(variance(X))
25/48
```

References

[R640]https://en.wikipedia.org/wiki/Yule%E2%80%93Simon_distribution

sympy.stats. **Zeta**(*name*, *s*) [source]

Create a discrete random variable with a Zeta distribution.

The density of the Zeta distribution is given by

$$f(k) := \frac{1}{k^s \zeta(s)}$$

Parameters: **s: A value greater than 1**

Returns: A RandomSymbol.

Show SymPy Live Shell

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Zeta, density, E, variance
>>> from sympy import Symbol
```

Run code block in SymPy Live

```
>>> s = 5
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Zeta("x", s)
```

Run code block in SymPy Live

```
>>> density(X)(z)
1/(z**5*zeta(5))
```

Run code block in SymPy Live

```
>>> E(X)
pi**4/(90*zeta(5))
```

Run code block in SymPy Live

```
>>> variance(X)
-pi**8/(8100*zeta(5)**2) + zeta(3)/zeta(5)
```

References

[R641]https://en.wikipedia.org/wiki/Zeta_distribution

Continuous Types

sympy.stats. **Arcsin**(*name*, *a=0*, *b=1*) [source]

Creates a Continuous Random Variable with Arcsine distribution.

The density of the Arcsine distribution is given by

Show SymPy Live Shell

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

with $x \in (a, b)$. It must hold that $-\infty < a < b < \infty$.

Parameters: **a** : Real number, the left interval boundary
b : Real number, the right interval boundary

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Arcsin, density, cdf
>>> from sympy import Symbol, simplify
```

Run code block in SymPy Live

```
>>> a = Symbol("a", real=True)
>>> b = Symbol("b", real=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Arcsin("x", a, b)
```

Run code block in SymPy Live

```
>>> density(X)(z)
1/(pi*sqrt((-a + z)*(b - z)))
```

Run code block in SymPy Live

```
>>> cdf(X)(z)
Piecewise((0, a > z),
        (2*asin(sqrt((-a + z)/(-a + b)))/pi, b >= z),
        (1, True))
```

References

Show SymPy Live Shell

[R642]https://en.wikipedia.org/wiki/Arcsine_distribution

sympy.stats. **Benini**(*name*, *alpha*, *beta*, *sigma*) [source]

Creates a Continuous Random Variable with Benini distribution.

The density of the Benini distribution is given by

$$f(x) := e^{-\alpha \log \frac{x}{\sigma} - \beta \log^2 \left[\frac{x}{\sigma}\right]} \left(\frac{\alpha}{x} + \frac{2\beta \log \frac{x}{\sigma}}{x}\right)$$

This is a heavy-tailed distrubtion and is also known as the log-Rayleigh distribution.

Parameters:
alpha : Real number, $\alpha > 0$, a shape
beta : Real number, $\beta > 0$, a shape
sigma : Real number, $\sigma > 0$, a scale

Returns:
A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Benini, density, cdf
>>> from sympy import Symbol, simplify, pprint
```

Run code block in SymPy Live

```
>>> alpha = Symbol("alpha", positive=True)
>>> beta = Symbol("beta", positive=True)
>>> sigma = Symbol("sigma", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Benini("x", alpha, beta, sigma)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
```

Show SymPy Live Shell

```
/                 /  z  \\             /  z  \            2/  z  \
|       2*beta*log|-----||  - alpha*log|-----| - beta*log  |-----|
|alpha            \sigma/|             \sigma/             \sigma/
|----- + -----------------|*e
\  z             z        /
```

Run code block in SymPy Live

```
>>> cdf(X)(z)
Piecewise((1 - exp(-alpha*log(z/sigma) - beta*log(z/sigma)**2), sigma <= z),
        (0, True))
```

References

[R643]https://en.wikipedia.org/wiki/Benini_distribution

[R644]http://reference.wolfram.com/legacy/v8/ref/BeniniDistribution.html

`sympy.stats.` **Beta**(*name*, *alpha*, *beta*) [source]

Creates a Continuous Random Variable with Beta distribution.

The density of the Beta distribution is given by

$$f(x) := \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha,\beta)}$$

with $x \in [0,1]$.

Parameters:
alpha : Real number, $\alpha > 0$, a shape
beta : Real number, $\beta > 0$, a shape

Returns:
A RandomSymbol.

Examples

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> from sympy.stats import Beta, density, E, variance
>>> from sympy import Symbol, simplify, pprint, factor
```

Run code block in SymPy Live

```
>>> alpha = Symbol("alpha", positive=True)
>>> beta = Symbol("beta", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Beta("x", alpha, beta)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
 alpha - 1        beta - 1
z         *(1 - z)
--------------------------
      B(alpha, beta)
```

Run code block in SymPy Live

```
>>> simplify(E(X))
alpha/(alpha + beta)
```

Run code block in SymPy Live

```
>>> factor(simplify(variance(X)))  #doctest: +SKIP
alpha*beta/((alpha + beta)**2*(alpha + beta + 1))
```

References

[R645]https://en.wikipedia.org/wiki/Beta_distribution

[R646]http://mathworld.wolfram.com/BetaDistribution.html

`sympy.stats.` **BetaPrime**(*name*, *alpha*, *beta*) [source]

Creates a Continuous Random Variable with Beta prime distribution.

Show SymPy Live Shell

The density of the Beta prime distribution is given by

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

with $x > 0$.

Parameters:
alpha : Real number, $\alpha > 0$, a shape
beta : Real number, $\beta > 0$, a shape

Returns:
A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import BetaPrime, density
>>> from sympy import Symbol, pprint
```

Run code block in SymPy Live

```
>>> alpha = Symbol("alpha", positive=True)
>>> beta = Symbol("beta", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = BetaPrime("x", alpha, beta)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
 alpha - 1        -alpha - beta
z         *(z + 1)
------------------------------
         B(alpha, beta)
```

References

Show SymPy Live Shell

[R647]https://en.wikipedia.org/wiki/Beta_prime_distribution

[R648]http://mathworld.wolfram.com/BetaPrimeDistribution.html

sympy.stats. **Cauchy**(*name*, *x0*, *gamma*) [source]

Creates a Continuous Random Variable with Cauchy distribution.

The density of the Cauchy distribution is given by

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

Parameters: **x0** : Real number, the location
gamma : Real number, $\gamma > 0$, a scale

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Cauchy, density
>>> from sympy import Symbol
```

Run code block in SymPy Live

```
>>> x0 = Symbol("x0")
>>> gamma = Symbol("gamma", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Cauchy("x", x0, gamma)
```

Run code block in SymPy Live

```
>>> density(X)(z)
1/(pi*gamma*(1 + (-x0 + z)**2/gamma**2))
```

Show SymPy Live Shell

References

[R649]https://en.wikipedia.org/wiki/Cauchy_distribution

[R650]http://mathworld.wolfram.com/CauchyDistribution.html

`sympy.stats.` **Chi**(*name*, *k*) [source]

Creates a Continuous Random Variable with Chi distribution.

The density of the Chi distribution is given by

$$f(x) := \frac{2^{1-k/2}x^{k-1}e^{-x^2/2}}{\Gamma(k/2)}$$

with $x \geq 0$.

Parameters: **k** : Positive integer, The number of degrees of freedom

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Chi, density, E
>>> from sympy import Symbol, simplify
```

Run code block in SymPy Live

```
>>> k = Symbol("k", integer=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Chi("x", k)
```

Run code block in SymPy Live

```
>>> density(X)(z)
2**(1 - k/2)*z**(k - 1)*exp(-z**2/2)/gamma(k/2)
```

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> simplify(E(X))
sqrt(2)*gamma(k/2 + 1/2)/gamma(k/2)
```

References

[R651]https://en.wikipedia.org/wiki/Chi_distribution

[R652]http://mathworld.wolfram.com/ChiDistribution.html

sympy.stats. **ChiNoncentral**(*name*, *k*, *l*) [source]

Creates a Continuous Random Variable with Non-central Chi distribution.

The density of the Non-central Chi distribution is given by

$$f(x) := \frac{e^{-(x^2+\lambda^2)/2} x^k \lambda}{(\lambda x)^{k/2}} I_{k/2-1}(\lambda x)$$

with $x \geq 0$. Here, $I_\nu(x)$ is the modified Bessel function of the first kind.

Parameters:
k : A positive Integer, $k > 0$, the number of degrees of freedom
lambda : Real number, $\lambda > 0$, Shift parameter

Returns:
A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import ChiNoncentral, density
>>> from sympy import Symbol
```

Run code block in SymPy Live

```
>>> k = Symbol("k", integer=True)
>>> l = Symbol("l")
```

Show SymPy Live Shell

```
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = ChiNoncentral("x", k, l)
```

Run code block in SymPy Live

```
>>> density(X)(z)
l*z**k*(l*z)**(-k/2)*exp(-l**2/2 - z**2/2)*besseli(k/2 - 1, l*z)
```

References

[R653]https://en.wikipedia.org/wiki/Noncentral_chi_distribution

`sympy.stats.` **ChiSquared**(*name*, *k*) [source]

Creates a Continuous Random Variable with Chi-squared distribution.

The density of the Chi-squared distribution is given by

$$f(x) := \frac{1}{2^{\frac{k}{2}}\Gamma\left(\frac{k}{2}\right)} x^{\frac{k}{2}-1} e^{-\frac{x}{2}}$$

with $x \geq 0$.

Parameters: **k** : Positive integer, The number of degrees of freedom

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import ChiSquared, density, E, variance, moment
>>> from sympy import Symbol
```

Run code block in SymPy Live

```
>>> k = Symbol("k", integer=True, positive=True)
```

Show SymPy Live Shell

```
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = ChiSquared("x", k)
```

Run code block in SymPy Live

```
>>> density(X)(z)
2**(-k/2)*z**(k/2 - 1)*exp(-z/2)/gamma(k/2)
```

Run code block in SymPy Live

```
>>> E(X)
k
```

Run code block in SymPy Live

```
>>> variance(X)
2*k
```

Run code block in SymPy Live

```
>>> moment(X, 3)
k**3 + 6*k**2 + 8*k
```

References

[R654]https://en.wikipedia.org/wiki/Chi_squared_distribution

[R655]http://mathworld.wolfram.com/Chi-SquaredDistribution.html

`sympy.stats.` **Dagum**(*name*, *p*, *a*, *b*) [source]

Creates a Continuous Random Variable with Dagum distribution.

The density of the Dagum distribution is given by

$$f(x) := \frac{ap}{x}\left(\frac{\left(\frac{x}{b}\right)^{ap}}{\left(\left(\frac{x}{b}\right)^{a}+1\right)^{p+1}}\right)$$

Show SymPy Live Shell

with $x > 0$.

Parameters:
p : Real number, $p > 0$, a shape
a : Real number, $a > 0$, a shape
b : Real number, $b > 0$, a scale

Returns:
A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Dagum, density, cdf
>>> from sympy import Symbol
```

Run code block in SymPy Live

```
>>> p = Symbol("p", positive=True)
>>> a = Symbol("a", positive=True)
>>> b = Symbol("b", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Dagum("x", p, a, b)
```

Run code block in SymPy Live

```
>>> density(X)(z)
a*p*(z/b)**(a*p)*((z/b)**a + 1)**(-p - 1)/z
```

Run code block in SymPy Live

```
>>> cdf(X)(z)
Piecewise(((1 + (z/b)**(-a))**(-p), z >= 0), (0, True))
```

References

[R656]https://en.wikipedia.org/wiki/Dagum_distribution

`sympy.stats.` **Erlang**(*name*, *k*, *l*) [source]

Show SymPy Live Shell

Creates a Continuous Random Variable with Erlang distribution.

The density of the Erlang distribution is given by

$$f(x) := \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!}$$

with $x \in [0, \infty]$.

Parameters: **k** : Positive integer
l : Real number, $\lambda > 0$, the rate

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Erlang, density, cdf, E, variance
>>> from sympy import Symbol, simplify, pprint
```

Run code block in SymPy Live

```
>>> k = Symbol("k", integer=True, positive=True)
>>> l = Symbol("l", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Erlang("x", k, l)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
 k  k - 1  -l*z
l *z     *e
---------------
    Gamma(k)
```

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> C = cdf(X)(z)
>>> pprint(C, use_unicode=False)
/lowergamma(k, l*z)
|------------------  for z > 0
<     Gamma(k)
|
\        0           otherwise
```

Run code block in SymPy Live

```
>>> E(X)
k/l
```

Run code block in SymPy Live

```
>>> simplify(variance(X))
k/l**2
```

References

[R657]https://en.wikipedia.org/wiki/Erlang_distribution

[R658]http://mathworld.wolfram.com/ErlangDistribution.html

`sympy.stats.` **`Exponential`**(*name*, *rate*) [source]

Create a continuous random variable with an Exponential distribution.

The density of the exponential distribution is given by

$$f(x) := \lambda \exp(-\lambda x)$$

with $x > 0$. Note that the expected value is $1/\lambda$.

Parameters: **rate** : A positive Real number, $\lambda > 0$, the rate (or inverse scale/inverse mean)

Returns: A RandomSymbol.

Show SymPy Live Shell

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Exponential, density, cdf, E
>>> from sympy.stats import variance, std, skewness
>>> from sympy import Symbol
```

Run code block in SymPy Live

```
>>> l = Symbol("lambda", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Exponential("x", l)
```

Run code block in SymPy Live

```
>>> density(X)(z)
lambda*exp(-lambda*z)
```

Run code block in SymPy Live

```
>>> cdf(X)(z)
Piecewise((1 - exp(-lambda*z), z >= 0), (0, True))
```

Run code block in SymPy Live

```
>>> E(X)
1/lambda
```

Run code block in SymPy Live

```
>>> variance(X)
lambda**(-2)
```

Run code block in SymPy Live

```
>>> skewness(X)
2
```

Run code block in SymPy Live

```
>>> X = Exponential('x', 10)
```

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> density(X)(z)
10*exp(-10*z)
```

Run code block in SymPy Live

```
>>> E(X)
1/10
```

Run code block in SymPy Live

```
>>> std(X)
1/10
```

References

[R659]https://en.wikipedia.org/wiki/Exponential_distribution

[R660]http://mathworld.wolfram.com/ExponentialDistribution.html

`sympy.stats.` **FDistribution**(*name*, *d1*, *d2*) [source]

Create a continuous random variable with a F distribution.

The density of the F distribution is given by

$$f(x) := \frac{\sqrt{\frac{(d_1 x)^{d_1} d_2^{d_2}}{(d_1 x + d_2)^{d_1 + d_2}}}}{x \mathrm{B}\left(\frac{d_1}{2}, \frac{d_2}{2}\right)}$$

with $x > 0$.

Parameters:
d1 : $d_1 > 0$, where d_1 is the degrees of freedom (n_1 - 1)
d2 : $d_2 > 0$, where d_2 is the degrees of freedom (n_2 - 1)

Returns:
A RandomSymbol.

Examples

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> from sympy.stats import FDistribution, density
>>> from sympy import Symbol, simplify, pprint
```

Run code block in SymPy Live

```
>>> d1 = Symbol("d1", positive=True)
>>> d2 = Symbol("d2", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = FDistribution("x", d1, d2)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
  d2
  --    ______________________________
  2    /       d1            -d1 - d2
d2  *\/  (d1*z)  *(d1*z + d2)
-------------------------------------
              /d1  d2\
           z*B|--, --|
              \2   2 /
```

References

[R661]https://en.wikipedia.org/wiki/F-distribution

[R662]http://mathworld.wolfram.com/F-Distribution.html

`sympy.stats.` **FisherZ**(*name*, *d1*, *d2*) [source]

Create a Continuous Random Variable with an Fisher's Z distribution.

The density of the Fisher's Z distribution is given by

Show SymPy Live Shell

$$f(x) := \frac{2d_1^{d_1/2} d_2^{d_2/2}}{\mathrm{B}(d_1/2, d_2/2)} \frac{e^{d_1 z}}{\left(d_1 e^{2z} + d_2\right)^{(d_1+d_2)/2}}$$

Parameters: **d1** : $d_1 > 0$, degree of freedom
d2 : $d_2 > 0$, degree of freedom

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import FisherZ, density
>>> from sympy import Symbol, simplify, pprint
```

Run code block in SymPy Live

```
>>> d1 = Symbol("d1", positive=True)
>>> d2 = Symbol("d2", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = FisherZ("x", d1, d2)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
                            d1   d2
    d1   d2               - -- - --
    --   --                 2    2
    2    2  /    2*z     \          d1*z
2*d1  *d2  *\d1*e    + d2/         *e
-----------------------------------------
                /d1  d2\
               B|--, --|
                \2   2 /
```

Show SymPy Live Shell

References

[R663]https://en.wikipedia.org/wiki/Fisher%27s_z-distribution

[R664]http://mathworld.wolfram.com/Fishersz-Distribution.html

`sympy.stats.` **Frechet**(*name*, *a*, *s=1*, *m=0*) [source]

Create a continuous random variable with a Frechet distribution.

The density of the Frechet distribution is given by

$$f(x) := \frac{\alpha}{s}\left(\frac{x-m}{s}\right)^{-1-\alpha} e^{-(\frac{x-m}{s})^{-\alpha}}$$

with $x \geq m$.

Parameters:
a : Real number, $a \in (0, \infty)$ the shape
s : Real number, $s \in (0, \infty)$ the scale
m : Real number, $m \in (-\infty, \infty)$ the minimum

Returns:
A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Frechet, density, E, std, cdf
>>> from sympy import Symbol, simplify
```

Run code block in SymPy Live

```
>>> a = Symbol("a", positive=True)
>>> s = Symbol("s", positive=True)
>>> m = Symbol("m", real=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Frechet("x", a, s, m)
```

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> density(X)(z)
a*((-m + z)/s)**(-a - 1)*exp(-((-m + z)/s)**(-a))/s
```

Run code block in SymPy Live

```
>>> cdf(X)(z)
 Piecewise((exp(-((-m + z)/s)**(-a)), m <= z), (0, True))
```

References

[R665]https://en.wikipedia.org/wiki/Fr%C3%A9chet_distribution

`sympy.stats.` **Gamma**(*name*, *k*, *theta*) [source]

Create a continuous random variable with a Gamma distribution.

The density of the Gamma distribution is given by

$$f(x) := \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-\frac{x}{\theta}}$$

with $x \in [0, 1]$.

Parameters:
k : Real number, $k > 0$, a shape
theta : Real number, $\theta > 0$, a scale

Returns:
A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Gamma, density, cdf, E, variance
>>> from sympy import Symbol, pprint, simplify
```

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> k = Symbol("k", positive=True)
>>> theta = Symbol("theta", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Gamma("x", k, theta)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
                  -z
                -----
     -k  k - 1  theta
theta  *z     *e
---------------------
       Gamma(k)
```

Run code block in SymPy Live

```
>>> C = cdf(X, meijerg=True)(z)
>>> pprint(C, use_unicode=False)
/            /     z  \
|k*lowergamma|k, -----|
|            \   theta/
<----------------------  for z >= 0
|     Gamma(k + 1)
|
\          0             otherwise
```

Run code block in SymPy Live

```
>>> E(X)
k*theta
```

Run code block in SymPy Live

```
>>> V = simplify(variance(X))
>>> pprint(V, use_unicode=False)
       2
k*theta
```

Show SymPy Live Shell

References

[R666]https://en.wikipedia.org/wiki/Gamma_distribution

[R667]http://mathworld.wolfram.com/GammaDistribution.html

sympy.stats. **GammaInverse**(*name*, *a*, *b*) [source]

Create a continuous random variable with an inverse Gamma distribution.

The density of the inverse Gamma distribution is given by

$$f(x) := \frac{\beta^\alpha}{\Gamma(\alpha)} x^{-\alpha-1} \exp\left(\frac{-\beta}{x}\right)$$

with $x > 0$.

Parameters:
a : Real number, $a > 0$ a shape
b : Real number, $b > 0$ a scale

Returns:
A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import GammaInverse, density, cdf, E, variance
>>> from sympy import Symbol, pprint
```

Run code block in SymPy Live

```
>>> a = Symbol("a", positive=True)
>>> b = Symbol("b", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = GammaInverse("x", a, b)
```

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
            -b
            ---
 a  -a - 1   z
b *z      *e
---------------
   Gamma(a)
```

Run code block in SymPy Live

```
>>> cdf(X)(z)
Piecewise((uppergamma(a, b/z)/gamma(a), z > 0), (0, True))
```

References

[R668]https://en.wikipedia.org/wiki/Inverse-gamma_distribution

sympy.stats. **Kumaraswamy**(*name*, *a*, *b*) [source]

Create a Continuous Random Variable with a Kumaraswamy distribution.

The density of the Kumaraswamy distribution is given by

$$f(x) := abx^{a-1}(1-x^a)^{b-1}$$

with $x \in [0,1]$.

Parameters:
a : Real number, $a > 0$ a shape
b : Real number, $b > 0$ a shape

Returns:
A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Kumaraswamy, density, E, variance, cdf
```

Show SymPy Live Shell

```
>>> from sympy import Symbol, simplify, pprint
```

Run code block in SymPy Live

```
>>> a = Symbol("a", positive=True)
>>> b = Symbol("b", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Kumaraswamy("x", a, b)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
                   b - 1
     a - 1 /     a\
a*b*z     *\1 - z /
```

Run code block in SymPy Live

```
>>> cdf(X)(z)
Piecewise((0, z < 0), (1 - (1 - z**a)**b, z <= 1), (1, True))
```

References

[R669]https://en.wikipedia.org/wiki/Kumaraswamy_distribution

`sympy.stats.` **Laplace**(*name*, *mu*, *b*) [source]

Create a continuous random variable with a Laplace distribution.

The density of the Laplace distribution is given by

$$f(x) := \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right)$$

Parameters: **mu** : Real number or a list/matrix, the location (mean) or the location vector

Show SymPy Live Shell

b : Real number or a positive definite matrix, representing a scale or the covariance matrix.

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Laplace, density, cdf
>>> from sympy import Symbol, pprint
```

Run code block in SymPy Live

```
>>> mu = Symbol("mu")
>>> b = Symbol("b", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Laplace("x", mu, b)
```

Run code block in SymPy Live

```
>>> density(X)(z)
exp(-Abs(mu - z)/b)/(2*b)
```

Run code block in SymPy Live

```
>>> cdf(X)(z)
Piecewise((exp((-mu + z)/b)/2, mu > z), (1 - exp((mu - z)/b)/2, True))
```

Run code block in SymPy Live

```
>>> L = Laplace('L', [1, 2], [[1, 0], [0, 1]])
>>> pprint(density(L)(1, 2), use_unicode=False)
 5        /     ____\
e *besselk\0, \/ 35 /
---------------------
          pi
```

Show SymPy Live Shell

References

[R670]https://en.wikipedia.org/wiki/Laplace_distribution

[R671]http://mathworld.wolfram.com/LaplaceDistribution.html

`sympy.stats.` **Logistic**(*name*, *mu*, *s*) [source]

Create a continuous random variable with a logistic distribution.

The density of the logistic distribution is given by

$$f(x) := \frac{e^{-(x-\mu)/s}}{s\left(1+e^{-(x-\mu)/s}\right)^2}$$

Parameters: **mu** : Real number, the location (mean)
s : Real number, $s > 0$ a scale

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Logistic, density, cdf
>>> from sympy import Symbol
```

Run code block in SymPy Live

```
>>> mu = Symbol("mu", real=True)
>>> s = Symbol("s", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Logistic("x", mu, s)
```

Run code block in SymPy Live

```
>>> density(X)(z)
exp((mu - z)/s)/(s*(exp((mu - z)/s) + 1)**2)
```

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> cdf(X)(z)
1/(exp((mu - z)/s) + 1)
```

References

[R672]https://en.wikipedia.org/wiki/Logistic_distribution

[R673]http://mathworld.wolfram.com/LogisticDistribution.html

sympy.stats. **LogNormal**(*name*, *mean*, *std*) [source]

Create a continuous random variable with a log-normal distribution.

The density of the log-normal distribution is given by

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

with $x \geq 0$.

Parameters: **mu** : Real number, the log-scale
sigma : Real number, $\sigma^2 > 0$ a shape

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import LogNormal, density
>>> from sympy import Symbol, simplify, pprint
```

Run code block in SymPy Live

```
>>> mu = Symbol("mu", real=True)
>>> sigma = Symbol("sigma", positive=True)
>>> z = Symbol("z")
```

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> X = LogNormal("x", mu, sigma)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
                     2
       -(-mu + log(z))
       ----------------
                  2
  ___         2*sigma
\/ 2 *e
------------------------
      ____
    2*\/ pi *sigma*z
```

Run code block in SymPy Live

```
>>> X = LogNormal('x', 0, 1) # Mean 0, standard deviation 1
```

Run code block in SymPy Live

```
>>> density(X)(z)
sqrt(2)*exp(-log(z)**2/2)/(2*sqrt(pi)*z)
```

References

[R674]https://en.wikipedia.org/wiki/Lognormal

[R675]http://mathworld.wolfram.com/LogNormalDistribution.html

`sympy.stats.` **`Maxwell`**(*name*, *a*) [source]

Create a continuous random variable with a Maxwell distribution.

The density of the Maxwell distribution is given by

Show SymPy Live Shell

$$f(x) := \sqrt{\frac{2}{\pi}} \frac{x^2 e^{-x^2/(2a^2)}}{a^3}$$

with $x \geq 0$.

Parameters: **a** : Real number, $a > 0$

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Maxwell, density, E, variance
>>> from sympy import Symbol, simplify
```

Run code block in SymPy Live

```
>>> a = Symbol("a", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Maxwell("x", a)
```

Run code block in SymPy Live

```
>>> density(X)(z)
sqrt(2)*z**2*exp(-z**2/(2*a**2))/(sqrt(pi)*a**3)
```

Run code block in SymPy Live

```
>>> E(X)
2*sqrt(2)*a/sqrt(pi)
```

Run code block in SymPy Live

```
>>> simplify(variance(X))
a**2*(-8 + 3*pi)/pi
```

References

Show SymPy Live Shell

[R676]https://en.wikipedia.org/wiki/Maxwell_distribution

[R677]http://mathworld.wolfram.com/MaxwellDistribution.html

sympy.stats. **Nakagami**(*name*, *mu*, *omega*) [source]

Create a continuous random variable with a Nakagami distribution.

The density of the Nakagami distribution is given by

$$f(x) := \frac{2\mu^{\mu}}{\Gamma(\mu)\omega^{\mu}} x^{2\mu-1} \exp\left(-\frac{\mu}{\omega}x^2\right)$$

with $x > 0$.

Parameters:

mu : Real number, $\mu \geq \frac{1}{2}$ a shape

omega : Real number, $\omega > 0$, the spread

Returns:

A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Nakagami, density, E, variance, cdf
>>> from sympy import Symbol, simplify, pprint
```

Run code block in SymPy Live

```
>>> mu = Symbol("mu", positive=True)
>>> omega = Symbol("omega", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Nakagami("x", mu, omega)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
```

Show SymPy Live Shell

```
                                 2
                              -mu*z
                              -------
    mu      -mu  2*mu - 1   omega
2*mu  *omega   *z         *e
---------------------------------
            Gamma(mu)
```

Run code block in SymPy Live

```
>>> simplify(E(X))
sqrt(mu)*sqrt(omega)*gamma(mu + 1/2)/gamma(mu + 1)
```

Run code block in SymPy Live

```
>>> V = simplify(variance(X))
>>> pprint(V, use_unicode=False)
                    2
         omega*Gamma (mu + 1/2)
omega - ----------------------
        Gamma(mu)*Gamma(mu + 1)
```

Run code block in SymPy Live

```
>>> cdf(X)(z)
Piecewise((lowergamma(mu, mu*z**2/omega)/gamma(mu), z > 0),
        (0, True))
```

References

[R678]https://en.wikipedia.org/wiki/Nakagami_distribution

`sympy.stats.` **`Normal`**(*name*, *mean*, *std*) [source]

Create a continuous random variable with a Normal distribution.

The density of the Normal distribution is given by

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

Show SymPy Live Shell

Parameters:

mu : Real number or a list representing the mean or the mean vector
sigma : Real number or a positive definite sqaure matrix,

$\sigma^2 > 0$ the variance

Returns:

A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Normal, density, E, std, cdf, skewness
>>> from sympy import Symbol, simplify, pprint, factor, together, factor_terms
```

Run code block in SymPy Live

```
>>> mu = Symbol("mu")
>>> sigma = Symbol("sigma", positive=True)
>>> z = Symbol("z")
>>> y = Symbol("y")
>>> X = Normal("x", mu, sigma)
```

Run code block in SymPy Live

```
>>> density(X)(z)
sqrt(2)*exp(-(-mu + z)**2/(2*sigma**2))/(2*sqrt(pi)*sigma)
```

Run code block in SymPy Live

```
>>> C = simplify(cdf(X))(z) # it needs a little more help...
>>> pprint(C, use_unicode=False)
   /  ___             \
   |\/ 2 *(-mu + z)|
erf|---------------|
   \    2*sigma    /   1
-------------------- + -
         2             2
```

Run code block in SymPy Live

```
>>> simplify(skewness(X))
```

Show SymPy Live Shell

```
0
```

Run code block in SymPy Live

```
>>> X = Normal("x", 0, 1) # Mean 0, standard deviation 1
>>> density(X)(z)
sqrt(2)*exp(-z**2/2)/(2*sqrt(pi))
```

Run code block in SymPy Live

```
>>> E(2*X + 1)
1
```

Run code block in SymPy Live

```
>>> simplify(std(2*X + 1))
2
```

Run code block in SymPy Live

```
>>> m = Normal('X', [1, 2], [[2, 1], [1, 2]])
>>> from sympy.stats.joint_rv import marginal_distribution
>>> pprint(density(m)(y, z))
       /1   y\ /2*y   z\   /    z\ /  y   2*z    \
       |- - -|*|--- - -| + |1 - -|*|- - + --- - 1|
  ___  \2   2/ \ 3    3/   \    2/ \  3    3     /
\/ 3 *e
------------------------------------------------
                      6*pi
```

Run code block in SymPy Live

```
>>> marginal_distribution(m, m[0])(1)
 1/(2*sqrt(pi))
```

References

[R679]https://en.wikipedia.org/wiki/Normal_distribution

[R680]http://mathworld.wolfram.com/NormalDistributionFunction.html

`sympy.stats.` **Pareto**(*name*, *xm*, *alpha*) [source]

Show SymPy Live Shell

Create a continuous random variable with the Pareto distribution.

The density of the Pareto distribution is given by

$$f(x) := \frac{\alpha\, x_m^\alpha}{x^{\alpha+1}}$$

with $x \in [x_m, \infty]$.

Parameters: **xm** : Real number, $x_m > 0$, a scale
alpha : Real number, $\alpha > 0$, a shape

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Pareto, density
>>> from sympy import Symbol
```

Run code block in SymPy Live

```
>>> xm = Symbol("xm", positive=True)
>>> beta = Symbol("beta", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Pareto("x", xm, beta)
```

Run code block in SymPy Live

```
>>> density(X)(z)
beta*xm**beta*z**(-beta - 1)
```

References

[R681]https://en.wikipedia.org/wiki/Pareto_distribution

[R682]http://mathworld.wolfram.com/ParetoDistribution.html

sympy.stats. **QuadraticU**(*name*, *a*, *b*) [source]

Create a Continuous Random Variable with a U-quadratic distribution.

The density of the U-quadratic distribution is given by

$$f(x) := \alpha(x-\beta)^2$$

with $x \in [a, b]$.

Parameters: **a** : Real number
b : Real number, $a < b$

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import QuadraticU, density, E, variance
>>> from sympy import Symbol, simplify, factor, pprint
```

Run code block in SymPy Live

```
>>> a = Symbol("a", real=True)
>>> b = Symbol("b", real=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = QuadraticU("x", a, b)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
/                2
|   /  a   b    \
|12*|- - - + z|
```

Show SymPy Live Shell

```
|   \  2   2    /
<-----------------  for And(b >= z, a <= z)
|            3
|    (-a + b)
|
\        0                 otherwise
```

References

[R683]https://en.wikipedia.org/wiki/U-quadratic_distribution

sympy.stats. **RaisedCosine**(*name*, *mu*, *s*) [source]

Create a Continuous Random Variable with a raised cosine distribution.

The density of the raised cosine distribution is given by

$$f(x) := \frac{1}{2s}\left(1 + \cos\left(\frac{x-\mu}{s}\pi\right)\right)$$

with $x \in [\mu - s, \mu + s]$.

Parameters: **mu** : Real number
s : Real number, $s > 0$

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import RaisedCosine, density, E, variance
>>> from sympy import Symbol, simplify, pprint
```

Run code block in SymPy Live

```
>>> mu = Symbol("mu", real=True)
>>> s = Symbol("s", positive=True)
```

Show SymPy Live Shell

```
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = RaisedCosine("x", mu, s)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
/   /pi*(-mu + z)\
|cos|------------| + 1
|   \     s      /
<---------------------  for And(z >= mu - s, z <= mu + s)
|         2*s
|
\          0                        otherwise
```

References

[R684]https://en.wikipedia.org/wiki/Raised_cosine_distribution

`sympy.stats.` **Rayleigh**(*name*, *sigma*) [source]

Create a continuous random variable with a Rayleigh distribution.

The density of the Rayleigh distribution is given by

$$f(x) := \frac{x}{\sigma^2} e^{-x^2/2\sigma^2}$$

with $x > 0$.

Parameters: **sigma** : Real number, $\sigma > 0$

Returns: A RandomSymbol.

Examples

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> from sympy.stats import Rayleigh, density, E, variance
>>> from sympy import Symbol, simplify
```

Run code block in SymPy Live

```
>>> sigma = Symbol("sigma", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Rayleigh("x", sigma)
```

Run code block in SymPy Live

```
>>> density(X)(z)
z*exp(-z**2/(2*sigma**2))/sigma**2
```

Run code block in SymPy Live

```
>>> E(X)
sqrt(2)*sqrt(pi)*sigma/2
```

Run code block in SymPy Live

```
>>> variance(X)
-pi*sigma**2/2 + 2*sigma**2
```

References

[R685]https://en.wikipedia.org/wiki/Rayleigh_distribution

[R686]http://mathworld.wolfram.com/RayleighDistribution.html

`sympy.stats.` **`StudentT`**(*name*, *nu*) [source]

Create a continuous random variable with a student's t distribution.

The density of the student's t distribution is given by

Show SymPy Live Shell

$$f(x) := \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

Parameters: **nu** : Real number, $\nu > 0$, the degrees of freedom

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import StudentT, density, E, variance, cdf
>>> from sympy import Symbol, simplify, pprint
```

Run code block in SymPy Live

```
>>> nu = Symbol("nu", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = StudentT("x", nu)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
           nu   1
         - -- - -
           2    2
 /     2\
 |    z |
 |1 + --|
 \    nu/
-----------------
  ____  /     nu\
\/ nu *B|1/2, --|
        \     2 /
```

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> cdf(X)(z)
1/2 + z*gamma(nu/2 + 1/2)*hyper((1/2, nu/2 + 1/2), (3/2,),
                                -z**2/nu)/(sqrt(pi)*sqrt(nu)*gamma(nu/2))
```

References

[R687]https://en.wikipedia.org/wiki/Student_t-distribution

[R688]http://mathworld.wolfram.com/Studentst-Distribution.html

`sympy.stats.` **`Triangular`**(*name*, *a*, *b*, *c*) [source]

Create a continuous random variable with a triangular distribution.

The density of the triangular distribution is given by

$$f(x) := \begin{cases} 0 & \text{for } x < a, \\ \frac{2(x-a)}{(b-a)(c-a)} & \text{for } a \leq x < c, \\ \frac{2}{b-a} & \text{for } x = c, \\ \frac{2(b-x)}{(b-a)(b-c)} & \text{for } c < x \leq b, \\ 0 & \text{for } b < x. \end{cases}$$

Parameters:
a : Real number, $a \in (-\infty, \infty)$
b : Real number, $a < b$
c : Real number, $a \leq c \leq b$

Returns:
A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Triangular, density, E
>>> from sympy import Symbol, pprint
```

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> a = Symbol("a")
>>> b = Symbol("b")
>>> c = Symbol("c")
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Triangular("x", a,b,c)
```

Run code block in SymPy Live

```
>>> pprint(density(X)(z), use_unicode=False)
/    -2*a + 2*z
|-----------------  for And(a <= z, c > z)
|(-a + b)*(-a + c)
|
|       2
|     ------              for c = z
<     -a + b
|
|   2*b - 2*z
|----------------   for And(b >= z, c < z)
|(-a + b)*(b - c)
|
\        0                otherwise
```

References

[R689]https://en.wikipedia.org/wiki/Triangular_distribution

[R690]http://mathworld.wolfram.com/TriangularDistribution.html

`sympy.stats.` **`Uniform`**(*name*, *left*, *right*) [source]

Create a continuous random variable with a uniform distribution.

The density of the uniform distribution is given by

Show SymPy Live Shell

$$f(x) := \begin{cases} \frac{1}{b-a} & \text{for } x \in [a, b] \\ 0 & \text{otherwise} \end{cases}$$

with $x \in [a, b]$.

Parameters: **a** : Real number, $-\infty < a$ the left boundary
b : Real number, $a < b < \infty$ the right boundary

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Uniform, density, cdf, E, variance, skewness
>>> from sympy import Symbol, simplify
```

Run code block in SymPy Live

```
>>> a = Symbol("a", negative=True)
>>> b = Symbol("b", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Uniform("x", a, b)
```

Run code block in SymPy Live

```
>>> density(X)(z)
Piecewise((1/(-a + b), (b >= z) & (a <= z)), (0, True))
```

Run code block in SymPy Live

```
>>> cdf(X)(z)  # doctest: +SKIP
-a/(-a + b) + z/(-a + b)
```

Run code block in SymPy Live

```
>>> simplify(E(X))
a/2 + b/2
```

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> simplify(variance(X))
a**2/12 - a*b/6 + b**2/12
```

References

[R691]https://en.wikipedia.org/wiki/Uniform_distribution_%28continuous%29

[R692]http://mathworld.wolfram.com/UniformDistribution.html

sympy.stats. **UniformSum**(*name*, *n*) [source]

Create a continuous random variable with an Irwin-Hall distribution.

The probability distribution function depends on a single parameter n which is an integer.

The density of the Irwin-Hall distribution is given by

$$f(x) := \frac{1}{(n-1)!}\sum_{k=0}^{\lfloor x \rfloor}(-1)^k \binom{n}{k}(x-k)^{n-1}$$

Parameters: **n** : A positive Integer, $n > 0$

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import UniformSum, density, cdf
>>> from sympy import Symbol, pprint
```

Run code block in SymPy Live

```
>>> n = Symbol("n", integer=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = UniformSum("x", n)
```

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
floor(z)
  ___
  \  `
   \          k          n - 1 /n\
    )    (-1) *(-k + z)     *| |
   /                          \k/
  /__,
 k = 0
--------------------------------
            (n - 1)!
```

Run code block in SymPy Live

```
>>> cdf(X)(z)
Piecewise((0, z < 0), (Sum((-1)**_k*(-_k + z)**n*binomial(n, _k),
                (_k, 0, floor(z)))/factorial(n), n >= z), (1, True))
```

Compute cdf with specific ‘x’ and ‘n’ values as follows : >>> cdf(UniformSum(“x”, 5), evaluate=False)(2).doit() 9/40

The argument evaluate=False prevents an attempt at evaluation of the sum for general n, before the argument 2 is passed.

References

[R693]https://en.wikipedia.org/wiki/Uniform_sum_distribution

[R694]http://mathworld.wolfram.com/UniformSumDistribution.html

`sympy.stats.` **`VonMises`**(*name*, *mu*, *k*) [source]

Create a Continuous Random Variable with a von Mises distribution.

The density of the von Mises distribution is given by

Show SymPy Live Shell

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

with $x \in [0, 2\pi]$.

Parameters: **mu** : Real number, measure of location
k : Real number, measure of concentration

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import VonMises, density, E, variance
>>> from sympy import Symbol, simplify, pprint
```

Run code block in SymPy Live

```
>>> mu = Symbol("mu")
>>> k = Symbol("k", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = VonMises("x", mu, k)
```

Run code block in SymPy Live

```
>>> D = density(X)(z)
>>> pprint(D, use_unicode=False)
     k*cos(mu - z)
    e
------------------
2*pi*besseli(0, k)
```

References

[R695]https://en.wikipedia.org/wiki/Von_Mises_distribution

[R696]http://mathworld.wolfram.com/vonMisesDistribution.html

sympy.stats. **Weibull**(*name*, *alpha*, *beta*) [source]

Create a continuous random variable with a Weibull distribution.

The density of the Weibull distribution is given by

$$f(x) := \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Parameters: **lambda** : Real number, $\lambda > 0$ a scale
k : Real number, $k > 0$ a shape

Returns: A RandomSymbol.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Weibull, density, E, variance
>>> from sympy import Symbol, simplify
```

Run code block in SymPy Live

```
>>> l = Symbol("lambda", positive=True)
>>> k = Symbol("k", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

```
>>> X = Weibull("x", l, k)
```

Run code block in SymPy Live

```
>>> density(X)(z)
k*(z/lambda)**(k - 1)*exp(-(z/lambda)**k)/lambda
```

Run code block in SymPy Live

```
>>> simplify(E(X))
lambda*gamma(1 + 1/k)
```

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> simplify(variance(X))
lambda**2*(-gamma(1 + 1/k)**2 + gamma(1 + 2/k))
```

References

[R697]https://en.wikipedia.org/wiki/Weibull_distribution

[R698]http://mathworld.wolfram.com/WeibullDistribution.html

sympy.stats. **WignerSemicircle**(*name*, *R*) [source]

Create a continuous random variable with a Wigner semicircle distribution.

The density of the Wigner semicircle distribution is given by

$$f(x) := \frac{2}{\pi R^2}\sqrt{R^2 - x^2}$$

with $x \in [-R, R]$.

Parameters: **R** : Real number, $R > 0$, the radius

Returns: A $RandomSymbol$.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import WignerSemicircle, density, E
>>> from sympy import Symbol, simplify
```

Run code block in SymPy Live

```
>>> R = Symbol("R", positive=True)
>>> z = Symbol("z")
```

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> X = WignerSemicircle("x", R)
```

Run code block in SymPy Live

```
>>> density(X)(z)
2*sqrt(R**2 - z**2)/(pi*R**2)
```

Run code block in SymPy Live

```
>>> E(X)
0
```

References

[R699]https://en.wikipedia.org/wiki/Wigner_semicircle_distribution

[R700]http://mathworld.wolfram.com/WignersSemicircleLaw.html

`sympy.stats.` **ContinuousRV**(*symbol*, *density*, *set=Interval(-oo, oo)*) [source]

Creates a Continuous Random Variable given the following:

– a symbol – a probability density function – set on which the pdf is valid (defaults to entire real line)

Returns a RandomSymbol.

Many common continuous random variable types are already implemented. This function should be needed very rarely.

Examples

Run code block in SymPy Live

```
>>> from sympy import Symbol, sqrt, exp, pi
>>> from sympy.stats import ContinuousRV, P, E
```

Run code block in SymPy Live

```
>>> x = Symbol("x")
```

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> pdf = sqrt(2)*exp(-x**2/2)/(2*sqrt(pi)) # Normal distribution
>>> X = ContinuousRV(x, pdf)
```

Run code block in SymPy Live

```
>>> E(X)
0
>>> P(X>0)
1/2
```

Interface

sympy.stats. **P**(*condition*, *given_condition=None*, *numsamples=None*, *evaluate=True*, ***kwargs*)

Probability that a condition is true, optionally given a second condition

Parameters:

condition : Combination of Relationals containing RandomSymbols

The condition of which you want to compute the probability

given_condition : Combination of Relationals containing RandomSymbols

A conditional expression. P(X > 1, X > 0) is expectation of X > 1 given X > 0

numsamples : int

Enables sampling and approximates the probability with this many samples

evaluate : Bool (defaults to True)

In case of continuous systems return unevaluated integral

Examples

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> from sympy.stats import P, Die
>>> from sympy import Eq
>>> X, Y = Die('X', 6), Die('Y', 6)
>>> P(X > 3)
1/2
>>> P(Eq(X, 5), X > 2) # Probability that X == 5 given that X > 2
1/4
>>> P(X > Y)
5/12
```

class sympy.stats. **Probability** [source]

Symbolic expression for the probability.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Probability, Normal
>>> from sympy import Integral
>>> X = Normal("X", 0, 1)
>>> prob = Probability(X > 1)
>>> prob
Probability(X > 1)
```

Integral representation:

Run code block in SymPy Live

```
>>> prob.rewrite(Integral)
Integral(sqrt(2)*exp(-_z**2/2)/(2*sqrt(pi)), (_z, 1, oo))
```

Evaluation of the integral:

Run code block in SymPy Live

```
>>> prob.evaluate_integral()
```

Show SymPy Live Shell

```
sqrt(2)*(-sqrt(2)*sqrt(pi)*erf(sqrt(2)/2) + sqrt(2)*sqrt(pi))/(4*sqrt(pi))
```

sympy.stats. **E**(*expr*, *condition=None*, *numsamples=None*, *evaluate=True*, ***kwargs*)

Returns the expected value of a random expression

Parameters:

expr : Expr containing RandomSymbols

The expression of which you want to compute the expectation value

given : Expr containing RandomSymbols

A conditional expression. E(X, X>0) is expectation of X given X > 0

numsamples : int

Enables sampling and approximates the expectation with this many samples

evalf : Bool (defaults to True)

If sampling return a number rather than a complex expression

evaluate : Bool (defaults to True)

In case of continuous systems return unevaluated integral

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import E, Die
>>> X = Die('X', 6)
>>> E(X)
```

Show SymPy Live Shell

```
7/2
>>> E(2*X + 1)
8
```

Run code block in SymPy Live

```
>>> E(X, X > 3) # Expectation of X given that it is above 3
5
```

class sympy.stats. **Expectation** [source]

Symbolic expression for the expectation.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Expectation, Normal, Probability
>>> from sympy import symbols, Integral
>>> mu = symbols("mu")
>>> sigma = symbols("sigma", positive=True)
>>> X = Normal("X", mu, sigma)
>>> Expectation(X)
Expectation(X)
>>> Expectation(X).evaluate_integral().simplify()
mu
```

To get the integral expression of the expectation:

Run code block in SymPy Live

```
>>> Expectation(X).rewrite(Integral)
Integral(sqrt(2)*X*exp(-(X - mu)**2/(2*sigma**2))/(2*sqrt(pi)*sigma), (X, -oo, oo))
```

The same integral expression, in more abstract terms:

Run code block in SymPy Live

```
>>> Expectation(X).rewrite(Probability)
```

Show SymPy Live Shell

```
Integral(x*Probability(Eq(X, x)), (x, -oo, oo))
```

This class is aware of some properties of the expectation:

Run code block in SymPy Live

```
>>> from sympy.abc import a
>>> Expectation(a*X)
Expectation(a*X)
>>> Y = Normal("Y", 0, 1)
>>> Expectation(X + Y)
Expectation(X + Y)
```

To expand the `Expectation` into its expression, use `doit()`:

Run code block in SymPy Live

```
>>> Expectation(X + Y).doit()
Expectation(X) + Expectation(Y)
>>> Expectation(a*X + Y).doit()
a*Expectation(X) + Expectation(Y)
>>> Expectation(a*X + Y)
Expectation(a*X + Y)
```

sympy.stats. **density**(*expr*, *condition=None*, *evaluate=True*, *numsamples=None*, ***kwargs*) [source]

Probability density of a random expression, optionally given a second condition.

This density will take on different forms for different types of probability spaces. Discrete variables produce Dicts. Continuous variables produce Lambdas.

Parameters:

expr : Expr containing RandomSymbols

The expression of which you want to compute the density value

condition : Relational containing RandomSymbols

Show SymPy Live Shell

A conditional expression. density(X > 1, X > 0) is density of X > 1 given X > 0

numsamples : int

Enables sampling and approximates the density with this many samples

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import density, Die, Normal
>>> from sympy import Symbol
```

Run code block in SymPy Live

```
>>> x = Symbol('x')
>>> D = Die('D', 6)
>>> X = Normal(x, 0, 1)
```

Run code block in SymPy Live

```
>>> density(D).dict
{1: 1/6, 2: 1/6, 3: 1/6, 4: 1/6, 5: 1/6, 6: 1/6}
>>> density(2*D).dict
{2: 1/6, 4: 1/6, 6: 1/6, 8: 1/6, 10: 1/6, 12: 1/6}
>>> density(X)(x)
sqrt(2)*exp(-x**2/2)/(2*sqrt(pi))
```

sympy.stats. **given**(*expr*, *condition=None*, ***kwargs*) [source]

Conditional Random Expression From a random expression and a condition on that expression creates a new probability space from the condition and returns the same expression on that conditional probability space.

Examples

Show SymPy Live Shell

Run code block in SymPy Live

```
>>> from sympy.stats import given, density, Die
>>> X = Die('X', 6)
>>> Y = given(X, X > 3)
>>> density(Y).dict
{4: 1/3, 5: 1/3, 6: 1/3}
```

Following convention, if the condition is a random symbol then that symbol is considered fixed.

Run code block in SymPy Live

```
>>> from sympy.stats import Normal
>>> from sympy import pprint
>>> from sympy.abc import z
```

Run code block in SymPy Live

```
>>> X = Normal('X', 0, 1)
>>> Y = Normal('Y', 0, 1)
>>> pprint(density(X + Y, Y)(z), use_unicode=False)
                2
       -(-Y + z)
       -----------
  ___         2
\/ 2 *e
------------------
      ____
  2*\/ pi
```

sympy.stats. **where**(*condition*, *given_condition=None*, ***kwargs*) [source]

Returns the domain where a condition is True.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import where, Die, Normal
```

Show SymPy Live Shell

```
>>> from sympy import symbols, And
```

Run code block in SymPy Live

```
>>> D1, D2 = Die('a', 6), Die('b', 6)
>>> a, b = D1.symbol, D2.symbol
>>> X = Normal('x', 0, 1)
```

Run code block in SymPy Live

```
>>> where(X**2<1)
Domain: (-1 < x) & (x < 1)
```

Run code block in SymPy Live

```
>>> where(X**2<1).set
Interval.open(-1, 1)
```

Run code block in SymPy Live

```
>>> where(And(D1<=D2 , D2<3))
Domain: (Eq(a, 1) & Eq(b, 1)) | (Eq(a, 1) & Eq(b, 2)) | (Eq(a, 2) & Eq(b, 2))
```

sympy.stats. **variance**(*X*, *condition=None*, ***kwargs*) [source]

Variance of a random expression

Expectation of (X-E(X))**2

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Die, E, Bernoulli, variance
>>> from sympy import simplify, Symbol
```

Run code block in SymPy Live

```
>>> X = Die('X', 6)
>>> p = Symbol('p')
>>> B = Bernoulli('B', p, 1, 0)
```

Run code block in SymPy Live

Show SymPy Live Shell

```
>>> variance(2*X)
35/3
```

Run code block in SymPy Live

```
>>> simplify(variance(B))
p*(1 - p)
```

class sympy.stats. **Variance** [source]

Symbolic expression for the variance.

Examples

Run code block in SymPy Live

```
>>> from sympy import symbols, Integral
>>> from sympy.stats import Normal, Expectation, Variance, Probability
>>> mu = symbols("mu", positive=True)
>>> sigma = symbols("sigma", positive=True)
>>> X = Normal("X", mu, sigma)
>>> Variance(X)
Variance(X)
>>> Variance(X).evaluate_integral()
sigma**2
```

Integral representation of the underlying calculations:

Run code block in SymPy Live

```
>>> Variance(X).rewrite(Integral)
Integral(sqrt(2)*(X - Integral(sqrt(2)*X*exp(-(X - mu)**2/(2*sigma**2))/(2*sqrt(pi)*sig
```

Integral representation, without expanding the PDF:

Run code block in SymPy Live

```
>>> Variance(X).rewrite(Probability)
```

Show SymPy Live Shell

```
-Integral(x*Probability(Eq(X, x)), (x, -oo, oo))**2 + Integral(x**2*Probability(Eq(X, x
```

Rewrite the variance in terms of the expectation

Run code block in SymPy Live

```
>>> Variance(X).rewrite(Expectation)
-Expectation(X)**2 + Expectation(X**2)
```

Some transformations based on the properties of the variance may happen:

Run code block in SymPy Live

```
>>> from sympy.abc import a
>>> Y = Normal("Y", 0, 1)
>>> Variance(a*X)
Variance(a*X)
```

To expand the variance in its expression, use `doit()`:

Run code block in SymPy Live

```
>>> Variance(a*X).doit()
a**2*Variance(X)
>>> Variance(X + Y)
Variance(X + Y)
>>> Variance(X + Y).doit()
2*Covariance(X, Y) + Variance(X) + Variance(Y)
```

`sympy.stats.` **`covariance`**(*X*, *Y*, *condition=None*, ***kwargs*) [source]

Covariance of two random expressions

The expectation that the two variables will rise and fall together

Covariance(X,Y) = E((X-E(X)) * (Y-E(Y)))

Show SymPy Live Shell

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Exponential, covariance
>>> from sympy import Symbol
```

Run code block in SymPy Live

```
>>> rate = Symbol('lambda', positive=True, real=True, finite=True)
>>> X = Exponential('X', rate)
>>> Y = Exponential('Y', rate)
```

Run code block in SymPy Live

```
>>> covariance(X, X)
lambda**(-2)
>>> covariance(X, Y)
0
>>> covariance(X, Y + rate*X)
1/lambda
```

class sympy.stats. **Covariance** [source]

Symbolic expression for the covariance.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Covariance
>>> from sympy.stats import Normal
>>> X = Normal("X", 3, 2)
>>> Y = Normal("Y", 0, 1)
>>> Z = Normal("Z", 0, 1)
>>> W = Normal("W", 0, 1)
>>> cexpr = Covariance(X, Y)
>>> cexpr
Covariance(X, Y)
```

Show SymPy Live Shell

Evaluate the covariance, X and Y are independent, therefore zero is the result:

Run code block in SymPy Live

```
>>> cexpr.evaluate_integral()
0
```

Rewrite the covariance expression in terms of expectations:

Run code block in SymPy Live

```
>>> from sympy.stats import Expectation
>>> cexpr.rewrite(Expectation)
Expectation(X*Y) - Expectation(X)*Expectation(Y)
```

In order to expand the argument, use `doit()`:

Run code block in SymPy Live

```
>>> from sympy.abc import a, b, c, d
>>> Covariance(a*X + b*Y, c*Z + d*W)
Covariance(a*X + b*Y, c*Z + d*W)
>>> Covariance(a*X + b*Y, c*Z + d*W).doit()
a*c*Covariance(X, Z) + a*d*Covariance(W, X) + b*c*Covariance(Y, Z) + b*d*Covariance(W,
```

This class is aware of some properties of the covariance:

Run code block in SymPy Live

```
>>> Covariance(X, X).doit()
Variance(X)
>>> Covariance(a*X, b*Y).doit()
a*b*Covariance(X, Y)
```

`sympy.stats.` **std**(*X*, *condition=None*, ***kwargs*)

Standard Deviation of a random expression

Show SymPy Live Shell

Square root of the Expectation of (X-E(X))**2

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Bernoulli, std
>>> from sympy import Symbol, simplify
```

Run code block in SymPy Live

```
>>> p = Symbol('p')
>>> B = Bernoulli('B', p, 1, 0)
```

Run code block in SymPy Live

```
>>> simplify(std(B))
sqrt(p*(1 - p))
```

sympy.stats. **sample**(*expr*, *condition=None*, ***kwargs*) [source]

A realization of the random expression

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Die, sample
>>> X, Y, Z = Die('X', 6), Die('Y', 6), Die('Z', 6)
```

Run code block in SymPy Live

```
>>> die_roll = sample(X + Y + Z) # A random realization of three dice
```

sympy.stats. **sample_iter**(*expr*, *condition=None*, *numsamples=oo*, ***kwargs*) [source]

Returns an iterator of realizations from the expression given a condition

Parameters: **expr: Expr**

Random expression to be realized

Show SymPy Live Shell

condition: Expr, optional

A conditional expression

numsamples: integer, optional

Length of the iterator (defaults to infinity)

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import Normal, sample_iter
>>> X = Normal('X', 0, 1)
>>> expr = X*X + 3
>>> iterator = sample_iter(expr, numsamples=3)
>>> list(iterator) # doctest: +SKIP
[12, 4, 7]
```

See also: `Sample`, `sampling_P`, `sampling_E`, `sample_iter_lambdify`, `sample_iter_subs`

Mechanics

SymPy Stats employs a relatively complex class hierarchy.

`RandomDomain`s are a mapping of variables to possible values. For example, we might say that the symbol `Symbol('x')` can take on the values $\{1, 2, 3, 4, 5, 6\}$.

class `sympy.stats.rv.` **RandomDomain** [source]

A `PSpace`, or Probability Space, combines a `RandomDomain` with a density to provide probabilistic information. For example the above domain could be enhanced by a finite density `{1:1/6, 2:1/6, 3:1/6, 4:1/6, 5:1/6, 6:1/6}` to fully define the roll of a fair die named `x`.

Show SymPy Live Shell

class sympy.stats.rv. **PSpace** [source]

A RandomSymbol represents the PSpace's symbol 'x' inside of SymPy expressions.

class sympy.stats.rv. **RandomSymbol** [source]

The RandomDomain and PSpace classes are almost never directly instantiated. Instead they are subclassed for a variety of situations.

RandomDomains and PSpaces must be sufficiently general to represent domains and spaces of several variables with arbitrarily complex densities. This generality is often unnecessary. Instead we often build SingleDomains and SinglePSpaces to represent single, univariate events and processes such as a single die or a single normal variable.

class sympy.stats.rv. **SinglePSpace** [source]

class sympy.stats.rv. **SingleDomain** [source]

Another common case is to collect together a set of such univariate random variables. A collection of independent SinglePSpaces or SingleDomains can be brought together to form a ProductDomain or ProductPSpace. These objects would be useful in representing three dice rolled together for example.

class sympy.stats.rv. **ProductDomain** [source]

class sympy.stats.rv. **ProductPSpace** [source]

The Conditional adjective is added whenever we add a global condition to a RandomDomain or PSpace. A common example would be three independent dice where we know their sum to be greater than 12.

class sympy.stats.rv. **ConditionalDomain** [source]

We specialize further into Finite and Continuous versions of these classes to represent finite (such as dice) and continuous (such as normals) random variables.

class sympy.stats.frv. **FiniteDomain** [source]

class sympy.stats.frv. **FinitePSpace** [source]

class sympy.stats.crv. **ContinuousDomain** [source]

class sympy.stats.crv. **ContinuousPSpace** [source]

Additionally there are a few specialized classes that implement certain common random variable types. There is for example a DiePSpace that implements SingleFinitePSpace and a NormalPSpace that implements SingleContinuousPSpace.

class sympy.stats.frv_types. **DiePSpace**

class sympy.stats.crv_types. **NormalPSpace**

RandomVariables can be extracted from these objects using the PSpace.values method.

As previously mentioned SymPy Stats employs a relatively complex class structure. Inheritance is widely used in the implementation of end-level classes. This tactic was chosen to balance between the need to allow SymPy to represent arbitrarily defined random variables and optimizing for common cases. This complicates the code but is structured to only be important to those working on extending SymPy Stats to other random variable types.

Users will not use this class structure. Instead these mechanics are exposed through variable creation functions Die, Coin, FiniteRV, Normal, Exponential, etc.... These build the appropriate SinglePSpaces and return the corresponding RandomVariable. Conditional and Product spaces are formed in the natural construction of SymPy expressions and the use of interface functions E, Given, Density, etc....

sympy.stats. **Die**()

sympy.stats. **Normal**()

There are some additional functions that may be useful. They are largely used internally.

sympy.stats.rv. **random_symbols**(*expr*) [source]

Returns all RandomSymbols within a SymPy Expression.

sympy.stats.rv. **pspace**(*expr*) [source]

Returns the underlying Probability Space of a random expression.

For internal use.

Examples

Run code block in SymPy Live

```
>>> from sympy.stats import pspace, Normal
>>> from sympy.stats.rv import IndependentProductPSpace
>>> X = Normal('X', 0, 1)
>>> pspace(2*X + 1) == X.pspace
True
```

sympy.stats.rv. **rs_swap**(*a*, *b*) [source]

Build a dictionary to swap RandomSymbols based on their underlying symbol.

i.e. if `X = ('x', pspace1)` and `Y = ('x', pspace2)` then `X` and `Y` match and the key, value pair `{X:Y}` will appear in the result

Inputs: collections a and b of random variables which share common symbols Output: dict mapping RVs in a to RVs in b

Show SymPy Live Shell