Module statistics_lib.special_functions Reference

Scope

This document describes the intended usage, design and implementation of the functionality implemented in the module **distribution_classes** of the library **statistics_lib**. The API reference is also provided.

The concerted functional elements are functions:

- *permutation*
- *combination*
- *log_beta*
- *beta*
- *inv_erf*
- *lower_gamma*
- *log_lower_gamma*
- *lower_gamma_reg*
- *upper_gamma*
- *log_upper_gamma*
- *upper_gamma_reg*
- *incomplete_beta*
- *log_incomplete_beta*
- *incomplete_beta_reg*

Intended Use and Functionality

The main purpose of this module is to implement the special mathematical functions required for the calculation of the probability density, cummulative probability density and quantile functions of the continuous and discrete distributions (see DE002 document). However, those special mathematical functions can be helpful in a number of ways not in the content of the random number distributions, since they arise often during integration of power and exponential functions.

The implemented special mathematical functions are listed below.

The *inverse error function*

$$\mathbf{erf}^{-1}(-1 < y < 1) \rightarrow x;:;\mathbf{erf}(x) = y;\mathbf{where};\mathbf{erf}(x) = \mathbf{sign}(x) \times \frac{2}{\sqrt{\pi}} \int_0^{|x|} e^{-t^2} dt$$

The *lower* and *upper incomplete gamma functions*

$$ \gamma(x > 0,y \geq 0) = \int_0^{y} {t^{x-1} e^{-t} dt} \newline

\Gamma(x > 0 ,y \geq 0) = \int_y^{\infin} {t^{x-1} e^{-t} dt} $$

and their *regularized* versions

$$ P(x > 0,y \geq 0) = \frac{\gamma(x,y)} {\Gamma(x)} \newline

Q(x > 0,y \geq 0) = \frac{\Gamma(x,y)} {\Gamma(x)} $$

where

$$\Gamma(x > 0) = \int_0^{\infin} t^{x-1} e^{-t} dt$$

is the normal *gamma function*, which is implemented in the Standard Python Library's module *math*.

The *beta function*

$$B(x > 0, y > 0) = \int_0^1 t^{x-1} (1-t)^{y-1} dt = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$$

The *incomplete beta function*

$$B(0 \leq z \leq 1; x > 0, y > 0) = \int_0^z t^{x-1} (1-t)^{y-1} dt$$

and its *regularized* version

$$I_z(x,y) = \frac{B(z;x,y)}{B(x,y)}$$

Design and Implementation

The implemented functions employ input data sanity checks in order to avoid unpredictable run-time arithmetics related exceptions. Basically, the input arguments must be real numbers (integer or floating point types **int** and **float** respectively), whith their values being in the approriate range, for which the function is defined. If the input data sanity check fails either **UT_TypeError** (sub-classes **TypeError**) or **UT_ValueError** (sub-classes **ValueError**) exception is raised. These custom exceptoions are defined in the module *base_exceptions* of the library *introspection_lib*.

These sanity checks are implemented as '*private*' helper functions, which are outside the scope of this document, since multiple functions are supposed to perform exactly the same checks.

The *inverse error function* is calculated using AS241 algorithm (see DE003 document), which is based on the 7th-7th power *rational function* approximation with three different approximations for the 'core', 'tails' and 'far tails' regions.

The *incomplete gamma functions* are implemented via power series (for y < x + 1) and continued fractions (for y >= x + 1) calculations using the algorithms provided in the *Numerical Recipes in C: The Art of Scientific Computing* book (see DE003 document).

The *incomplete beta functions* are implemented via computation of the continued fractions using the algorithms provided in the *Numerical Recipes in C: The Art of Scientific Computing* book (see DE003 document).

The actual computation of the power series and continued fractions as well as the evaluation of the ration functions and polynomial is delegated to the '*private*' helper functions, which are outside the scope of this document. Thus, the '*public*' functions described in this document are more like *process flow manager*,

ensuring the application of the input data sanity checks and selection of the appropriate calculation algorithm based on the values of the passed arguments.

In addition, two combinatorics functions are defined in the module: *permutation*() and *combination*(). In the case of the Python interpreter version 3.8 or newer they simply wrap the calls to the Standard Python Library functions *math.perm*() and *math.comb*() respectively. For the earlier versions of the Python interpreter they implement calculation of the respective factorials ratios using iterative multiplication, taking advantage of Python's support for the arbitrary length integers.

API Reference

Functions

permutation(n, k)

Signature:

int >= 0, int >= 0 -> int > 0

Args:

- *n*: **int** >= 0; the total number of objects available
- *k*: **int** >= 0; the number of objects taken

Raises:

- **UT_TypeError**: either of the arguments is not integer
- **UT_ValueError**: either of the arguments is negative, or k > n

Description:

Calculates 'n permute k' value, i.e. $A_n^k \equiv {}_nP_k \equiv P(n,k) = \frac{n!}{(n-k)!}$.

combination(n, k)

Signature:

int >= 0, int >= 0 -> int > 0

Args:

- *n*: **int** >= 0; the total number of objects available
- *k*: **int** >= 0; the number of objects taken

Raises:

- **UT_TypeError**: either of the arguments is not integer
- **UT_ValueError**: either of the arguments is negative, or k > n

Description:

Calculates 'n chose k' value, i.e. $C_n^k \equiv \binom{n}{k} = \frac{n!}{(n-k)!k!}$.

log_beta(x, y)

Signature:

int > 0 OR float > 0, int > 0 OR float > 0 -> float

Args:

- *x*: **int** > 0 OR **float** > 0; any real number first argument
- *y*: **int** > 0 OR **float** > 0; any real number second argument

Raises:

- **UT_TypeError**: either of the arguments is not integer or float
- **UT_ValueError**: either of the arguments is not positive

Description:

Calculates the value of the natural logarithm of beta function *ln(B(x, y))*.

beta(x, y)

Signature:

int > 0 OR float > 0, int > 0 OR float > 0 -> float > 0

Args:

- *x*: **int** > 0 OR **float** > 0; any real number first argument
- *y*: **int** > 0 OR **float** > 0; any real number second argument

Raises:

- **UT_TypeError**: either of the arguments is not integer or float
- **UT_ValueError**: either of the arguments is not positive

Description:

Calculates the value of the beta function *B(x, y)*.

inv_erf(x)

Signature:

int = 0 OR -1 < float < 1 -> float

Raises:

- **UT_TypeError**: the argument is not integer or float
- **UT_ValueError**: the argument is not in the range (-1, 1)

Description:

Calculates the value of the inverse error function $\texttt{erf}^{-1}(x)$.

lower_gamma(x, y)

Signature:

int > 0 OR float > 0, int >= 0 OR float > 0 -> float >= 0

Args:

- *x*: **int** > 0 OR **float** > 0; power parameter of the function
- *y*: **int** >= 0 OR **float** > 0; the integeral boundary parameter of the function

Raises:

- **UT_TypeError**: either of the arguments is neither integer nor float
- **UT_ValueError**: either of the arguments is negative, OR the first argument is zero
- **Exception**: maximum number of iteration is reached

Description:

Calculates the value of the lower incomplete gamma function $\gamma(x, y)$.

log_lower_gamma(x, y)

Signature:

int > 0 OR float > 0, int > 0 OR float > 0 -> float

Args:

- *x*: **int** > 0 OR **float** > 0; power parameter of the function
- *y*: **int** > 0 OR **float** > 0; the integeral boundary parameter of the function

Raises:

- **UT_TypeError**: either of the arguments is neither integer nor float
- **UT_ValueError**: either of the arguments is negative or zero
- **Exception**: maximum number of iteration is reached

Description:

Calculates the value of the natural logarithm of the lower incomplete gamma function $\ln(\gamma(x, y))$.

lower_gamma_reg(x, y)

Signature:

int > 0 OR float > 0, int >= 0 OR float > 0 -> 0 <= float < 1

Args:

- *x*: **int** > 0 OR **float** > 0; power parameter of the function
- *y*: **int** >= 0 OR **float** > 0; the integeral boundary parameter of the function

Raises:

- **UT_TypeError**: either of the arguments is neither integer nor float
- **UT_ValueError**: either of the arguments is negative, OR the first argument is zero
- **Exception**: maximum number of iteration is reached

Description:

Calculates the value of the regularized lower incomplete gamma function $P(x, y) = \frac{\gamma(x,y)}{\Gamma(x)}$.

upper_gamma(x, y)

Signature:

int > 0 OR float > 0, int >= 0 OR float > 0 -> float > 0

Args:

- *x*: **int** > 0 OR **float** > 0; power parameter of the function
- *y*: **int** >= 0 OR **float** > 0; the integeral boundary parameter of the function

Raises:

- **UT_TypeError**: either of the arguments is neither integer nor float
- **UT_ValueError**: either of the arguments is negative, OR the first argument is zero
- **Exception**: maximum number of iteration is reached

Description:

Calculates the value of the upper incomplete gamma function $\Gamma(x, y)$.

log_upper_gamma(x, y)

Signature:

int > 0 OR float > 0, int >= 0 OR float > 0 -> float

Args:

- *x*: **int** > 0 OR **float** > 0; power parameter of the function
- *y*: **int** >= 0 OR **float** > 0; the integeral boundary parameter of the function

Raises:

- **UT_TypeError**: either of the arguments is neither integer nor float
- **UT_ValueError**: either of the arguments is negative, OR the first argument is zero
- **Exception**: maximum number of iteration is reached

Description:

Calculates the value of the natural logarithm of the upper incomplete gamma function $\ln(\Gamma(x, y))$.

upper_gamma_reg(x, y)

Signature:

int > 0 OR float > 0, int >= 0 OR float > 0 -> 0 < float <= 1

Args:

- *x*: **int** > 0 OR **float** > 0; power parameter of the function
- *y*: **int** >= 0 OR **float** > 0; the integeral boundary parameter of the function

Raises:

- **UT_TypeError**: either of the arguments is neither integer nor float
- **UT_ValueError**: either of the arguments is negative, OR the first argument is zero
- **Exception**: maximum number of iteration is reached

Description:

Calculates the value of the regularized upper incomplete gamma function $Q(x, y) = \frac{\Gamma(x,y)}{\Gamma(x)}$.

incomplete_beta(z, x, y)

Signature:

0 <= int <= 1 OR 0 < float < 1, int > 0 OR float > 0, int > 0 OR float > 0 -> float >= 0

Args:

- *z*: 0 <= **int** <= 1 OR 0 < **float** < 1; the integeral boundary parameter of the function
- *x*: **int** > 0 OR **float** > 0; the first power parameter of the function
- *y*: **int** > 0 OR **float** > 0; the second power parameter of the function

Raises:

- **UT_TypeError**: either of the arguments is neither integer nor float
- **UT_ValueError**: the first argument is not in the range [0, 1], OR either of the other arguments is zero or negative
- **Exception**: maximum number of iteration is reached

Description:

Calculates the value of the incomplete beta function $B(z; x, y)$.

log_incomplete_beta(z, x, y)

Signature:

int = 1 OR 0 < float < 1, int > 0 OR float > 0, int > 0 OR float > 0 -> float

Args:

- *z*: **int** = 1 OR 0 < **float** < 1; the integeral boundary parameter of the function
- *x*: **int** > 0 OR **float** > 0; the first power parameter of the function
- *y*: **int** > 0 OR **float** > 0; the second power parameter of the function

Raises:

- **UT_TypeError**: either of the arguments is neither integer nor float
- **UT_ValueError**: the first argument is not in the range (0, 1], OR either of the other arguments is zero or negative
- **Exception**: maximum number of iteration is reached

Description:

Calculates the value of the natural logarithm of the incomplete beta function $\ln(B(z;x,y))$.

incomplete_beta_reg(z, x, y)

Signature:

0 <= int <= 1 OR 0 < float < 1, int > 0 OR float > 0, int > 0 OR float > 0 -> 0 <= float <= 1

Args:

- *z*: 0 <= **int** <= 1 OR 0 < **float** < 1; the integeral boundary parameter of the function
- *x*: **int** > 0 OR **float** > 0; the first power parameter of the function
- *y*: **int** > 0 OR **float** > 0; the second power parameter of the function

Raises:

- **UT_TypeError**: either of the arguments is neither integer nor float
- **UT_ValueError**: the first argument is not in the range [0, 1], OR either of the other arguments is zero or negative
- **Exception**: maximum number of iteration is reached

Description:

Calculates the value of the regularized incomplete beta function $I_z(x,y) = \frac{B(z;x,y)}{B(x,y)}$.