Module statistics_lib.base_functions Reference

Scope

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

This module provides functions for calculation of:

- 1D statstics
  - Arithmetic mean - *GetMean*()
  - Variance with the Bessel correction (sample variance) and without the correction (population variance) - *GetVarianceS*() and *GetVarianceP*()
  - Standard deviation with the Bessel correction (sample deviation) and without the correction (population deviation) - *GetStdevS*() and *GetStdevP*()
  - Standard error of the mean of a data set (as population - i.e. without Bessel correction) - *GetSE*()
  - Skewness with the Bessel correction (sample skewness) and without the correction (population skewness) - *GetSkewnessS*() and *GetSkewnessP*()
  - Excess kurtosis with the Bessel correction (sample kurtosis) and without the correction (population kurtosis) - *GetKurtosisS*() and *GetKurtosisP*()
  - Generic Nth moment of a data set distributiion (as population - i.e. without Bessel correction) - both central and non-central variants as well as normalized and not normalized - *GetMoment*()
  - 'Full' standard error of the mean of a data set as population and including the individual data points 'measurement uncertainties' - *GetFullSE*()
- 2D statics
  - Covariance of a 2D data set (as population - i.e. without Bessel correction) - *GetCovariance*()
  - Pearson's coefficient of correlation *r* of a 2D data set (as population - i.e. without Bessel correction) - *GetPearsonR*()
  - Generic Nth-Mth moment of a 2D data set distributiion (as population - i.e. without Bessel correction) - both central and non-central variants as well as normalized and not normalized - *GetMoment2*()

Intended Use and Functionality

The module should provide functionality to calcualte specific statistic moments of 1D or 2D data sets, which could be used as the back-end for higher abstraction functionality within this library as well as called from another libraries or modules by an end user.

The unique feature of this module is that the statistic properties can be computed not only on sequences of real numbers (integers or floating point), but the measurements with uncertainties can also be included into the data set. A measurement with uncertainty is, basically, a tuple of two real numbers (x_i, z_i), where x_i represents the 'mean' of the measured value, and $z_i \geq 0$ is the measurement uncertainty. Thus, any real number x_i can also be represented as a measurement with *zero* uncertainty, i.e. $(x_i, 0)$.

The functions defined in this module can accept an arbirary mixture of real numbers and such measurements with uncertainties tuples - as a generic sequence argument - and treat it as a list or tuple of real numbers (ignoring the measurement uncertainties in all but one function).

The aritmetic mean of the data distribution is calculated as the first non-central moment

$$ ☑ = \frac{\sum\limits_{i=1}^{N}{x_i}}{N}$$

The generic K-th non-central moment is

$$E[X^K] = \frac{\sum\limits_{i=1}^{N}{x_i^K}}{N}$$

For instance, the mean of squares is the 2nd non-central moment

$$< X^2 >= E[X^2] = \frac{\sum\limits_{i=1}^{N}{x_i^2}}{N}$$

The generic K-th central moment is

$$E[(X-)^K] = \frac{\sum\limits_{i=1}^{N}{(x_i -)^K}}{N}$$

For instance, the variance is the second central moment

$$Var(X) = E[(X-)^2] = \frac{\sum\limits_{i=1}^{N}{(x_i -)^2}}{N}$$

The standard deviation (population) is computed as the square root of the variance

$$\sigma(X) = \sqrt{Var(X)}$$

The standard error of the mean is

$$SE(X) = \frac{\sigma(X)}{\sqrt{N}} = \frac{\sqrt{\sum\limits_{i=1}^{N}{(x_i -)^2}}}{N}$$

However, for the measurements with the associated individual uncertainties these uncertainties must be taken into account - the so called 'full standard error'

$$SE_F(X) = \frac{\sqrt{\sum\limits_{i=1}^{N}{(x_i -)^2 + \left(\sum\limits_{i=1}^{N}{z_i^2}\right)}}}{N}$$

Note that, generally, $SE_F(X) \geq SE(X)$, and $SE_F(X) = SE(X)$ only if the data set consists of only real numbers, or all associated individual uncertainties are zero. This functionality is added for the implementation of the standard error propagation model, specifically - averaging of multiple measurements.

It is also possible to define the normalized non-central moment as

$$E\left[\left(\frac{X}{\sigma(X)}\right)^K\right] = \frac{\sum\limits_{i=1}^{N}{x_i^K}}{N \times \sigma(X)^K}$$

and normalized central moment as

$$E\left[\left(\frac{X-}{\sigma(X)}\right)^K\right] = \frac{\sum\limits_{i=1}^{N}{(x_i -)^K}}{N \times {\sigma(X)}^K}$$

The two special cases are skewness and kurtosis:

$$Skew(X) = E\left[\left(\frac{X-}{\sigma(X)}\right)^3\right] = \frac{\sum\limits_{i=1}^{N}{(x_i -)^3}}{N \times {\sigma(X)}^3}$$

$$KurtEx(X) = E\left[\left(\frac{X-}{\sigma(X)}\right)^4\right] - 3 = \frac{\sum\limits_{i=1}^{N}{(x_i -)^3}}{N \times {\sigma(X)}^4} - 3$$

Note, that the *excess kurtosis* is, in fact, calculated.

In general, especially in the cases of small samples (N ~ 10) then mean and standard deviation calculated from a sample are not, generally speaking, the same as the *true* mean and standard deviation of the distribution of the entire population, from which the sample is taken: $ \neq \mu$and$\sigma(X) \neq \sigma_X$. The formulas above are *biased* estimators of the *true* moments based on and\sigma(X) $insteadof$\mu$and$\sigma_X$. The Bessel correction provides *unbiased* (but not *per se* accurate) estimators, known as *sample* variance, standard deviation, skewness and (excess) kurtosis.

$$Var_S(X) = \frac{\sum\limits_{i=1}^{N-1}{(x_i -)^2}}{N -1} = \frac{N}{N-1} \times Var(S)\approx E[(X-\mu)^2]$$

$$\sigma_S(X) = \sqrt{Var_S(X)} = \sqrt{\frac{N}{N-1}} \times \sigma_X \approx \sigma_X$$

$$Skew_S(X) = \frac{\sum\limits_{i=1}^{N}{(x_i -)^3}}{{N \times \sigma(X)}^3} \times \frac{\sqrt{N \times (N-1)}}{N-2} = Skew(X) \times \frac{\sqrt{N \times (N-1)}}{N-2} \approx E\left[\left(\frac{X-\mu}{\sigma_X}\right)^3\right]$$

$$KurtEx_S(X) = \frac{\sum\limits_{i=1}^{N}{(x_i -)^4}}{{N \times\sigma(X)}^4} \times \frac{(N-1) \times (N+1)}{(N-2) \times (N-3)} - 3 \times \frac{\left(N-1\right)^2}{(N-2) \times (N-3)} \approx E\left[\left(\frac{X-\mu}{\sigma_X}\right)^4\right] - 3$$

$$KurtEx_S(X) = \frac{(N-1) \times ((N+1) \times KurtEx(X) + 6)}{(N-2) \times (N-3)}$$

Finally, the module implements calculation of the basic 2D statistic properties ignoring the individual uncertainties and Bessel correction, e.g. covariance

$$Cov(X, Y) = E[(X-) \times (Y -)] = \frac{\sum\limits_{i=1}^{N}{(x_i -) \times (y_i -)}}{N}$$

Non-central generic cross-moment

$$E[X^K \times Y^M] = \frac{\sum\limits_{i=1}^{N} x_i^K \times y_i^M}{N}$$

Central generic cross-moment

$$E[(X-)^K \times (Y-)^M] = \frac{\sum\limits_{i=1}^{N}{(x_i -)^K \times (y_i -)^M}}{N}$$

Non-central normalized generic cross-moment

$$E[\left(\frac{X}{\sigma_x}\right)^K \times \left(\frac{Y}{\sigma_y}\right)^M] = \frac{\sum_{i=1}^{N} x_i^K \times y_i^M}{N \times \sigma_x^K \times \sigma_y^M}$$

Central normalized generic cross-moment

$$E[\left(\frac{X - \langle X \rangle}{\sigma_x}\right)^K \times \left(\frac{Y - \langle Y \rangle}{\sigma_y}\right)^M] = \frac{\sum_{i=1}^{N} (x_i - \langle X \rangle)^K \times (y_i - \langle Y \rangle)^M}{N \times \sigma_x^K \times \sigma_y^M}$$

And the Pearson's coefficient of correlation

$$r = \frac{Cov(X,Y)}{\sqrt{Var(X) \times Var(Y)}} = E[\left(\frac{X - \langle X \rangle}{\sigma_x}\right) \times \left(\frac{Y - \langle Y \rangle}{\sigma_y}\right)]$$

Design and Implementation

The required functionality is implemented as a number of functions. The table below provides the correspondence between the implemented function names, their functionality and the corresponding spreadsheet function (MS Excel or LibreOffice Calc):

Function name	Functionality	Spreadsheet
GetMean	E[X]	AVERAGE
GetMeanSqrSE	see text above	N/A
GetVarianceP	E[(X-<X>)^2]	VAR.P
GetVarianceS	N*E[(X-<X>)^2]/(N-1)	VAR.S
GetStdevP	SQRT(E[(X-<X>)^2])	STDEV.P
GetStdevS	SQRT(N*E[(X-<X>)^2]/(N-1))	STDEV.S
GetSE	SQRT(E[(X-<X>)^2]/N)	N/A
GetFullSE	see text above	N/A
GetSkewnessP	E[(X-<X>)^3] / E[(X-<X>)^2]^3/2	SKEWP
GetSkewnessS	see text above	SKEWS
GetKurtosisP	(E[(X-<X>)^4] / E[(X-<X>)^2]^2) - 3	KURT
GetKurtosisS	see text above	N/A
GetMoment	see text above	N/A
GetCovariance	E[(X-<X>)*(Y-<Y>)]	COVARIANCE.P
GetMoment2	see text	N/A
GetPearsonR	E[(X-<X>)*(Y-<Y>)]/SQRT(E[(X-<X>)^2]*E[(Y-<Y>)^2])	PEARSON

All these functions are designed to accept a generic sequence (or two sequences) containing only real number (integer or floating point) and / or instances of a class implementing measurements with uncertainty, which is expected to be API compatible with the **phyqus_lib.base_classes.MeasuredValue** class, i.e. to have fields / properties *Value* and *SE*. A different type of the input data set results in a **TypeError** typed exception. An empty data sequence, or unequal length X- and Y-data sequences result in **ValueError** typed exception.

Since the described input data sanity check is a routine common for all functions, it is implemented in special, helper functions (separate for 1D and 2D statistic functions), which are not intended to be used outside this module, and they are not discussed in the 'API Reference' section.

The checks on the argument(s) type, i.e. the input data being a sequence, is performed as 'IS A' test, but using the generic, ABC **collections.abc.Sequence**. The elements of a sequence are checked 2-ways (with OR conjunction):

- 'IS A' test on being instance of **int** or **float** type
- 'HAS A' test on having *Value* and *SE* attributes

Apart from the data sanity checks these helper functions also 'unify' the input, i.e. they return the same length sequence of only real numbers, into which each real number from the input sequence is copied, and for the measurements with uncertainty the value its attribute *Value* is copied instead.

There is also the third helper function, which extracts the measurements uncertainties from a mixed sequence. It returns the same length sequence of only real numbers, into which for each real number from the input sequence a zero is placed, and for the measurements with uncertainty the value its attribute *SE* is copied instead. Note, that in this case 'HAS A' check is based on the presence of *SE* attribute.

In case of an improper input data these helper functions raise **UT_TypeError** and **UT_ValueError** exceptions defined in *introspection_lib.base_exceptions* module, and explicitely indicate to skip the 2 innermost frames from the traceback analysis. E.g., consider the following code, there the raised exception is caught.

```python
import statistics_lib.base_functions as bsf
...
#do something
...
#make faulty call
try:
    Mean = bsf.GetMean([1, 2.3, '1'])
except (TypeError, ValueError) as err:
    print(err.__class__.__name__, ':', err)
    print(err.Traceback.Info)
...
```

In this case, the traceback printed by the statement `print(err.Traceback.Info)` will end in the frame of the call `Mean = bsf.GetMean([1, 2.3, '1'])`, thus hiding the implementation details, i.e. where inside *GetMean*() function the helper function was called, and where inside the helper function the exception was actually raised.

Note that if the raised exception is not caught, and it has propagated to the interactive console, the printed (system) traceback will include all frames - for the *GetMean*() and the helper functions, with the helper functions being the innermost (last) frames.

The functions implementing Bessel corrected calculations: *GetStdevS*(), *GetVarianceS*(), *GetSkewnessS*() and *GetKurtosisS*() also checks that the length of the sequence is not smaller than the reduced 'degrees of freedom' number (2, 3 and 4 respectively), and they raise **UT_ValueError** otherwise, indicating to skip the innermost frame from the traceback analysis.

The functions *GetMoment*() and *GetMoment2*() also check that the passed required power(s) of the moment to be calculated is (are) positive integer(s), raising **UT_TypeError** or **UT_ValueError** otherwise, and they indicate to skip the innermost frame from the traceback analysis.

All 2D statistics functions check that the both sequences (X and Y data) have the same length, otherwise **UT_ValueError** exception is raised.

Thus, using *try...except* paradigm, the traceback analysis provided by the custom exceptions allows 'hiding' of the implementation details, which facilites the debugging procedure, since the last (innermost) frame will point to the 'offending' call.

The actual calculations performed by the provided functions are, in principle, reduced to finding a specific central or non-central, normalized or non-normalized (cross-) moment of the input data distribution(s), which is implemented using the built-in *pow*() function (since all powers in moments are positive integers) and the built-in *sum*() function.

Special, edge-cases

The special case is the *constant value* sequence, i.e. such where all elements are the same number, which includes a sequence of one element as a partial case. The following rules are applied:

- Central K-th moment, normalized or not normalized is 0, which rule also applies to
  - standard error of the mean and population variance, standard deviation, skewness and kurtosis
- Non-central not normalized K-th moment is the (any) element to the K-th power
- Non-cental normalized K-th moment is not defined, **UT_ValueError** exception is raised
- Central K-th / N-th cross-moment, normalized or not normalized is 0, which rule also applies to
  - covariance, but not to Person's correlation coefficient r
- Non-central not normalized K-th / N-th cross-moment is the (any) element of X data to the K-th power times the (any) element of Y data to N-th power
- Non-cental normalized K-th / N-th cross-moment is not defined, **UT_ValueError** exception is raised

The calculation of the Person's correlation coefficient r treats such edge cases differently:

- If only one of the data sequences is contant, r = 0
- If both sequences are constant, r = 1

API conventions

The functions *GetMoment*() and *GetMoment2*() calculating the N-th moment and N-th / M-th cross-moment *by default* compute the non-central not normalized moments. This behaviour can be modified using two keyword-only boolean arguments *IsCentral* and *IsNormalized*. The both defaults to **False**; by passing *IsCentral* = **True** these functions are switched to the calculation of the central moments, and by passing *IsNormalized* = **True** the normalized moments are calculated. The both keyword arguments can be used simultaneously.

All defined functions also support the keyword-only arguments *SkipFrames* (defaults to 1, should be a positive integer) and *DoCheck* (defaults to **True**, should be any data type, which can be used in the Boolean context). Both these arguments are used for the optimization purposes, and they are not supposed to be used by the end-clients of the module directly.

The *SkipFrames* argument is used for truncation of the exception traceback (see module *base_exceptions* in the library **introspection_lib**) for the purposes of debuging and errors logging, when it is more important to find the place of the improper input data when to show all the path to where the exception is raised. Consider the previous example

```
import statistics_lib.base_functions as bsf
...
#do something
...
#make faulty call
try:
    Mean = bsf.GetMean([1, 2.3, '1'])
except (TypeError, ValueError) as err:
    print(err.__class__.__name__, ':', err)
    print(err.Traceback.Info)
...
```

In this case the innermost frame shown in the traceback will be `Mean = bsf.GetMean([1, 2.3, '1'])`, whereas the internal process within th function *GetMean* will not be shown. If the same function is called inside another function as:

```
import statistics_lib.base_functions as bsf

def Wrapper(Data)
  #do something
  return bsf.GetMean(Data)
...
#make faulty call
try:
    Mean = Wrapper([1, 2.3, '1'])
except (TypeError, ValueError) as err:
    print(err.__class__.__name__, ':', err)
    print(err.Traceback.Info)
...
```

the shown frames will be `Mean = Wrapper([1, 2.3, '1'])` -> `return bsf.GetMean(Data)`. However, this behavior can be modified as:

```
import statistics_lib.base_functions as bsf

def Wrapper(Data):
  #do something
  return bsf.GetMean(Data, SkipFrames = 2)
...
#make faulty call
try:
    Mean = Wrapper([1, 2.3, '1'])
except (TypeError, ValueError) as err:
    print(err.__class__.__name__, ':', err)
    print(err.Traceback.Info)
...
```

in which case the traceback will be `Mean = Wrapper([1, 2.3, '1'])`. Such nesting (call chain) can be extended further as:

```
import statistics_lib.base_functions as bsf

def Outer(Data):
  #do something
  Temp = Inner(Data)
  #do something

def Inner(Data):
  #do something
  return bsf.GetMean(Data, SkipFrames = 3)
...
#make faulty call
try:
    Mean = Wrapper([1, 2.3, '1'])
except (TypeError, ValueError) as err:
    print(err.__class__.__name__, ':', err)
    print(err.Traceback.Info)
...
```

in which case the traceback will be `Mean = Outer([1, 2.3, '1'])`.

The second keyword-only argument *DoCheck* is used only for the optimization as in avoiding redundant data sanity checks and convertion (i.e. extraction of the 'mean' values from a mixed sequence of real numbers and measurements with uncertainty). For instance, if the input data is quaranteed to be a sequence of only real numbers, the input data sanity check and conversion is not needed. Basically, if the functions are called from other functions or class methods, which already sanitized the data, it is better to pass *DoCheck* = **False**, which is beneficial for the calculation speed.

Note, *DoCheck* = **False** should be passed only if the input is quaranteed to be a sequence of

- only real numbers (**int** or **float**) - for all functions except *GetMeanSqrSE*() and *GetFullSE*()

- a mixture of real numbers and 'measurements with uncertainty' - only for *GetMeanSqrSE*() and *GetFullSE*() functions

API Reference

Functions

All functions implemented in this module have calculation time complexity of O(N).

GetMean(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated mean value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is empty, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the arithmetic mean of a mixed sequence of real numbers and the measurements with uncertainty.

GetVarianceP(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated variance value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is empty, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the population variance of a mixed sequence of real numbers and the measurements with uncertainty.

GetStdevP(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated standard deviation value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is empty, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the population standard deviation of a mixed sequence of real numbers and the measurements with uncertainty.

GetVarianceS(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated variance (with Bessel correction) value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is less then 2 elements long, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the sample variance with Bessel correction of a mixed sequence of real numbers and the measurements with uncertainty.

GetStdevS(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated standard deviation (with Bessel correction) value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is less than 2 elements long, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the sample standard deviation with Bessel correction of a mixed sequence of real numbers and the measurements with uncertainty.

GetSE(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated standard error value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is empty, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the standard error of the mean of a mixed sequence of real numbers and the measurements with uncertainty.

GetMeanSqrSE(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated mean squared uncertainty value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is empty, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the mean squared uncertainty of a mixed sequence of real numbers and the measurements with uncertainty.

GetFullSE(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated full uncertainty of the mean value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is empty, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the full uncertainty of the mean of a mixed sequence of real numbers and the measurements with uncertainty.

GetMoment(Data, Power, *, IsCentral = False, IsNormalized = False, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue), int > 0/, *, bool, bool, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *Power*: **int** > 0; the moment power
- *IsCentral*: (keyword) **bool**; is the central moment is to be calculated, defaults to False
- *IsNormalized*: (keyword) **bool**; is the normalized moment is to be calculated, defaults to False
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated full uncertainty of the mean value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR the moment power is not an integer number, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is empty, OR the moment power is zero or negative integer, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the generic N-th moment of a mixed sequence of real numbers and the measurements with uncertainty, which can be central or non-central, normailized or not normalized, depending on the values of the keyword arguments.

GetSkewnessP(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated skewness value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is empty, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the population skewness of a mixed sequence of real numbers and the measurements with uncertainty.

GetSkewnessS(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated skewness (with Bessel correction) value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is less than 3 elements long, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the sample skewness with the Bessel correction of a mixed sequence of real numbers and the measurements with uncertainty.

GetKurtosisP(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated excess kurtosis value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is empty, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the population excess kurtosis of a mixed sequence of real numbers and the measurements with uncertainty.

GetKurtosisS(Data, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *Data*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty'
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated excess kurtosis (with Bessel correction) value

Raises:

- **UT_TypeError**: mandatory argument is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: passed mandatory sequence is less than 4 elements long, OR any keyword argument is of the proper type but unacceptable value

Description:

Calculates the sample excess kurtosis with the Bessel correction of a mixed sequence of real numbers and the measurements with uncertainty.

GetCovariance(DataX, DataY, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue), seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *DataX*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty' as X data sequence
- *DataY*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty' as Y data sequence
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated covariance value

Raises:

- **UT_TypeError**: any of mandatory data arguments is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: any of the passed mandatory sequence is empty, OR any keyword argument is of the proper type but unacceptable value, OR the X and Y sequences are of different length

Description:

Calculates the covariance of the paired mixed sequences of real numbers and the measurements with uncertainty.

GetMoment2(DataX, DataY, PowerX, PowerY *, IsCentral = False, IsNormalized = False, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue), seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *DataX*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty' as X data sequence
- *DataY*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty' as Y data sequence
- *PowerX*: **int** > 0; the moment power of X
- *PowerY*: **int** > 0; the moment power of Y
- *IsCentral*: (keyword) **bool**; is the central moment is to be calculated, defaults to False
- *IsNormalized*: (keyword) **bool**; is the normalized moment is to be calculated, defaults to False
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated moment value

Raises:

- **UT_TypeError**: any of mandatory data arguments is not a sequence of real numbers or measurements with uncertainty OR any moment power is not an integer number, OR any keyword argument is of improper type
- **UT_ValueError**: any of the passed mandatory sequence is empty, OR any moment power is zero or negative integer, OR any keyword argument is of the proper type but unacceptable value, OR the X and Y sequences are of different length

Description:

Calculates the generic N-th / M-th moment of the paired mixed sequences of real numbers and the measurements with uncertainty, which can be central or non-central, normailized or not normalized, depending on the values of the keyword arguments.

GetPearsonR(DataX, DataY, *, SkipFrames = 1, DoCheck = True)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue), seq(int OR float OR phyqus_lib.base_classes.MeasuredValue)/, *, int > 0, bool/ -> int OR float

Args:

- *DataX*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty' as X data sequence
- *DataY*: **seq**(**int**0 OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); a sequence of real numbers or 'measurements with uncertainty' as Y data sequence
- *SkipFrames*: (keyword) **int** > 0; how many frames to hide in the exception traceback, defaults to 1
- *DoCheck*: (keyword) **bool**; flag if to perform the input data sanity check and convert the mixed sequence into a list of only real numbers

Returns:

int OR **float**: the calculated covariance value

Raises:

- **UT_TypeError**: any of mandatory data arguments is not a sequence of real numbers or measurements with uncertainty, OR any keyword argument is of improper type
- **UT_ValueError**: any of the passed mandatory sequence is empty, OR any keyword argument is of the proper type but unacceptable value, OR the X and Y sequences are of different length

Description:

Calculates the Pearson`s correlation coefficient r of real numbers and the measurements with uncertainty.