Module statistics_lib.data_classes Reference

Scope

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

The concerted functional elements are classes:

- **Statistics1D**
- **Statistics2D**

Intended Use and Functionality

This module implements 'sequence on steroids' classes to perform basic statistics analysis of sequence(s) of the results of measurements with the associated uncertainties / expertimental errors. The exact values w/o an associated uncertainty (i.e. **int** or **float** type) are treated as measurements with zero uncertainty.

The classes provided in this module implement the following functionality:

- Immutability of the stored data - the values of each individual data points can be read-accessed, but not modified or removed, as well as new data points cannot be added
- Improved calculation speed - previously accessed statistical properties are cached and not re-calculated upon the consequent request
- Easy and convenient API - the statistical properties of the data sequence are accessible via read-only properties, instead of function or method call

The base design idea is that the entire data set is treated and passed around as a single object, which can calculate the own statistical properties. Thus, the API of such objects can made compatible / mostly similar to the classes implementing model continuous distributions. Therefore the functions performing statistical comparison can operate in the same manner on the measured (finite size samples) data and model distributions. Also, this approach simplifies implementation of the statistical tests functions.

The client of this module is supposed to instantiate 1D or 2D statistics class with an arbirary sequence / 2 arbitrary sequences of the same length containing either real numbers (type **int** or **float**), or 'measurements with uncertainty' - i.e. instances of classes API-compatible with **phyqus_lib.base_classes.MeasuredValue** (specifically - must have, at least, read-accessible attributes *Value* and *SE*), or a mixture of these.

In the case of 1D statistics class the input sequence is converted into 2 sequences of the same length and containing the 'mean / most probable' values of the measurements and the associated 'measurement uncertainty / error'. Both sequences are read-accessible via attributes *Values* and *Errors*, which also support the read access to the individual elements using indexing notation. For instance, the following (pseudo-) code

```
Data = Statistics1D([1, MeasuredValue(2.3, 0.3), -0.5])
print(Data.Values)
```

```
print(Data.Errors)
print(Data.Values[1], Data.Errors[1])
```

should result in the following output

```
(1, 2.3, -0.5)
(0, 0.3, 0)
2.3 0.3
```

The statistical properties of the stored data can be obtained via the attributes of the created object, e.g.

```
print(Data.Min, Data.Max, Data.Mean)
```

should output

```
-0.5 2.3 0.933333
```

The following statistical properties are available:

- Length of the data / number of elements *N*
- Arithmetic mean of the sample *Mean*
- Minimum value in the sample *Min*
- Maximum value in the sample *Max*
- Median value of the sample *Median*
- The first quartile of the sample *Q1*
- The third quartile of the sample *Q3*
- Standard error of the mean w/o contribution of the measurement uncertainties *SE* and with the contribution *FullSE*
- Population variance (w/o Bessel correction) of the data w/o contribution of the measurement uncertainties *Var* and with the contribution *FullVar*
- Population standard deviation (w/o Bessel correction) of the data w/o contribution of the measurement uncertainties *Sigma* and with the contribution *FullSigma*
- Population skewness (w/o Bessel correction) of the data w/o contribution of the measurement uncertainties *Skew*
- Population excess kurtosis (w/o Bessel correction) of the data w/o contribution of the measurement uncertainties *Kurt*

Exceptions from this design are the histogram of the distribution and the generic k-th of m-quantile, which are implemented as *methods* (*getHistogram* and *getQuantile* respectively), since they require parameters passed as arguments of the call.

The 2D statistics class must be instantiated with two sequences of the same length, representing the 'paired' observations, e.g. two properties measured on the same subjects / objects with or without the

associated 'measurement uncertainties'. The passed sequences are converted into two instances of 1D statistics class, which are read-accessible via attributes *X* and *Y*.

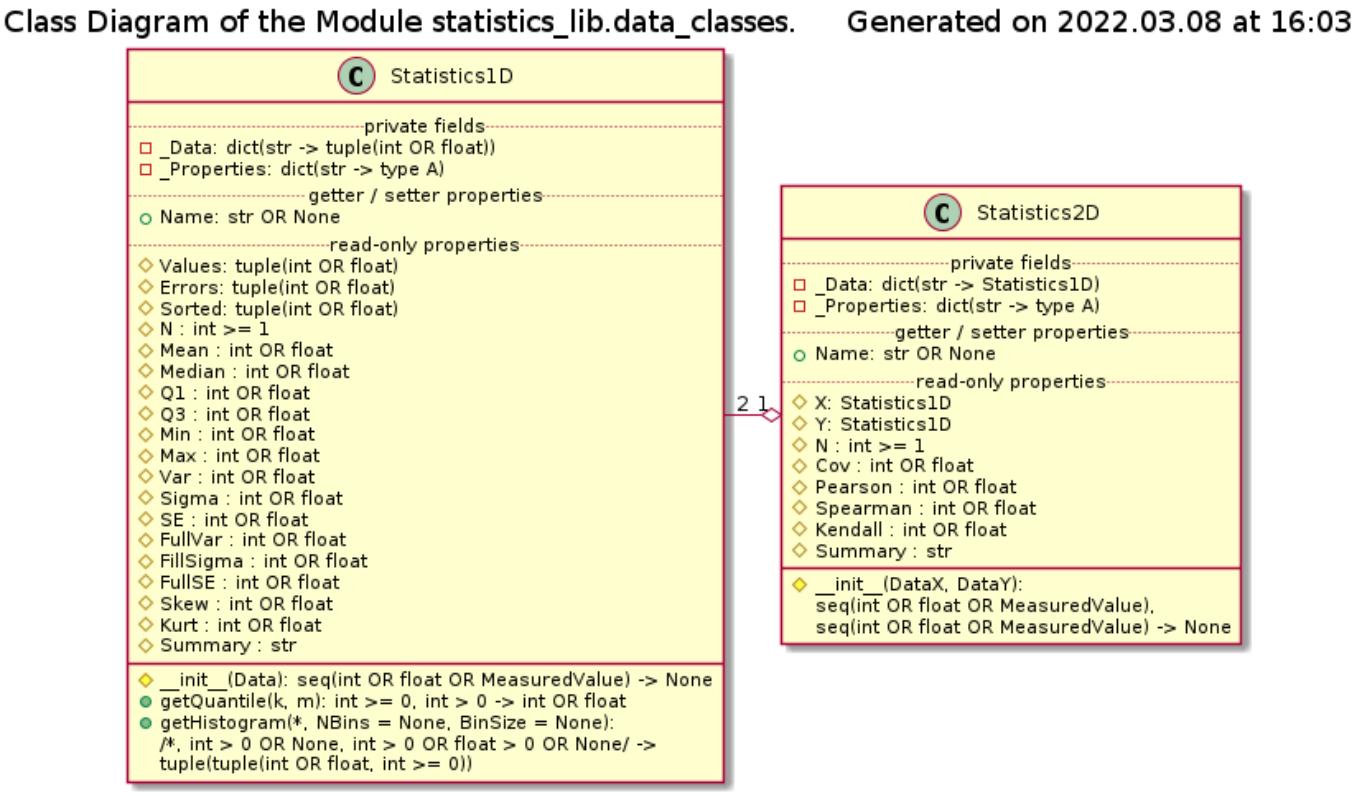
In addition, the following statistical properties can be accessed as attributes:

- Length of the data / number of pairs of elements *N*
- Covariance *Cov*
- Pearson's correlation coefficient r - *Pearson*
- Spearman rank correlation coefficient ρ - *Spearman*
- Kendall rank correlation coefficient τ-b - *Kendall*

Finally, the both classes have attribute *Summary*, which provides concise but human-readable and complete textual report on the statistical properties of the 1D / 2D data sample; and the attribute *Name*, which allows assigment and reading-out of an arbitrary string identifier of the data set.

Design and Implementation

The class diagram of the module is shown below.



The actual sample data is stored in the 'private' instance field *_Data* as tuples (1D) or instances of **Statistics1D** class (2D), and is interfaced via read-only properties *Values* and *Errors* (1D) and *X* and *Y* (2D). This design ensures the immutability of the stored data, whilst allowing the read-access to the actual data as sequences or per element.

Note, that **Statistics1D** class also has read-only property *Sorted*, which returns the elements of *Values* as a tuple and being sorted in the ascending order. In fact, this sequence is also stored in the private instance field *_Data*, but it is not created automatically upon instantiation, but upon the first explicit or implict access to the property *Sorted*. Basically, the sorting of the data is requied for the calculation of any generic quantile, includin median value, first or third quartile. If one of these properties is requested then, most

probably, the same or different quantile will be requested again during the analysis, possibly even multiple times. Therefore, it is benefical to sacrifice the memory usage (higher amount of memory) in favour of the computation speed / complexity (reduced to O(1) instead of O(N*ln(N)) using sorting each time).

Basically, the *caching of the already used data* approach is the core desing feature of the both classes. The statistical properties are not defined upon the instantiation, but are calculated upon the first access to the respective property, and then the calculated values are stored in the 'private' instance field *_Properties*. Thus the 'slow' calculations - e.g. O(N) for moment-related properties like *Mean*, *Var*, *Kurt*, *Cov* and *Pearson*, O(N*ln(N)) for *Spearman* and O(N*N) for *Kendall* - are performed only once. With the consequent access the same property the calculation speed / complexity is always O(1).

This design emphasises the re-usability of the already obtained statistical properties, resulting in the performance speed optimization on the expense of the increased memory footprint. **Note** that these classes are not intended to be used with 'big data', however several hundreds / few thousands data-points sets should not be a problem on the modern computers.

The functions defined in the modules *statistics_lib.base_functions* and *statistics_lib.ordered_functions* are used in the calculations of the statistical properties. Since the data sanity checks and convertion of the input data into sequences of real numbers is already performed, these functions are called with explicit indication to skip the data sanity checks and data convertion. Thus the use of these functons instead of direct implementation of the calculations in the methods imposes minimal overhead, with the benefit of absence of code duplication.

API Reference

Class Statistics1D

Data storage class encapsulating 1D data set and ensuring its immutability. The statistical properties are calculated 'on demand', cached and interfaced via read-only properties (attributes).

Must be instantiated with one sequence of (a mix of) real numbers or instances of classes implementing 'measurements with uncertainty' of the same length.

Properties:

- *Name*: **str**; arbitrary identifier of the data set
- *Values*: (read-only) **tuple**(**int** OR **float**); the stored 'mean / most probable' values of the data set
- *Errors*: (read-only) **tuple**(**int** >= 0 OR **float** >= 0); the stored 'errors / uncertainties' values of the measurements in the data set
- *Sorted*: (read-only) **tuple**(**int** OR **float**); the stored 'mean / most probable' values of the data set, sorted in the ascending order
- *N*: (read-only) **int** > 0; the length of the data set (number of points)
- *Mean*: (read-only) **int** OR **float**; the arithmetic mean of the stored data
- *Median*: (read-only) **int** OR **float**; the median value of the stored data
- *Q1*: (read-only) **int** OR **float**; the first quartile of the stored data
- *Q3*: (read-only) **int** OR **float**; the third quartile of the stored data
- *Min*: (read-only) **int** OR **float**; the minimum value within the stored data
- *Max*: (read-only) **int** OR **float**; the maximum value within the stored data
- *Var*: (read-only) **int** >= 0 OR **float** >= 0; the (population) variance of the data set

- *Sigma*: (read-only) **int** >= 0 OR **float** >= 0; the (population) standard deviation of the data set
- *SE*: (read-only) **int** >= 0 OR **float** >= 0; the (population) standard error of the mean of the data set
- *FullVar*: (read-only) **int** >= 0 OR **float** >= 0; the (population) full variance of the data set, including the contribution of the measurements uncertainties
- *FullSigma*: (read-only) **int** >= 0 OR **float** >= 0; the (population) full standard deviation of the data set, including the contribution of the measurements uncertainties
- *FullSE*: (read-only) **int** >= 0 OR **float** >= 0; the (population) full standard error of the mean of the data set, including the contribution of the measurements uncertainties
- *Skew*: (read-only) **int** OR **float**; the (population) skewness of the stored data
- *Kurt*: (read-only) **int** OR **float**; the (population) excess kurtosis of the stored data
- *Summary*: (read-only) **str**; the summary of the statistical properties of the data set

Instantiation:

__init__(Data)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue) -> None

Args:

Data: **seq**(**int** OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); generic sequence of the measurements data to be stored

Raises:

- **UT_TypeError**: argument is not a sequence of real numbers or measurements with uncertainty
- **UT_ValueError**: passed sequence is empty

Description:

Initialization method. Perfroms the input data sanity check, extaction of the 'means' and uncertainties of the measurements, and encapsulation of the data.

Methods:

getQuantile(k, m)

Signature:

0<= int k <= int m -> int OR float

Args:

- *k*: **int** >= 0; the quantile index, between 0 and m inclusively
- *m*: **int** > 0; the total number of quantiles

Raises:

- **UT_TypeError**: quantile index is not an integer, OR the total number of quantiles is not an integer
- **UT_ValueError**: the total number of quantilies is negative integer or zero, OR the quantile index is negative integer or integer greater than the total number of quantiles, OR the stored sequence is of

length 1

Description:

Calculates the k-th of m-quantile value of the stored data set. The computation speed is O(N*log(N)), if the sorted copy of the stored data haven't been accessed yet, otherwise - O(1), including consequtive calculation of different quantiles. The proper relations are:

- 0<= k <=m
- m > 0

getHistogram(*, NBins = None, BinSize = None)

Signature:

/*, int > 0 OR None, int > 0 OR float > 0 OR None/ -> tuple(tuple(int OR float, int >= 0))

Args:

- *NBins*: (keyword) **int** > 0 OR **None**; the desired number of bins
- *BinSize*: (keyword) **int** > 0 OR **float** > 0 OR **None**; the desired bin size, ignored is NBins is passed as not None value

Returns:

tuple(**tuple**(**int** OR **float**, **int** >= 0)): the calculated histogram as tuple of pairs (nested tuples) of the central value and the associated frequency (number of elements in the bin)

Raises:

- **UT_TypeError**: any keyword argument is of improper type
- **UT_ValueError**: any keyword argument is of the proper type but unacceptable value

Description:

Calculates the histogram of number of apperance of 'mean' values belonging to the respective bins for the data sample. Either total number of bins OR the desired bin width can be specified, where number of bins takes the precedence. When neither value is defined, the default number of bins is 20. Computation speed is always O(N).

Class Statistics2D

Data storage class encapsulating 2D data set and ensuring its immutability. The statistical properties are calculated 'on demand', cached and interfaced via read-only properties (attributes).

Must be instantiated with two sequences of (a mix of) real numbers or instances of classes implementing 'measurements with uncertainty' of the same length.

Properties:

- *Name*: **str**; arbitrary identifier of the data set
- *X*: (read-only) **Statistics1D**; the stored X data sub-set
- *Y*: (read-only) **Statistics1D**; the stored Y data sub-set

- *N*: (read-only) **int** > 0; the length of the data set (number of points)
- *Cov*: (read-only) **int** OR **float**; covariance of the data set
- *Pearson*: (read-only) **int** OR **float**; Pearson's correlation coefficient r of the data set
- *Spearman*: (read-only) **int** OR **float**; Spearman rank correlation coefficient rho of the data set
- *Kendall*: (read-only) **int** OR **float**; Kendall rank correlation coefficient tau-b of the data set
- *Summary*: (read-only) **str**; the summary of the statistical properties of the data set

Instantiation:

__init__(DataX, DataY)

Signature:

seq(int OR float OR phyqus_lib.base_classes.MeasuredValue), seq(int OR float OR phyqus_lib.base_classes.MeasuredValue) -> None

Args:

- *DataX*: **seq**(**int** OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); generic sequence of the measurements data to be stored as X sub-set
- *DataY*: **seq**(**int** OR **float** OR **phyqus_lib.base_classes.MeasuredValue**); generic sequence of the measurements data to be stored as Y sub-set

Raises:

- **UT_TypeError**: any of the arguments is not a sequence of real numbers or measurements with uncertainty
- **UT_ValueError**: any of the passed sequences is empty, or they have unequal length

Description:

Initialization method. Perfroms the input data sanity check, extaction of the 'means' and uncertainties of the measurements, and encapsulation of the data.