Test Report on the Module statistics_lib.data_classes

Conventions

Each test is defined following the same format. Each test receives a unique test identifier and a reference to the ID(s) of the requirements it covers (if applicable). The goal of the test is described to clarify what is to be tested. The test steps are described in brief but clear instructions. For each test it is defined what the expected results are for the test to pass. Finally, the test result is given, this can be only pass or fail.

The test format is as follows:

Test Identifier: TEST-[I/A/D/T]-XYZ

Requirement ID(s): REQ-uvw-xyz

Verification method: I/A/D/T

Test goal: Description of what is to be tested

Expected result: What test result is expected for the test to pass

Test steps: Step by step instructions on how to perform the test

Test result: PASS/FAIL

The test ID starts with the fixed prefix 'TEST'. The prefix is followed by a single letter, which defines the test type / verification method. The last part of the ID is a 3-digits *hexadecimal* number (0..9|A..F), with the first digit identifing the module, the second digit identifing a class / function, and the last digit - the test ordering number for this object. E.g. 'TEST-T-112'. Each test type has its own counter, thus 'TEST-T-112' and 'TEST-A-112' tests are different entities, but they refer to the same object (class or function) within the same module.

The verification method for a requirement is given by a single letter according to the table below:

Term	Definition
Inspection (I)	Control or visual verification
Analysis (A)	Verification based upon analytical evidences
Test (T)	Verification of quantitative characteristics with quantitative measurement
Demonstration (D)	Verification of operational characteristics without quantitative measurement

Tests definition (Analysis)

Test Identifier: TEST-A-300

Requirement ID(s): REQ-FUN-300

Verification method: A

Test goal: All required functionality is implemented and performs correctly.

Expected result: The classes implenting 1D and 2D statistics on encapsulated data are present and function as expected, i.e. all TEST-T-3xy tests defined in this document are passed.

Test steps: Analyze the source code of the module data_classes as well as of the unit-test module /Tests/UT003_data_classes. Execute the mentioned unit-test module. Also perform the demonstration test defined in the module /Test/DT003_data_classes.

Test result: PASS

Tests definition (Test)

Test Identifier: TEST-T-310

Requirement ID(s): REQ-AWM-300

Verification method: T

Test goal: A sub-class of **TypeError** exception is raised if 1D statistics initialization method recieves improper data type argument.

Expected result: The said exception is raised by the instantiation method if

- The passed argument is not a sequence, OR
- The pased argument is a sequence, but, at least, one element of it is neither integer, or floating point number, or an instance of a measurement with the associated uncertainty data type class

Test steps: Try to instantiate the class with the different data types, which are not a proper sequence (str is not a proper sequence) as well as a sequence containing, at least, one element, which is not a real number of measurement with uncertainty. Check that the expected error is raised each time.

Test result: PASS

Test Identifier: TEST-T-311

Requirement ID(s): REQ-AWM-301

Verification method: T

Test goal: Not possible to instantiate with an empty sequence.

Expected result: A sub-class of **ValueError** exception is raised if 1D statistics initialization method recieves an empty sequence as its argument.

Test steps: Try to instantiate the class as described above. Check that the expected exception is raised.

Test result: PASS

Test Identifier: TEST-T-312

Requirement ID(s): REQ-AWM-302

Verification method: T

Test goal: Deletion or modification of the read-only attributes is not possible.

Expected result: A sub-class of **AttributeError** exception is raised if an attempt is made to delete or assign to any read-only property of the class' instance, specifically:

- N
- Values
- Mean
- Var
- Sigma
- SE
- Skew
- Kurt
- Median
- Q1
- Q3
- Min
- Max
- FullVar
- FullSigma
- FullSE

Test steps: Instantiate a class with an arbitrary data set. Try to delete each of the listed attributes of the instance. Try to assign any value to each of the listed attributes. Check that the expected exception is raised in each of the attempts.

Test result: PASS

Test Identifier: TEST-T-313

Requirement ID(s): REQ-AWM-310

Verification method: T

Test goal: Check that it is not possible to modify a data point

Expected result: A sub-class of **TypeError** exception is raised by 1D statistics class instance if an attempt is made to modify or delete a data point.

Test steps: Instantiate the class with an arbitrary proper data set. Access the property *Values* and attempt to assign an arbitrary value to any of its elements by index (0 to N-1). Check that the expected exception is raised. Try to delete any existing element by index (0 to N-1). Check that the expected exception is raised.

Repeat the process with the property *Errors*.

Test result: PASS

Test Identifier: TEST-T-314

Requirement ID(s): REQ-AWM-311

Verification method: T

Test goal: Check that methods accept only proper data type arguments

Expected result: A sub-class of **TypeError** exception is raised by 1D statistics class instance if its method receives, at least, one argument of improper data type:

- *getQuantile*():
  - Total number of quantiles is not an integer number
  - Requested quantile index is not an integer number
- *getHistogram*():
  - Number of bins requested is not an integer number
  - Number of bins is not requested (OR None) and the requested bin size is neither integer nor floating point number

Test steps: Instantiate the class with an arbitrary proper data set. Try to call the mentioned methods with the first, the second and both arguments being of improper data type. Check that the expected exception is raised. Repeat the test with the different improper data types.

Test result: PASS

Test Identifier: TEST-T-315

Requirement ID(s): REQ-AWM-312

Verification method: T

Test goal: Check that methods do not accept an improper value of an argument

Expected result: A sub-class of **ValueError** exception is raised by 1D statistics class instance if its method receives, at least, one argument of a proper data type, but of improper value:

- *getQuantile*():
  - Total number of quantiles is an integer number, but not positive
  - Requested quantile index is an integer number, but either negative or greater than the total number of quantiles
  - Length of the stored data sequence is less than 2
- *getHistogram*():
  - Number of bins requested is an integer number, but not positive
  - Number of bins is not requested (OR None) and the requested bin size is integer or floating point number, but not positive

Test steps: Instantiate the class with an arbitrary proper data set. Try to call the mentioned methods with the first, the second and both arguments being of improper value. Check that the expected exception is raised. Repeat the test with the different improper values. Repeat the test with the stored data length of 1 element - only for *getQuantile*().

Test result: PASS

Test Identifier: TEST-T-316

Requirement ID(s): REQ-FUN-310, REQ-FUN-311

Verification method: T

Test goal: Check that the 1D statisctics class is properly instantiated, and provides read-access to the encapsulated data.

Expected result: The class can be instantiated with an arbitrary sequence of a mixture of real numbers and instances of measurement with uncertainty class. Each real number in the input is treated as a measurement with the zero uncertainty. The encapsulated stored data can be accessed as two immutable sequences: 'mean' values and uncertainties / errors. Within these sequences each individual element is read-accessible using indexing.

Test steps: Preparation

- Generate a random length lists of:
  - All integers (positive and negative are allowed) as *AllInt*
  - All floating point numbers (positive and negative are allowed) as *AllFloat*
  - A random mixture of integers and floating point numbers (positive and negative are allowed) as *Mixed*
- Generate a list of non-negative floating point numbers of the length equal the largest length of the lists above as *Errors*
- Construct the lists of instances of **phyqus_lib.base_classes.MeasuredValue** using:
  - Values from *AllInt* as the 'means' and the same index (position) values from *Errors* as the ucertainties
  - Values from *AllFloat* as the 'means' and the same index (position) values from *Errors* as the ucertainties
  - Values from *Mixed* as the 'means' and the same index (position) values from *Errors* as the ucertainties
- Construct a mixed list of real numbers and **phyqus_lib.base_classes.MeasuredValue** instances from *Mixed* list, where some elements from the original list are randomly converted into 'measurement with uncertainty' using the respective (same index) element from *Errors* as the uncertainty

Instantiate the class with each of the prepared lists in turn. Check that no exception is raised. Check that both the *Values* and *Errors* properties return sequences of the same length, equal to the length of the initialization sequence argument, and by values they are as expected for the 'means' and 'errors' of the input data.

Note - the immutability of the *Values* and *Errors* properties is already tested in TEST-T-312.

Repeat the test with each prepared list converted into a tuple before passing into the instantiation methods.

Test result: PASS

Test Identifier: TEST-T-317

Requirement ID(s): REQ-FUN-312, REQ-FUN-313

Verification method: T

Test goal: Check that the statistical properties of the encapsulated data are properly calculated.

Expected result: The 1D statistics class should provide the following properites:

- *N* - int > 0, length of the data set (number of data points)
- *Mean* - int OR float, aritmetic mean
- *Var* - int OR float, variance
- *Sigma* - int OR float, standard deviation
- *SE* - int OR float, standard error of the mean
- *Skew* - int OR float, skewness of the distribution
- *Kurt* - int OR float, excess kurtosis of distribution
- *Median* - int OR float, median of the distibution
- *Q1* - int OR float, the first quartile of the distribution
- *Q3* - int OR float, the third quartile of the distribution
- *Min* - int OR float, the minimum value within the distribution
- *Max* - int OR float, the maximum value within the distribution

The Bessel correction should not be applied, the data sample should be treated as the whole population.

In addition, the following methods should be implemented:

- *getQuartile*(k, m) - 0<= int k <= int m -> int OR float, generic k-th of m-quantile (m > 0)
- *getHistogram*(*, NBins = None, BinSize = None) - /int > 0 OR None, int > 0 OR float > 0 OR None/ -> tuple(tuple(int OR float, int >= 0))

The 1D statistics class should provide also the following properites:

- *FullVar* - int OR float, variance, including the contribution of the measurements uncertainties
- *FullSigma* - int OR float, standard deviation, including the contribution of the measurements uncertainties
- *FullSE* - int OR float, standard error of the mean, including the contribution of the measurements uncertainties

The Bessel correction should not be applied, the data sample should be treated as the whole population.

Calculations values returned by this properties and methods MUST be equal to those returned by the corresponding functions in the modules **base_functions** and **ordered_functions** (with type casting, when required).

Test steps: Prepare the test sequences as in TEST-T-316. Instantiate the class being tested with each of the sequences in turn and check that:

- Property *N* returns the same value as the built-in function *len*() applied to the argument of the instantiation method
- Property *Mean* returns the same value as the function *base_functions.GetMean*() with the same argument as the instantition method
- Property *Var* returns the same value as the function *base_functions.GetVarianceP*() with the same argument as the instantition method

- Property *Sigma* returns the same value as the function *base_functions.GetStdevP*() with the same argument as the instantition method
- Property *SE* returns the same value as the function *base_functions.GetSE*() with the same argument as the instantition method
- Property *Skew* returns the same value as the function *base_functions.GetSkewnessP*() with the same argument as the instantition method
- Property *Kurt* returns the same value as the function *base_functions.GetKurtosisP*() with the same argument as the instantition method
- Property *Median* returns the same value as the function *ordered_functions.GetMedian*() with the same argument as the instantition method
- Property *Q1* returns the same value as the function *ordered_functions.GetFirstQuartile*() with the same argument as the instantition method
- Property *Q3* returns the same value as the function *ordered_functions.GetThirdQuartile*() with the same argument as the instantition method
- Property *Min* returns the same value as the function *ordered_functions.GetMin*() with the same argument as the instantition method
- Property *Max* returns the same value as the function *ordered_functions.GetMax*() with the same argument as the instantition method
- Property *FullVar* returns the same value as the sum of results of function *base_functions.GetVarianceP*() adn *base_functions.GetMeanSqrSE*() with the same argument as the instantition method
- Property *FullSigma* returns the same value as the square root of the sum of results of function *base_functions.GetVarianceP*() adn *base_functions.GetMeanSqrSE*() with the same argument as the instantition method
- Property *FullSE* returns the same value as the function *base_functions.GetFullSE*() with the same argument as the instantition method

Check that the method *getQuartile*() returns the same value as the function *ordered_functions.GetQuantile*() with the same data, total number of quantiles and the requested quantile index passed.

Check that the method *getHistogram*() returns a tuple of 2-element tuples with the same values as the key : value pairs in the dictionary returned by the function *ordered_functions.GetHistogram*() with the same data and bin size / number of bin requested. Check all three options - requesting specific number of bins, requesting specific bin size and using the default behaviour.

Test result: PASS

Test Identifier: TEST-T-318

Requirement ID(s): REQ-FUN-314

Verification method: T

Test goal: Check assignment and read-out of the data set name.

Expected result: An arbitrary string name can be assigned to 2D data set, and it can be read-out. Until set, the name is set to **None**.

Test steps: Instantiate 2D statistics class with arbitrary but proper data. Check that it has attribute *Name*, and its value is **None**. Assign (in turn) arbitrary string, integer, floating point number, boolean value and **None** to this attribute, and check that the value of the attribute is the passed value converted into string.

Test result: PASS

Test Identifier: TEST-T-320

Requirement ID(s): REQ-AWM-300

Verification method: T

Test goal: A sub-class of **TypeError** exception is raised if 2D statistics initialization method recieves, at least, one improper data type argument.

Expected result: The said exception is raised by the instantiation method if

- At least, one of the passed arguments is not a sequence, OR
- At least, one of the passed arguments is a sequence, but, at least, one element of it is neither integer, or floating point number, or an instance of a measurement with the associated uncertainty data type class

Test steps: Try to instantiate the class with the different data types as the first argument, which are not a proper sequence (str is not a proper sequence) as well as a sequence containing, at least, one element, which is not a real number of measurement with uncertainty. Pass a proper simple sequence of real numbers as the second argument. Check that the expected error is raised each time.

Repeat the test with the arguments being swapped. Check that the expected error is raised each time.

Repeat the test with the both arguments being of improper type. Check that the expected error is raised each time.

Test result: PASS

Test Identifier: TEST-T-321

Requirement ID(s): REQ-AWM-301

Verification method: T

Test goal: A sub-class of **ValueError** exception is raised if the lengths of the arguments of 2D statistics class initialization are not equal or zero.

Expected result: The said exception is raised if one or both arguments of the initialization methods is an empty sequence, or the both arguments are proper sequences but of unequal length.

Test steps: Try to instantiate the class with the first argument being an empty sequence and the second - a proper, not empty sequence of real numbers. Check that the expected exception is raised.

Try to instantiate the class with the same arguments being swapped. Check that the expected exception is raised.

Try to instantiate the class with the both arguments being empty sequences. Check that the expected exception is raised.

Try to instantiate the classe with the both arguments being proper non-empty sequences of real numbers, but of unequal length. Check that the expected exception si raised.

Test result: PASS

Test Identifier: TEST-T-322

Requirement ID(s): REQ-AWM-302

Verification method: T

Test goal: Deletion or modification of the read-only attributes is not possible.

Expected result: A sub-class of **AttributeError** exception is raised if an attempt is made to delete or assign to any read-only property of the class' instance, specifically:

- N
- Cov
- Pearson
- Spearman
- Kendall
- X
- Y

Test steps: Instantiate a class with two arbitrary data sets of the same length. Try to delete each of the listed attributes of the instance. Try to assign any value to each of the listed attributes. Check that the expected exception is raised in each of the attempts.

Test result: PASS

Test Identifier: TEST-T-323

Requirement ID(s): REQ-FUN-320, REQ-FUN-321

Verification method: T

Test goal: Check that the 2D statistics class is properly instantiated, and provides read-access to the encapsulated data.

Expected result: The class can be instantiated with two arbitrary sequences of a mixture of real numbers and instances of measurement with uncertainty class. Each real number in the input is treated as a measurement with the zero uncertainty. The stored encapsulated data is accessible via two read-only properties, which return instances of the 1D statistics class.

Test steps: Preparation

- Generate random but equal length pairs of lists of:
    - All integers (positive and negative are allowed) as *AllIntX* and *AllIntY*

    - All floating point numbers (positive and negative are allowed) as *AllFloatX* and *AllFloatY*
    - A random mixture of integers and floating point numbers (positive and negative are allowed) as *MixedX* and *MixedY*
- Generate tow list of non-negative floating point numbers of the length equal the largest length of the lists above as *ErrorsX* and *ErrorsY*
- Construct the lists of instances of **phyqus_lib.base_classes.MeasuredValue** using:
    - Values from *AllIntX* / *AllIntY* as the 'means' and the same index (position) values from *ErrorsX* / *ErrorsY* as the ucertainties
    - Values from *AllFloatX* / *AllFloatY* as the 'means' and the same index (position) values from *Errors* / *ErrorsY* as the ucertainties
    - Values from *MixedX* / *MixedY* as the 'means' and the same index (position) values from *Errors* / *ErrorsY* as the ucertainties
- Construct two mixed lists of real numbers and **phyqus_lib.base_classes.MeasuredValue** instances from *MixedX* / *MixedY* lists, where some elements from the original list are randomly converted into 'measurement with uncertainty' using the respective (same index) element from *ErrorsX* / *ErrorsY* as the uncertainty

Instantiate the class with each of the prepared lists pairs in turn. Check that no exception is raised. Check that both the *X* and *Y* properties return instances of the 1D statistics class,; whereas the properties *Values* and *Errors* of each return the proper 'means' and 'errors' sequences for the X and Y data.

Note - the immutability of the *X* and *Y* properties is already tested in TEST-T-322.

Repeat the test with each prepared list converted into a tuple before passing into the instantiation methods.

Test result: PASS

Test Identifier: TEST-T-324

Requirement ID(s): REQ-FUN-322

Verification method: T

Test goal: Proper calculation of the 2D statistical properties

Expected result: The following properties are present, and return the values are expected for the encapsulated data:

- *N* - int > 0, length of the 2D data set (number of paired X-Y data points)
- *Cov* - int OR float, covariance
- *Pearson* - int OR float, Pearson's correlation coefficient
- *Spearman* - int OR float, Spearman rank correlation coefficient
- *Kendall* - int OR float, Kendall τ-*b* rank correlation coefficient

Test steps: Prepare the test sequences as in TEST-T-323. Instantiate the class being tested with each pair of the sequences in turn and check that:

- Property *N* returns the same value as the built-in function *len*() applied to one of the arguments of the instantiation method

- Property *Cov* returns the same value as the function *base_functions.GetCovariance*() with the same arguments as the instantition method
- Property *Pearson* returns the same value as the function *base_functions.GetPerasonR*() with the same arguments as the instantition method
- Property *Spearman* returns the same value as the function *ordered_functions.GetSpearman*() with the same arguments as the instantition method
- Property *Kendall* returns the same value as the function *ordered_functions.GetKendall*() with the same arguments as the instantition method

Test result: PASS

Test Identifier: TEST-T-325

Requirement ID(s): REQ-FUN-323

Verification method: T

Test goal: Check assignment and read-out of the data set name.

Expected result: An arbitrary string name can be assigned to 2D data set, and it can be read-out. Until set, the name is set to **None**.

Test steps: Instantiate 2D statistics class with arbitrary but proper data. Check that it has attribute *Name*, and its value is **None**. Assign (in turn) arbitrary string, integer, floating point number, boolean value and **None** to this attribute, and check that the value of the attribute is the passed value converted into string.

Test result: PASS

Tests definition (Demonstration)

Test Identifier: TEST-D-300

Requirement ID(s): REQ-FUN-315, REQ-FUN-324

Verification method: D

Test goal: Generation of the statistical properties report

Expected result: The textual, human readable report is generated in the expected form, and contains the proper values.

Test steps: Instantiate 1D statistics class with arbitrary data. Print-out the value of its property *Summary*. Check that it contains all lines defined in the REQ-FUN-315, but it has no *Name* line. Print-out the values of the corresponding properties directly and visually compare with the report. Set an arbitrary string as *Name*. Print the report again. Check that the only change is that the respective line is added.

Instantiate 2D statistics class with arbitrary data. Print-out the value of its property *Summary*. Check that it contains all lines defined in the REQ-FUN-324, but it has no *Name* line, and the *X* and *Y* sub-sets reports are present as well. Print-out the values of the corresponding properties (2D) directly and visually compare with the report. Print-out the values of the corresponding properties of *X* and *Y* sub-sets directly and

visually compare with the report. Set some name in *X* sub-set. Check that it appears in the report. Set *Y* sub-set's name and the 2D set name. Check that they appear in the report.

Test result: PASS

Traceability

For traceability the relation between tests and requirements is summarized in the table below:

Requirement ID	**Covered in test(s)**	**Verified [YES/NO])**
REQ-FUN-300	TEST-A-300	YES
REQ-FUN-310	TEST-T-316	YES
REQ-FUN-311	TEST-T-316	YES
REQ-FUN-312	TEST-T-317	YES
REQ-FUN-313	TEST-T-317	YES
REQ-FUN-314	TEST-T-318	YES
REQ-FUN-315	TEST-D-300	YES
REQ-FUN-320	TEST-T-323	YES
REQ-FUN-321	TEST-T-323	YES
REQ-FUN-322	TEST-T-324	YES
REQ-FUN-323	TEST-T-325	YES
REQ-FUN-324	TEST-D-300	YES
REQ-AWM-300	TEST-T-310, TEST-T-320	YES
REQ-AWM-301	TEST-T-311, TEST-T-321	YES
REQ-AWM-302	TEST-T-312, TEST-T-322	YES
REQ-AWM-310	TEST-T-313	YES
REQ-AWM-311	TEST-T-314	YES
REQ-AWM-312	TEST-T-315	YES

Software ready for production [YES/NO]	**Rationale**
YES	All tests are passed