Test Report on the Module statistics_lib.ordered_functions

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-200

Requirement ID(s): REQ-FUN-200

Verification method: A

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

Expected result: The following functions are present:

- Minimum value in the sample
- Maximum value in the sample
- Mode(s) of the sample distribution
- Median value of the sample distribution
- First quartile of the sample distribution
- Third quartile of the sample distribution
- An arbitrary N/M quantile (N < M) of the sample distribution
- The histogram of the sample distribution with adjustable width of the bins
- Spearman ρ rank correlation coefficient of 2D data set
- Kendall τ rank correlation coefficient of 2D data set

All these function pass TEST-T-200, TEST-T-201 and TEST-T-202, as well as individual tests TEST-T-210 to TEST-T-2A0 concerning their returned values.

Test steps: Analyze the source code of the module ordered_functions as well as of the unit-test module /Tests/UT002_ordered_functions. Execute the mentioned unit-test module.

Test result: PASS

Tests definition (Test)

Test Identifier: TEST-T-200

Requirement ID(s): REQ-FUN-201

Verification method: T

Test goal: Check the acceptable input data types and the type of the returned value.

Expected result: All 1D statistics functions accept any non-empty sequence of either integer numbers, or floating point numbers, or instances of a class compatible with **phyqus_lib.base_classes.MeasuredValue**, i.e. having data attributes *Value* and *SE*, or any mixture of these three data types. All 2D statistics functions (correlation) accept two such sequences of equal length.

In all functions any instance of 'measurement with uncertainty' is treated as a real number equal to its *Value* attribute, i.e. (x_i, z_i) -> x_i. In the function calculating the mean squared uncertainty each real number is considered to be an instance of 'measurement with uncertainty', where uncertainty is zero, i.e. x_i -> (x_i, 0). The total error calculation function uses the both described data type casting mechanisms to find the respective parts of the total error / uncertainty.

The functions return the expected data types:

- a real number - integer of floating point - for the functions returning a single scalar value
- list of real numbers - for the function calculating mode(s) of the distribution

- a list of 2-element tuples of a real numbers or a dictionary (real -> int >= 0) for the function calculating a histogram

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

Pass each of the created lists into the function being tested. Check that no exception is raised. Check that the returned value data type.

Repeat the test coverting the same lists into tuples before passing into the function being tested.

In case of of the 2D statistic functions 2 sets of the lists of these data types should be generated. Before passing into a function, the longest of the two sequence arguments must be truncated to match the length of the shortest one.

This test is applied to all functions!

Test result: PASS

Test Identifier: TEST-T-201

Requirement ID(s): REQ-AWM-200

Verification method: T

Test goal: A sub-class of **TypeError** exception is raised in response to the improper type of the input data.

Expected result: Such an exception is raised in the following cases:

- A 1D data set (or one of the sub-sets of the 2D data set) is not a flat sequence of real numbers of 'real life measurements' with the associated uncertainties, which means:
- The respective argument is not a sequence, OR

- The respective 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
- The required quantile index or total number of quantiles argument is not an integer number
- The requested number of bins (in histogram) is not an integer number
- The requested bin size (in histogram) is not a real number

Test steps: Try to call the funcion being tested with an appropriate data type argument. Check that the expected exception is raised.

Test result: PASS

Test Identifier: TEST-T-202

Requirement ID(s): REQ-AWM-201

Verification method: T

Test goal:A sub-class of **ValueError** exception is raised in response to the improper values of the proper type of the input data.

Expected result: Such an exception is raised in the following cases:

- At least, one of the sequence data set arguments is an empty sequence
- Inequal length of the sequence arguments of a 2D statistics function (in general)
- The sequence is 1 element long in the case of quartile and quantile functions
- The total number of quantiles is an integer, but not positive - quantile function
- The required quantile index is negative or larger than the total number of quantiles - quantile function
- The requested number of bins (in histogram) is an integer but not positive
- The requested bin size (in histogram) is a real number but not positiv

Test steps: Try to call the funcion being tested with an appropriate argument (see above). Check that the expected exception is raised.

Test result: PASS

Test Identifier: TEST-T-210

Requirement ID(s): REQ-FUN-210

Verification method: T

Test goal: The performance of the function *GetMin*().

Expected result: With a random sequence of the mix of integer, floating point numbers and instances of the measurements with uncertainty class passed into the function, it returns the minimum value of all 'mean' values, i.e. the same value as the built-in function *min*() would return on the same sequence with all instances of the measurements with uncertainty class converted into their 'mean' values (integer or floating point). The returned value is an integer or floating point number.

Test steps: Prepare the test sequences same way as described in the TEST-T-200. Pass each of the test sequences into the function being tested. Check that the returned values is either an integer or floating point data type, and it is equal to the result returned by the built-in function *min*(), into wich the base real numbers only sequence is passed.

Test result: PASS

Test Identifier: TEST-T-220

Requirement ID(s): REQ-FUN-220

Verification method: T

Test goal: The performance of the function *GetMax*().

Expected result: With a random sequence of the mix of integer, floating point numbers and instances of the measurements with uncertainty class passed into the function, it returns the maximum value of all 'mean' values, i.e. the same value as the built-in function *max*() would return on the same sequence with all instances of the measurements with uncertainty class converted into their 'mean' values (integer or floating point). The returned value is an integer or floating point number.

Test steps: Prepare the test sequences same way as described in the TEST-T-200. Pass each of the test sequences into the function being tested. Check that the returned values is either an integer or floating point data type, and it is equal to the result returned by the built-in function *max*(), into wich the base real numbers only sequence is passed.

Test result: PASS

Test Identifier: TEST-T-230

Requirement ID(s): REQ-FUN-230

Verification method: T

Test goal: The performance of the function *GetMedian*().

Expected result: With a random sequence of the mix of integer, floating point numbers and instances of the measurements with uncertainty class passed into the function, it returns the median value of all 'mean' values, i.e. the same value as the Standard Library function *statistics.median*() would return on the same sequence with all instances of the measurements with uncertainty class converted into their 'mean' values (integer or floating point). The returned value is an integer or floating point number.

Test steps: Prepare the test sequences same way as described in the TEST-T-200. Pass each of the test sequences into the function being tested. Check that the returned values is either an integer or floating point data type, and it is equal to the result returned by the Standard Library function *statistics.median*(), into wich the base real numbers only sequence is passed.

Test result: PASS

Test Identifier: TEST-T-240

Requirement ID(s): REQ-FUN-240

Verification method: T

Test goal: The performance of the function *GetFirstQuartile*().

Expected result: With a random sequence of the mix of integer, floating point numbers and instances of the measurements with uncertainty class passed into the function (length >= 2), it returns the first quartile value of all 'mean' values using the linear interpolation between the values of the adjacent elements in the sorted sample. Using Python v3.8 or later it should return the same value as the first element of the list returned by the Standard Library function *statistics.quantiles*() with n=4 (default).

Test steps: Prepare the test sequences same way as described in the TEST-T-200. Pass each of the test sequences into the function being tested. Check that the returned values is either an integer or floating point data type. Sort the base real numbers only sequence. Check that the previously calculated value is greater than or equal to the sorted sequence element index (N-1) // 4 and less than or equal to the element index ((N-1) // 4) + 1, where N is the length of the data sequence. If the Python v3.8 or later interprer is used, check the calculated value against the first element of the list returned by *statistics.quantiles*() function with n=4 (default) and method = 'inclusive' keyword arguments, into which the base but nor sorted real numbers only sequence is passed.

Test result: PASS

Test Identifier: TEST-T-250

Requirement ID(s): REQ-FUN-250

Verification method: T

Test goal: The performance of the function *GetThirdQuartile*().

Expected result: With a random sequence of the mix of integer, floating point numbers and instances of the measurements with uncertainty class passed into the function (length >= 2), it returns the third quartile value of all 'mean' values using the linear interpolation between the values of the adjacent elements in the sorted sample. Using Python v3.8 or later it should return the same value as the last element of the list returned by the Standard Library function *statistics.quantiles*() with n=4 (default).

Test steps: Prepare the test sequences same way as described in the TEST-T-200. Pass each of the test sequences into the function being tested. Check that the returned values is either an integer or floating point data type. Sort the base real numbers only sequence. Check that the previously calculated value is greater than or equal to the sorted sequence element index ((N-1) * 3) // 4 and less than or equal to the element index (((N-1) * 3) // 4) + 1, where N is the length of the data sequence. If the Python v3.8 or later interprer is used, check the calculated value against the last element of the list returned by *statistics.quantiles*() function with n=4 (default) and method = 'inclusive' keyword arguments, into which the base but nor sorted real numbers only sequence is passed.

Test result: PASS

Test Identifier: TEST-T-260

Requirement ID(s): REQ-FUN-260

Verification method: T

Test goal: The performance of the function *GetQuantile*().

Expected result: With a random sequence of the mix of integer, floating point numbers and instances of the measurements with uncertainty class passed into the function (length >= 2), it returns the k-th of m-quartile value of all 'mean' values using the linear interpolation between the values of the adjacent elements in the sorted sample. Using Python v3.8 or later it should return the same value as the (k-1)th element (indexing from 0) of the list returned by the Standard Library function *statistics.quantiles*() with n=m, providing 0 < k < m. The 0-th quantile is the first element of the sorted in ascending order sequence of the 'mean' values, whereas the m-th quantile is the last element of that sequence.

Test steps: Prepare the test sequences same way as described in the TEST-T-200. For each of the test sequences do the following:

- Sort the base real numbers only sequence.
- Select m from [4, 10, 25, 33, 100]
- For each 0 < k < m:
    - Pass the current test sequence (not sorted) into the function being tested with the selected k and m arguments
    - Check that the returned values is either an integer or floating point data type.
    - Check that the previously calculated value is greater than or equal to the sorted sequence element index ((N-1) * k) // m and less than or equal to the element index (((N-1) * k) // m) + 1, where N is the length of the data sequence.
    - If the Python v3.8 or later interprer is used, check the calculated value against the (k-1)-th element (index starts at 0) of the list returned by *statistics.quantiles*() function with n=m (default) and method = 'inclusive' keyword arguments, into which the base but nor sorted real numbers only sequence is passed.
- Check that the result of the function call with k = 0 is the first element of the sorted base sequence
- Check that the result of the function call witk k = m is the last element of the sorted base sequence
- Repeat the process with other m values

Test result: PASS

Test Identifier: TEST-T-270

Requirement ID(s): REQ-FUN-270

Verification method: T

Test goal: The performance of the function *GetHistogram*().

Expected result: With a random sequence of the mix of integer, floating point numbers and instances of the measurements with uncertainty class passed into the function (length >= 2), it returns a histogram of all 'mean' values, i.e. how many data points fall into each of the equidistant bins. The following rules are applied:

- The minimum value observed in the data sample belong to the left-most (min value) bin

- The maximum value observed in the data sample belong to the right-most (max value) bin
- The value of a bin is the mid-point of the range that bin covers
- The bins are equidistant, i.e. each cover the same range, equal to the difference between the values of the adjacent bins
- All bins are present, even if they are empty
- The min/max values of the data sample, number of bins ***N*** and the bin size ***S*** are related as $N - 2 \leq \frac{max(X) - min(X)}{S} \leq N$
- The number of bins or the bin size can be requested specifically via keyword-only arguments:
    - If the number of bins ***N*** is passed, the minimum and maximum values in the sample are centered to the left- and right-most bins, and the step is defined as $S = \frac{max(X) - min(X)}{N - 1}$
    - If the bin size ***S*** is passed, the arithmetic mean of the sample is centered to its respective bin, and the total number of bins is defined as $N = \lceil \frac{max(X) - \langle X \rangle}{S} - \frac{1}{2} \rceil + \lceil \frac{\langle X \rangle - min(X)}{S} - \frac{1}{2} \rceil + 1$, where the first and the second parts of the sum are the number of bins to the right and to the left from the 'central' one
    - If neither ***N*** nor ***S*** are defined by the user, the ***N*** = 20 number of bins is used
    - If both ***N*** and ***S*** are defined by the user, the bin size argument is ignored, and value defined by the number of bins is used instead

Test steps: Prepare the test sequence (mixed types) same way as described in the TEST-T-200.

- Call the test function without keyword arguments (default 20 bins).
- Check that the returned result is a dictionary.
- Check that is has exactly the expected number N of key : value pairs
- Check that all keys are real numbers
- Check that all values are non-negative integers
- Check that keys are sorted in the ascending order, the first key is the almost equal to the minimum value in the data sample, the last key - to the maximum value in the sample
- Define the bin size as (max - min) / (N - 1), check that the difference between each pair of the adjacent keys almost equals this size
- Check that the sum of all values equals the length of the sample
- Check that for each key K and the calculated bin size S the amound of the data points between >= K - S/2 and < K + S/2 equals the value of that key
- Repeat the test with the explicitely passed desired number of bins with and without indication of the desired bin size
- Repeat the test only indicating the desired bin size; apply the following modifications to the test:
    - The bin size is as requested
    - Number of bins satisfies the double inequality above
- The edge case is when the data sample contain only the same value elements, in which case the histogram should contain only a single key : value pair

N.B. The exact number of the items per bin calculated in the function and in the unit test may differ due to the rounding errors in the case of the values close to the boundaries between the bins, especially if different algorithms are used in the unit test and in the actual function. The result of the test should be accepted even if the unit-testing sometimes fails, provided that:

- Total number of the elements in all bins is preserved
- The bin's number of elements comparison is the only reported failure

- The reported difference is small (~ 1 - 2 items)
- The failure doesn't happen often (< 10% of the consequitive runs)

Test result: PASS

Test Identifier: TEST-T-280

Requirement ID(s): REQ-FUN-280

Verification method: T

Test goal: The performance of the function *GetModes*().

Expected result: With a random sequence of the mix of integer, floating point numbers and instances of the measurements with uncertainty class passed into the function, it returns a list of all values, which occur most frequently occuring values, i.e. the mode(s) of the sample distribution. Using Python v3.8 or later it should return a list containing the same elements but, possibly, in a different order as the list returned by the Standard Library function *statistics.multimode*().

Test steps: Prepare the test sequence (mixed types). Manually define the expected mode(s) values. Pass the formed sequence into the function being tested, and compare the returned list with the expected modes list. Repeat the process with the samples containg 1, 2, 3 modes and the edge case of all values being unique, i.e. each value is a mode.

In running Python 3.8+ interpreter, use the same randomly generated sequences as in TEST-T-200, and compare the result with the return values of *statistics.multimode*() function, into wich the base real number sequences are passed.

Test result: PASS

Test Identifier: TEST-T-290

Requirement ID(s): REQ-FUN-290

Verification method: T

Test goal: The performance of the function *GetSpearman*().

Expected result: With a random sequence of the mix of integer, floating point numbers and instances of the measurements with uncertainty class passed into the function, it returns the Spearman rank correlation coefficient of the 'means', which is the Pearson's correlation of the *fractional ranks* of the sorted X and Y sequences.

Test steps: Prepare the test sequences (mixed types) of the same length. Pass them into the function being tested and compare the returned result with the expected value calculated manually. Do the same for the sequences containg also:

- Few X-ties only
- Few Y-ties only
- Few X-only ties and Y-only ties

- Few X-Y ties (identical, duplicating data points)
- Few X-only, Y-only and X-Y ties

For the edge cases the following sequences should be used:

- $y_i = \sqrt{(x_i)}$ - expected correlation coefficient ~ 1
- $y_i = \frac{1}{x_i}$ - expected correlation coefficient ~ -1
- $y_i = (-1)^i \times x_i$ - expected correlation coefficient ~ 0, expected in the range -0.25 to + 0.25

Test result: PASS

Test Identifier: TEST-T-2A0

Requirement ID(s): REQ-FUN-2A0

Verification method: T

Test goal: The performance of the function *GetKendall*().

Expected result: With a random sequence of the mix of integer, floating point numbers and instances of the measurements with uncertainty class passed into the function, it returns the Kendall rank correlation coefficient of the 'means' using the τ-b variant of the algorithm, i.e. accounting for the ties.

Test steps: Prepare the test sequences (mixed types) of the same length, which does not include any ties. Pass them into the function being tested and compare the returned result with the expected value calculated manually. Do the same for the sequences containg also:

- Few X-ties only
- Few Y-ties only
- Few X-only ties and Y-only ties
- Few X-Y ties (identical, duplicating data points)
- Few X-only, Y-only and X-Y ties

For the edge cases the same sequences as in TEST-T-290 should be used:

- $y_i = \sqrt{(x_i)}$ - expected correlation coefficient ~ 1
- $y_i = \frac{1}{x_i}$ - expected correlation coefficient ~ -1
- $y_i = (-1)^i \times x_i$ - expected correlation coefficient ~ 0, expected in the range -0.25 to + 0.25

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-200	TEST-A-200	YES
REQ-FUN-201	TEST-T-200	YES
REQ-FUN-210	TEST-T-210	YES

Requirement ID	Covered in test(s)	Verified [YES/NO])
REQ-FUN-220	TEST-T-220	YES
REQ-FUN-230	TEST-T-230	YES
REQ-FUN-240	TEST-T-240	YES
REQ-FUN-250	TEST-T-250	YES
REQ-FUN-260	TEST-T-260	YES
REQ-FUN-270	TEST-T-270	YES
REQ-FUN-280	TEST-T-280	YES
REQ-FUN-290	TEST-T-290	YES
REQ-FUN-2A0	TEST-T-2A0	YES
REQ-AWM-200	TEST-T-201	YES
REQ-AWM-201	TEST-T-202	YES

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