IbaAnalyzer 5.18.0 new functionality description

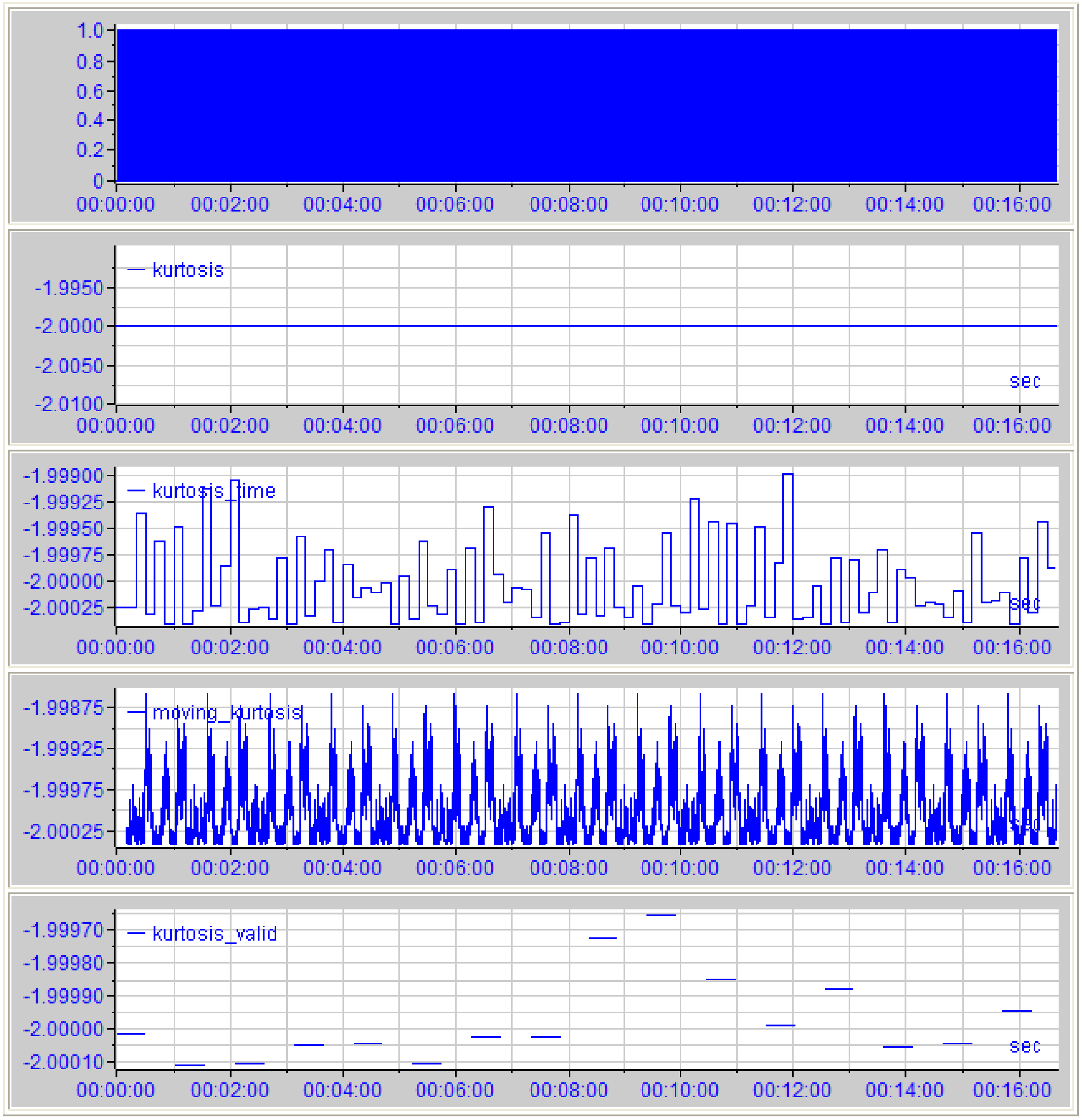
ibaAnalyzer new statistical functions

Kurtosis

For the definition of the kurtosis statistical function see
http://en.wikipedia.org/wiki/Kurtosis

Similar as for the ibaAnalyzer 'StdDev' function, five variants of the function are implemented:

- **Kurtosis ('Expression')**: This function returns the kurtosis of the dataset generated by the values of 'Expression'.
- **KurtosisInTime('Expression', 'Interval')**: This function splits the values of 'expression' along the X-axis in non-overlapping adjacent intervals and returns the kurtosis in each interval. The function takes the following parameters:
  - **Expression**: The signal whose values generate the dataset to take the kurtosis over.
  - **Interval**: The size of the intervals the kurtosis will be taken over, specified in X-axis units.
- **MKurtosis('Expression', 'Interval')**: Similar as KurtosisInTime but this time the intervals overlap for all but one measuring point. The interval the kurtosis is taken over can be envisioned to 'move' over the X-axis. The function takes the following parameters:
  - **Expression**: The signal whose values generate the dataset to take the kurtosis over.
  - **Interval**: The size of the intervals the kurtosis will be taken over, specified in X-axis units.
- **KurtosisValid('Expression', 'Valid')**: This function returns the kurtosis of 'Expression' for each signal where the control signal 'Valid' is true. The resulting signal is invalid where the control signal is false. The function takes the following parameters:
  - **Expression**: The signal whose values generate the dataset to take the kurtosis over.
  - **Valid**: The control signal to determine the intervals where the kurtosis needs to be taken.
- **VectorKurtosis('Vector')** : Returns a signal with for each sample the kurtosis value of all elements of *'Vector'* at that sample position. *'Vector'* is a multidimensional expression. Since the kurtosis can only be calculated for a data set of at least four samples, *'Vector'* should contain at least four elements.



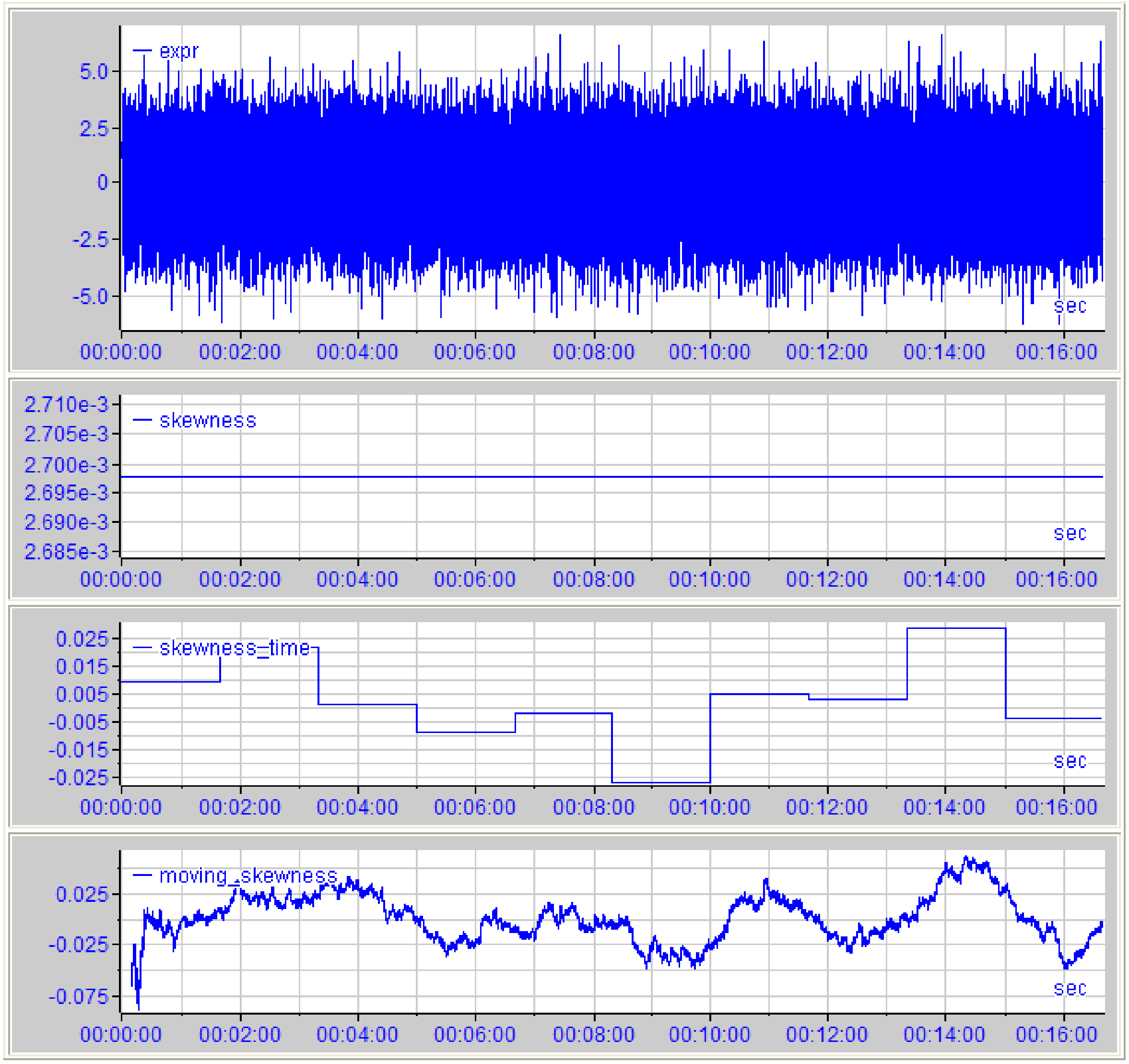
	SignalName	Expression	Comment 1
1	expr	Mod(Rand(1000000,0.001) , 2)	Simulated unbiased coin toss
2	kurtosis	Kurtosis([expr])	global kurtosis
3	kurtosis_time	KurtosisInTime([expr],10)	kurtosis at fixed intervals
4	moving_kurtosis	XMarkValid(MKurtosis([expr],10) , XValues([expr]) > 10)	kurtosis over gliding interval
5	kurtosis_valid	KurtosisValid([expr],Sin(Time(1000000,0.01)/10)>0)	kurtosis where a control signal is true

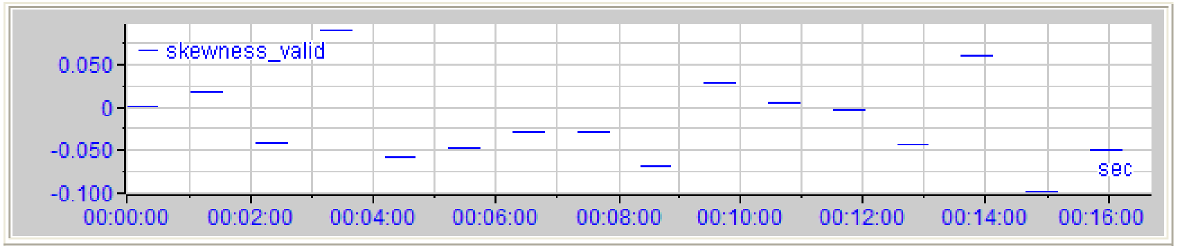
Skewness

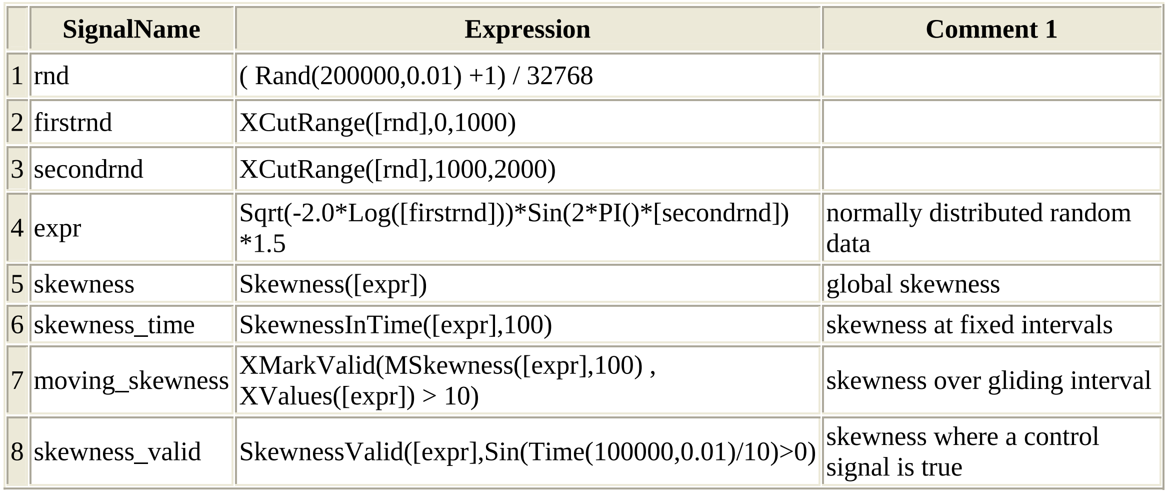
For the definition of the skewness statistical function see http://en.wikipedia.org/wiki/Skewness

The same variants as for the ‘Kurtosis’ and ‘StdDev’ functions are implemented:

- **Skewness (‘Expression’)**
- **SkewnessInTime(‘Expression’, ‘Interval’)**
- **MSkewness(‘Expression’, ‘Interval’)**
- **SkewnessValid(‘Expression’, ‘Valid’)**
- **VectorSkewness(‘Vector’)**







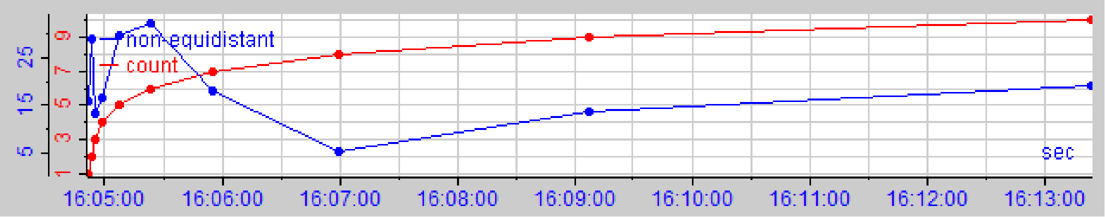
	SignalName	Expression	Comment 1
1	rnd	(Rand(200000,0.01) +1) / 32768	
2	firstrnd	XCutRange([rnd],0,1000)	
3	secondrnd	XCutRange([rnd],1000,2000)	
4	expr	Sqrt(-2.0*Log([firstrnd]))*Sin(2*PI()*[secondrnd]) *1.5	normally distributed random data
5	skewness	Skewness([expr])	global skewness
6	skewness_time	SkewnessInTime([expr],100)	skewness at fixed intervals
7	moving_skewness	XMarkValid(MSkewness([expr],100) , XValues([expr]) > 10)	skewness over gliding interval
8	skewness_valid	SkewnessValid([expr],Sin(Time(100000,0.01)/10)>0)	skewness where a control signal is true

CountSamples

For equidistantly sampled data, samples could adequately be counted with the ibaAnalyzer ‘*XSize*’ and ‘*XSizeValid*’ functions. For non-equidistant data this does not work, therefore the ‘CountSamples’ function was added. Contrary to ‘*XSize’* the function does not return a constant value but returns a signal that for every timestamp in the original signal returns the number of samples counted so far. The function takes the following parameters:

- **CountSamples(‘Expression’, ‘Reset’)**:
  - **Expression**: The signal whose samples need to be counted.
  - **Reset**: A control signal that resets the count to zero each time the signal is TRUE. This parameter can be omitted in which case the count is never reset.

Note that the function does not count invalid samples; hence it can be used in combination with XMarkValid to count samples satisfying a condition.



Text channels export/import to ASCII file.

Exporting/extracting text channels to an ASCII file.

It is now possible to export and extract text channels to an ASCII file.
Since every line of the ASCII file corresponds to a timestamp that differs a constant multiple of the selected time base with the initial reference timestamp, the timestamps of the text samples will be rounded to the largest possible timestamp earlier or equal than the timestamp of the text sample. Since generally there are more timestamps in the ASCII file than there are text samples, the column of the text channel will be empty for most of the rows. If there would be multiple text samples that are rounded to the same timestamp, only the first text sample will be present in the ASCII file. The text sample entries in the ASCII file will be quoted (`“”`) so that the text samples can contain tabs (i.e. the character used as delimiter between the fields) without breaking the ASCII file formatting.

Importing an ASCII file with extracted text channels.

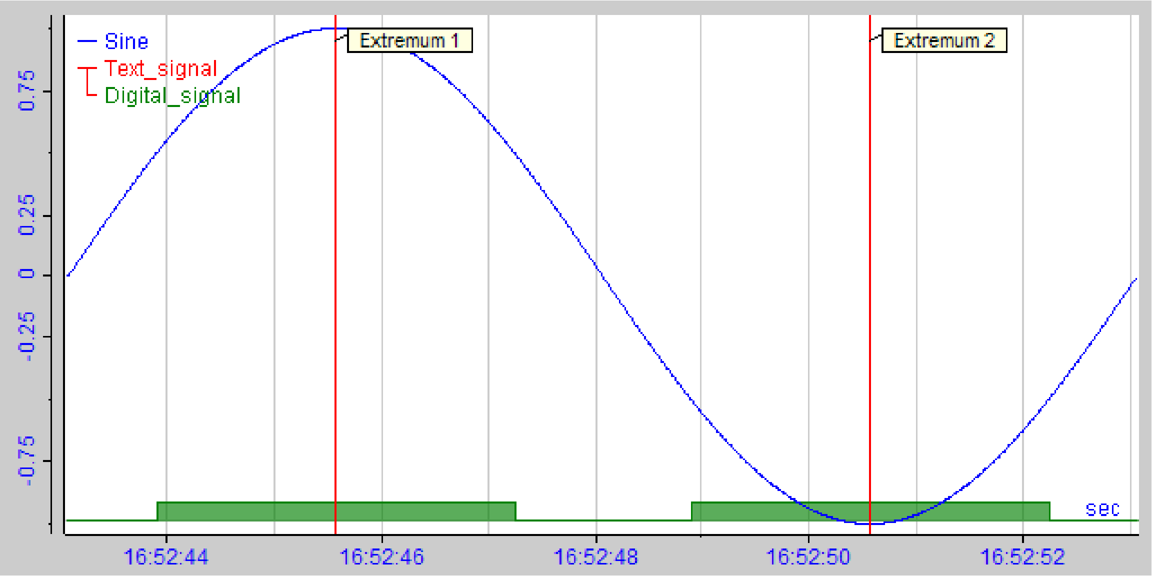
An extracted ASCII file can be imported back in ibaAnalyzer if the necessary license bit is set in the iba licensing dongle. If such an ASCII file contains text channels they will be imported as well. If you wish to import text channels from another source than ibaAnalyzer through an ASCII file, make sure that the ASCII file contains a correct header row, the header row has the same number as columns as the data rows and each field in the header row is formed as follows:

- The word “`Time`” for the time stamps column (if present).
- The word “`Length`” for the positions column (only present when extracting length based channels with a length profile)
- For an analog signal, a square opening bracket followed by the module number, a colon, the order number in the module and a square closing bracket.
  E.g.: `[5:0]`
- For a digital signal, a square opening bracket followed by the module number, a point, the order number in the module and a square closing bracket.
  E.g.: `[5.1]`
- For a text channel, the same code as an analog signal is used; however it is followed by “_text”.
  E.g.: `[4.1]_text`

Example of an exported ASCII file containing a text channel

The following file listing is an example of an exported file with an analog, text and digital channel, exported with a time base of one second:

```
Time                          [0:0]      [0:1]_text     [0.0]
time                          Sine       Text_signal    Digital_signal
sec
30.08.2010 16:52:43.070000    0                         0
30.08.2010 16:52:44.070000    0,587785                  1
30.08.2010 16:52:45.070000    0,951057   "Extremum 1"   1
30.08.2010 16:52:46.070000    0,951056                  1
30.08.2010 16:52:47.070000    0,587785                  1
30.08.2010 16:52:48.070000    -8,74E-08                 0
30.08.2010 16:52:49.070000    -0,58779                  1
30.08.2010 16:52:50.070000    -0,95106   "Extremum 2"   1
30.08.2010 16:52:51.070000    -0,95106                  1
30.08.2010 16:52:52.070000    -0,58779                  1
```


Sine
Text_signal
Digital_signal
Extremum 1
Extremum 2
sec
16:52:44
16:52:46
16:52:48
16:52:50
16:52:52