Automotive
Energy & Power Analysis
Field Service
Environmental
Research & Development

DEWETRON

Training Manual Math

Version 2.1
September 2011

worldwide
DEWETRON
www.dewetron.com

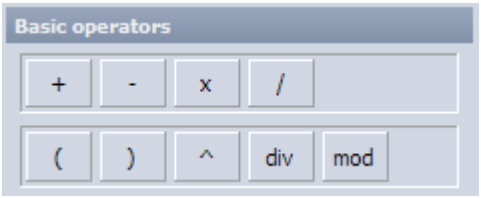
Table of contents

MATH - OVERVIEW

Formula

Most of the functions below are described with discrete numbers instead of channel names for easier understanding.

Basic Operators



General mathematics		
Function	**Syntax**	**Description**
+	*expression1* + *expression2*	*expression1* plus *expression2*
-	*expression1* - *expression2*	*expression1* minus *expression2*
×	*expression1* * *expression2*	multiplies *expression1* by *expression2*
/	*expression1* / *expression2*	divides *expression1* by *expression2*
( )	(*expression*)	brackets *expression*
^	*expression1* ^ *expression2*	raises *expression1* to the power of *expression2*

DIV – Division (Integer)

div: integer division operation

105.3 DIV 10 = 10 integer-value of 10.53 is 10

MOD – Modulus function

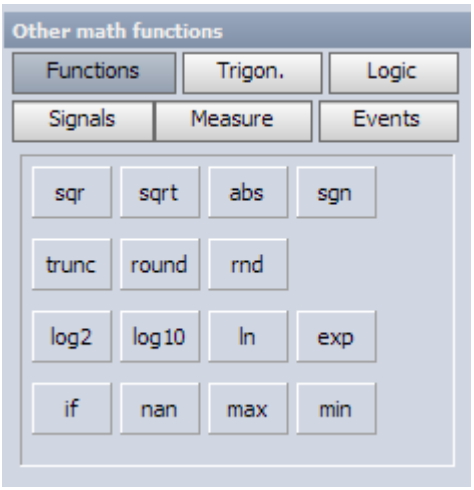
mod: delivers the integer remainder of a mathematical integer division operation

420 MOD 720 = 420 420 cannot be divided by 720, and we will get the remainder of the division which is 420.

740 MOD 720 = 20 740 can be divided by 720 one time, with a remainder of 20.

105.3 MOD 10 = 5 105 can be divided by 10 10 times, with a remainder of 5.3. The integer value returned is 5.

Functions



SQR -Square

sqr: The square of a number is the number multiplied with itself. No matter if the number is positive or negative the result will always be positive.

sqr(4)= 16

sqr(-3)= 9

SQRT - Square root

sqrt: The square root of number B is the number A^2. So it is the inverse function of SQR.

sqrt(16) = 4

sqrt(9)= 3

Info: SQRT of a negative number sqrt(-9) will deliver always 0 instead of a complex number in DEWESoft.

ABS – Absolute value

abs: Calculates the absolute value of number or a channel.

ABS(45.34) = 45.34

ABS(-33.12) = 33.12

ABS(0) = 0

SGN – Sign

sgn: Extracts the sign of a channel or a number.

Sign function delivers 0 if input channel or number is 0.

sgn(-8.124) = -1

sgn(19.345) = 1

sgn(0) = 0

TRUNC – Truncate function

trunc: Converts a number into integer, every number or channel which is converted with the trunc function will lose the part after the comma.

trunc(1452.457) = 1452

trunc(-1452.457) = -1452

It is not rounded up or down, so both:

trunc(86.248) = 86 and also

trunc(86.848) will give 86.

ROUND – Round function

round: rounds a number or channel depending on the digits after the comma to an integer value.

3.xxx if xxx is bigger or equal 0.5 it will be rounded up to the next integer value.

round(14.43) = 14

round(14.501) = 15

round(-14.492) = -14

round(-14.51) = -15

Hint: *if you want to round 136.3724 to 140 simply divide the value or channel by 10, round it and multiply the result by 10.*

round(136.3724 / 10) * 10 = 140

RND – Random

rnd: creates random numbers with the selected sampling rate between 0 and 1. So if a sample rate of 1000Hz is selected, 1000 values per second are created.

LOG2 – Logarithm base 2

log2: calculates the logarithm (base 2) of a number or an input channel.

Log2(8) = 3

Log2(a) = b the logarithm extracts b from an equation $2^b = a$.

LOG10 – Logarithm base 10

log10: calculates the logarithm (base 10) of a number or an input channel.

Log10(100) = 2

Log2(a) = b the logarithm extracts b from an equation $10^b = a$.

LN – Natural logarithm base e

ln: calculates the natural logarithm (base e=2.71828…) of a number or an input channel.

LN(100) = 2

LN(a) = b the logarithm extracts b from an equation $2.71828..^b = a$.

EXP – Expotential function of e

exp: calculates the exponential function of e from a number or an input channel.

EXP(1) = 2.71828…

EXP(b) = a the logarithm extracts b from an equation $2.71828..^{b} = a$.

IF – Conditional output

if(condition,result1,result2)

Outputs either the result1 or result2 depending on condition. If condition is true, result1 is output, else result2 is output; condition must be a logical operation, which gives true or false.

Example: IF('ID'>=1,'velocity','displacement')

'ID', 'velocity', and 'displacement' are analog input channels.

If the ID channel is equal to or greater than 1, the statement is true and the velocity channel will be chosen as output, otherwise displacement. So either velocity or displacement are used for output depending on the condition, both at the same time are not possible.

NAN – not a number function

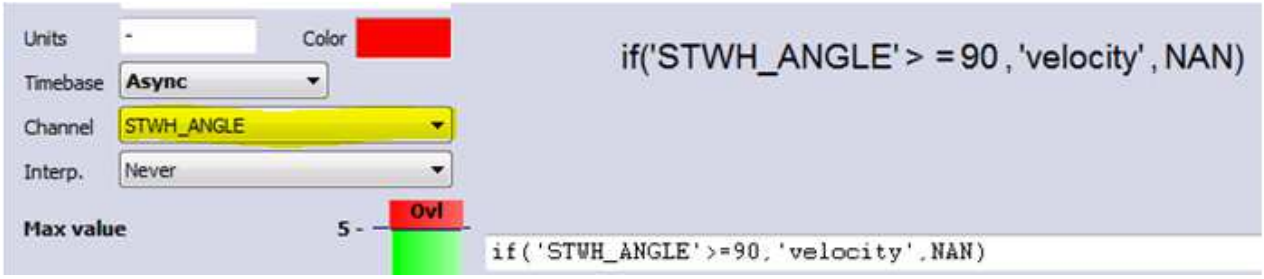
nan(not a number)

if('STWH_ANGLE'>=90,'velocity',NAN)

'STWH_ANGLE' = CAN Channel (asynchronous)

'Velocity' = analog input channel (synchronous)

The channel will output NAN only if STWH_Angle is smaller than 90 deg; the timebase of the math channel will be forced to asynchronous output. Otherwise because a synchronous channel is used in the formula (velocity) the formula will output a synchronous channel and NAN will become zero.



The picture above is showing the time base setting of the math channel.

The asynchronous channel has to be used in the formula otherwise it is not possible to select it in the channel selector. The interpolation has to be set to never.

MAX / MIN – Maximum / Minimum function of more channels

max/min: max(Channel1,Channel2) checks both channels and outputs the maximum /minimum value of one of the channels.

Example: max('pressure1','pressure2') - pressure1 and pressure2 are two analog input channels.

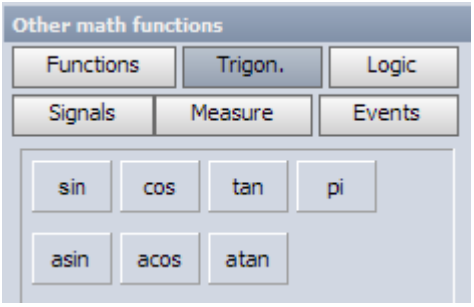
pressure1	3	4	6	8
pressure2	2	5	4	7
output	3	5	6	8

The higher value if both channels will be output.

INFO: Also multiple max could be used in one formula.

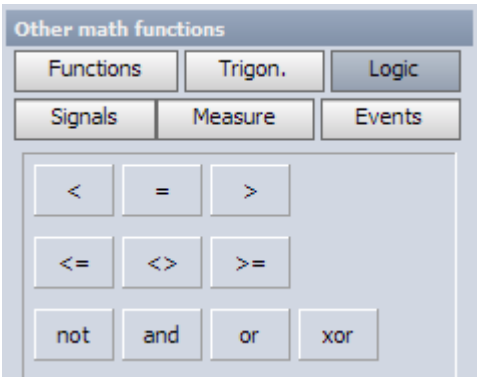
Max(max('pressure1','pressure2'),'pressure3')

Trigonometry



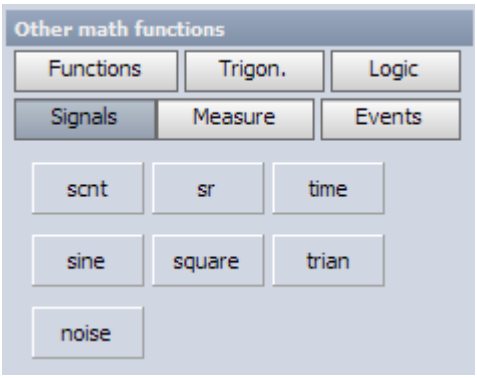
Trigonometric functions		
Function	**Syntax**	**Description**
SIN	SIN(*expression*)	sine of *expression*
COS	COS(*expression*)	cosine of *expression*
TAN	TAN(*expression*)	tangent of *expression*
PI	PI	constant pi
ASIN	ASIN(*expression*)	arcsine of *expression*
ACOS	ACOS(*expression*)	arccosine of *expression*
ATAN	ATAN(*expression*) or ATAN(x,y)	arctangent of *expression* or arctangent of x/y

Logic



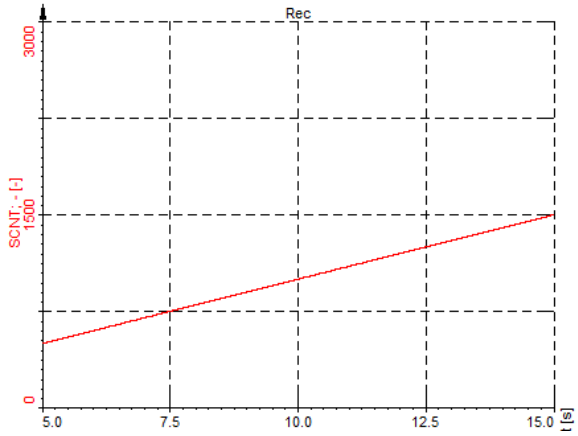
Logic functions		
Function	**Syntax**	**Description**
<	*expression1* < *expression2*	if *expression1* is less than *expression2* then output is 1, else 0
=	*expression1* = expression2	if *expression1* is equal to *expression2* then output is 1, else 0
>	*expression1* > *expression2*	if *expression1* is greater than *expression2* then output is 1, else 0
<=	*expression1* <= *expression2*	if *expression1* is less than or equal to *expression2* then output is 1, else 0
<>	*expression1* <> *expression2*	if *expression1* is less or greater than *expression2* then output is 1, else 0
>=	*expression1* >= *expression2*	if *expression1* is greater than or equal to *expression2* then output is 1, else 0
NOT	NOT expression	negation; *expression* = 0: 1; *expression* = 1: 0;
AND	*expression1 AND expression2*	logic and 1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0
OR	*expression1 OR expression2*	logic or 1 OR 1 = 1 1 OR 0 = 1 0 OR 1 = 1 0 OR 0 = 0
XOR	*expression1 XOR expression2*	logic exclusive or 1 XOR 1 = 0 1 XOR 0 = 1 0 XOR 1 = 1 0 XOR 0 = 0

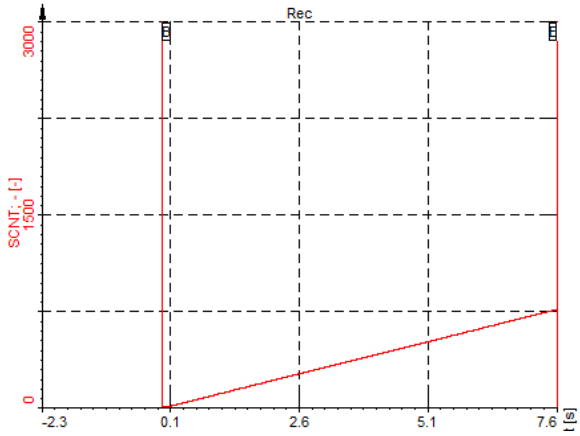
Signals



SCNT – Sample counter

scnt: delivers the samples acquired from start of the measurement. The counter will be reset at the start of storing.





The left picture is showing the result, while at the right picture the reset after start storing could be seen.

SR – Sample rate

sr: sample rate of data acquisition

Time – Time function

time: is providing the elapsed time of the measurement in seconds.

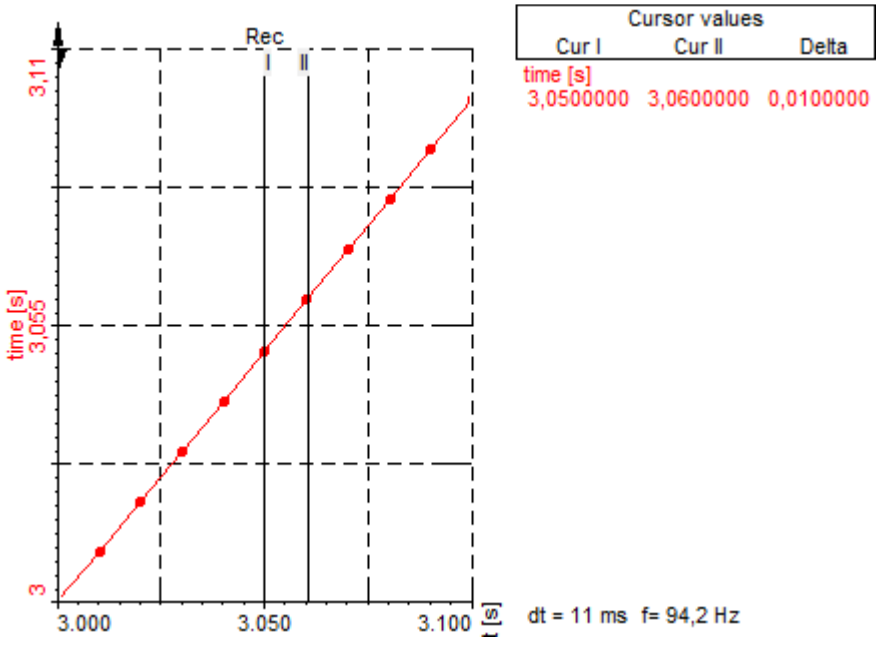
This is a function similar to SCNT. The only difference is, instead of samples
It outputs the time in seconds, regardless which sampling rate is used.

Note: the TIME function is reset at the start of storing.

The resolution is linked with the sampling rate.

The screen shot below shows the time function in a recorder,
We can see that the resolution is 0.01s and also dt=10ms which results in a sampling rate of 100 Hz.

So the time will count up 0.01s after every sample @ sampling rate of 100 Hz or 0.001s @ sampling rate 1 kHz



Sine – Sine wave

sine: creates sine wave signal with amplitude x, frequency f and optional phase shift in radian

x * sine(100, pi)

gives a sine wave with 100 Hz and shifted by 180°

Square – Square wave

square: creates square wave signal with frequency f and optional phase shift in radian

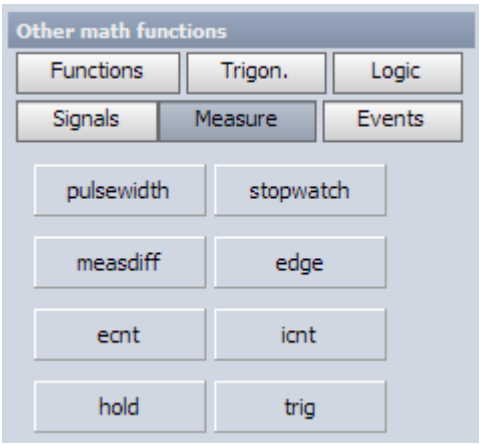
Trian – Triangular wave

trian: creates triangular wave signal with frequency f and optional phase shift in radian

Noise – Noise signal

noise: creates random white noise signal with amplitude from -1 to +1

Measure



Pulsewidth

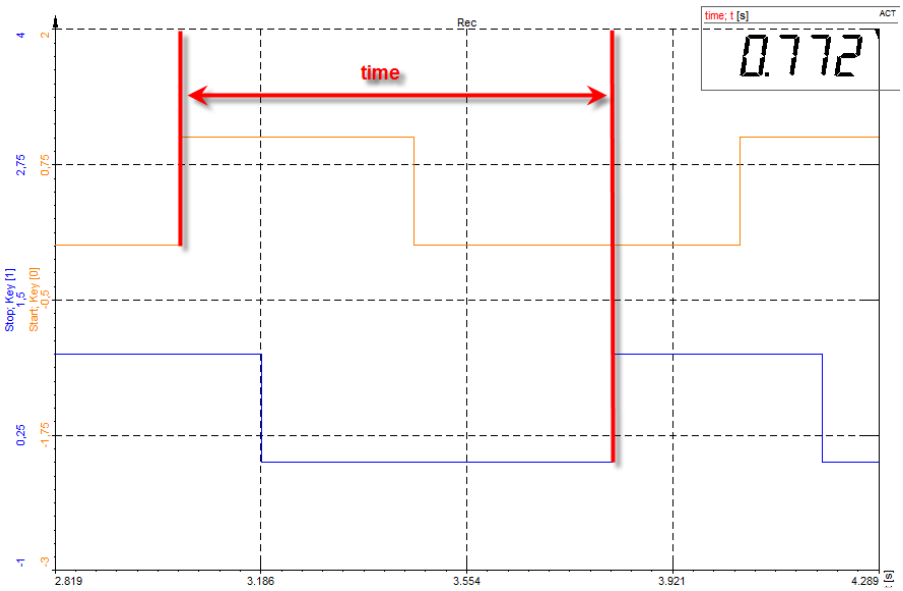
pulsewidth(cond [, rearm])

Measures the time in seconds between two condition edges (where cond changes from 0 to 1), rearm edge optional.

Stopwatch – Stopwatch function

stopwatch: **stopwatch(Condition1,Condition2)** measures the time in [s] between condition1 and condition2. Condition is a logic value that jumps from 0 to 1 (edge sensitive).

Example: **stopwatch('Start'>0.5,'Stop'>0.5)** Start and Stop could be analog or digital signal. The logic comparison '>' turns the condition into a logic value anyway.



Measdiff – measure difference

measdiff(value, cond1, cond2)

Measures value difference between cond1 and cond2 edges (where cond jumps from 0 to 1).

Edge .- detect signal edge

edge(cond [, rearm])

Returns 1 when the cond changes from 0 to 1, with optional rearm condition

Ecnt – Edge counter

ecnt(cond)

Counts number of edges for cond (where cond jumps from 0 to 1).

Icnt – Count high values

icnt(cond)

Counts all the samples where the condition has logical value 1.

Hold – Hold function

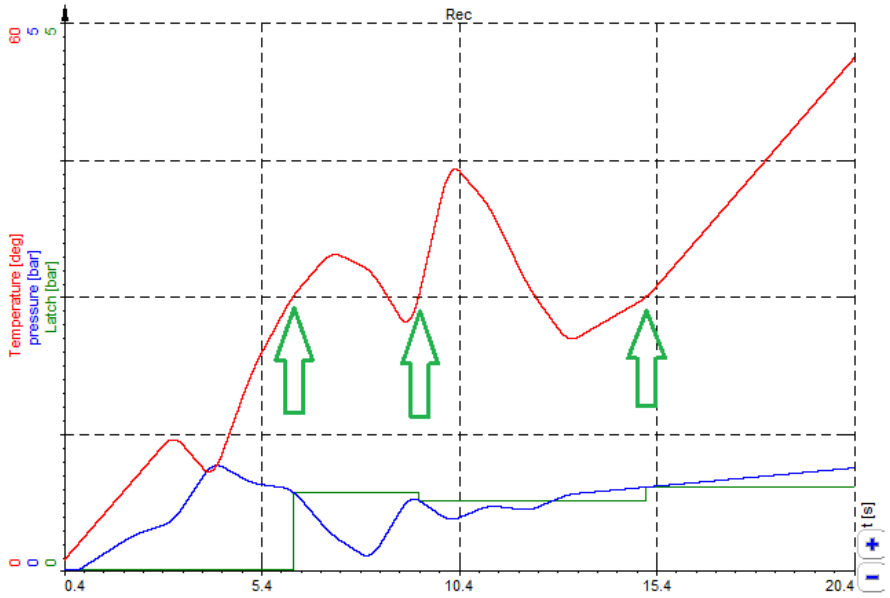
hold(value,latchcondition, [rearmcondition])

The hold function is used to latch or hold a single value if a condition is met.

Example1: Hold('pressure', 'Temperature'> 30)

Hold(value, latchcondition)

In the example above the function will hold the actual pressure if the temperature is higher or equal to 45 degrees.

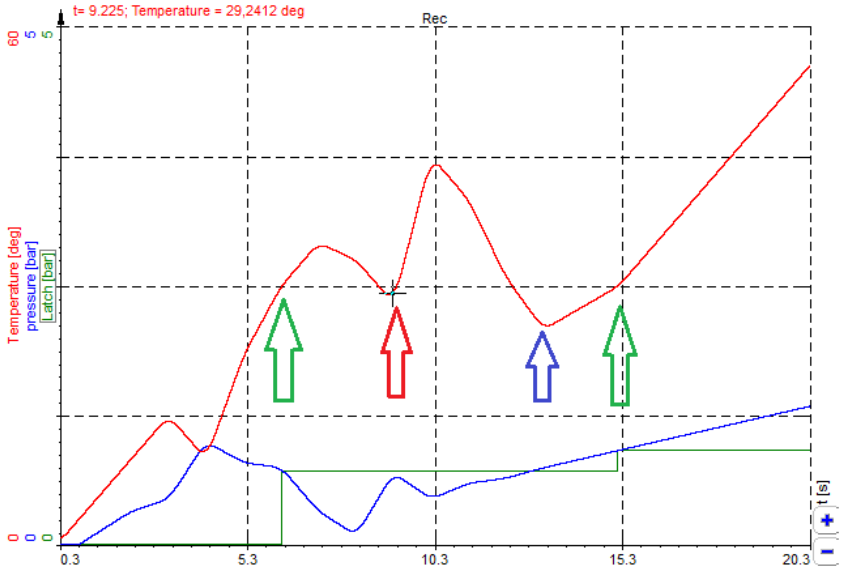


The picture above is showing the function in a recorder. All the time when the temperature exceeds 30 deg (green arrows), the actual pressure channel is latched.

Example2: Hold('pressure', 'Temperature'> 30, 'Temperature'< 28)

Hold(value, latchcondition,[rearmcondition]) optional.

The function can be extended with a rearm condition. After a latch condition occurs, the ream condition has to be met first before a new latch condition and therefore a new latch can occur.
This is used to filer the latch condition a little bit. Imagine the latch condition channel has noise on it, or is fluctuating around the level (30 deg +-0.2 deg) which would cause a unintentional LATCH.



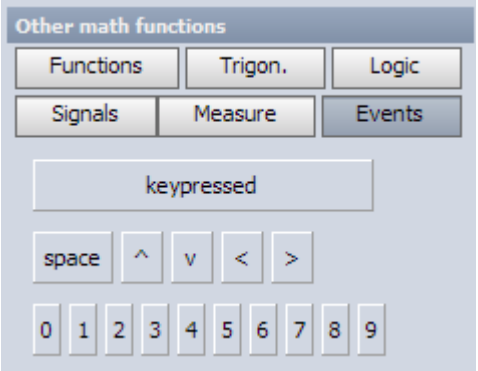
The picture above illustrates this function:

1.) Temperature rises above 30 deg → LATCH 1 green arrow
2.) Temperature goes below 30 deg and back above 30 deg → no LATCH because temperature did no go under 28 deg therefore the LATCH did not occur. Red arrow.
3.) Temperature goes below 28 deg → rearm condition is complied
4.) Temperature exceeds again 30 deg → LATCH is performed again

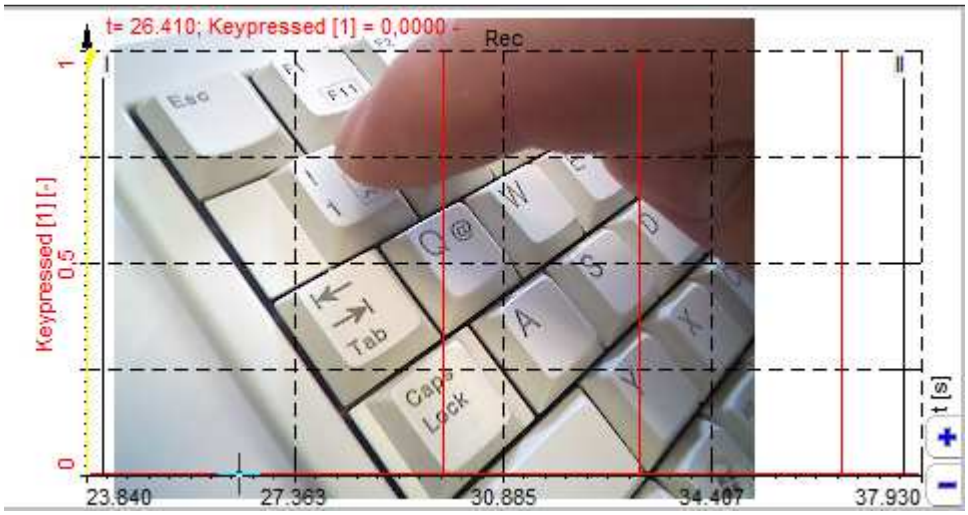
Trig

trig: has value of 1 when a store condition appears (1 pulse)

Events



Keypressed – Keypressed function



Almost any key can be used in the keypressed function.
The value in the brackets is reflecting the virtual key code in decimal.
Below you will find a list of the most popular keys. You have to convert them from Hex to decimal. So [Key 1] → 31hex → 49 dec. . The decimal value has to be entered Into the keypressed function. keypressed(49).

Major Key codes

0x41 ('A')	A
0x42 ('B')	B
0x43 ('C')	C
0x44 ('D')	D
0x45 ('E')	E
0x46 ('F')	F
0x47 ('G')	G
0x48 ('H')	H
0x49 ('I')	I
0x4A ('J')	J
0x4B ('K')	K
0x4C ('L')	L
0x4D ('M')	M
0x4E ('N')	N
0x4F ('O')	O
0x50 ('P')	P
0x51 ('Q')	Q
0x52 ('R')	R
0x53 ('S')	S
0x54 ('T')	T
0x55 ('U')	U
0x56 ('V')	V
0x57 ('W')	W
0x58 ('X')	X
0x59 ('Y')	Y
0x5A ('Z')	Z

0x60	Numpad 0
0x61	Numpad 1
0x62	Numpad 2
0x63	Numpad 3
0x64	Numpad 4
0x65	Numpad 5
0x66	Numpad 6
0x67	Numpad 7
0x68	Numpad 8
0x69	Numpad 9
0x2E	Delete
0x28	Arrow Down
0x23	End
0x70	F1
0x79	F10
0x7A	F11
0x72	F3
0x73	F4
0x74	F5
0x75	F6
0x76	F7
0x77	F8
0x78	F9

INFO: Do not use F2 because it is used as shortcut to enter Setup.

If you need any other key which is not present in the upper list, search for “key codes” or “ASCII table” in the internet.

Display screen selection

<F2> Switch to Setup screen

<F3> or <SHIFT> + <F1> Switch to Overview screen

<SHIFT> + <F2> Switch to Scope screen

<SHIFT> + <F3> Switch to Recorder screen

<SHIFT> + <F4> Switch to Vertical recorder screen

<SHIFT> + <F5> Switch to FFT screen

<SHIFT> + <F6> Switch to Power screen

<SHIFT> + <F10> Switch to Video screen

<SHIFT> + <F11> Switch to GPS screen

Filters

FIR and IIR filters

A finite impulse response (FIR) filter is a type of a signal processing filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which have internal feedback and may continue to respond indefinitely (usually decaying).

Comparison between IIR and FIR filter

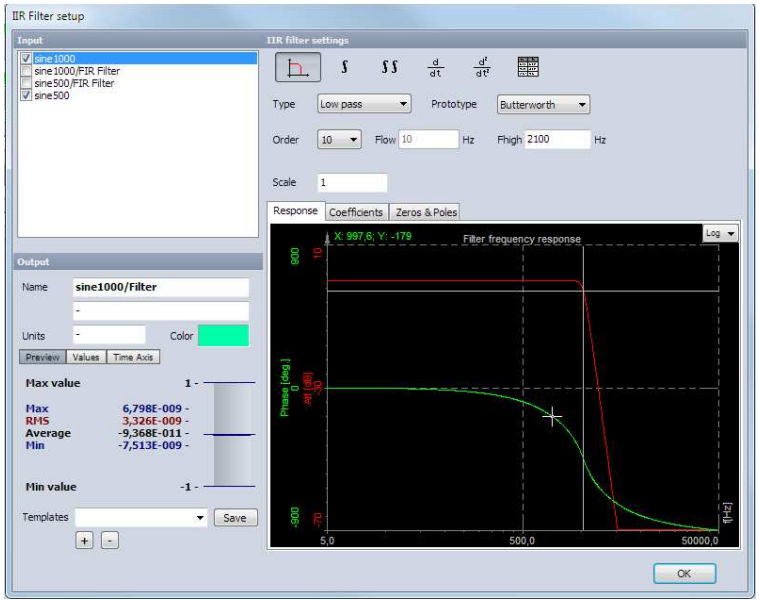
	IIR filter	FIR filter
fast (less computational cost)	✓	
always stable		✓
linear phase in transmission band		✓
high stop-band attenuation with low order	✓	
less ripple in pass-band		✓
can be used for integration and derivation	✓	

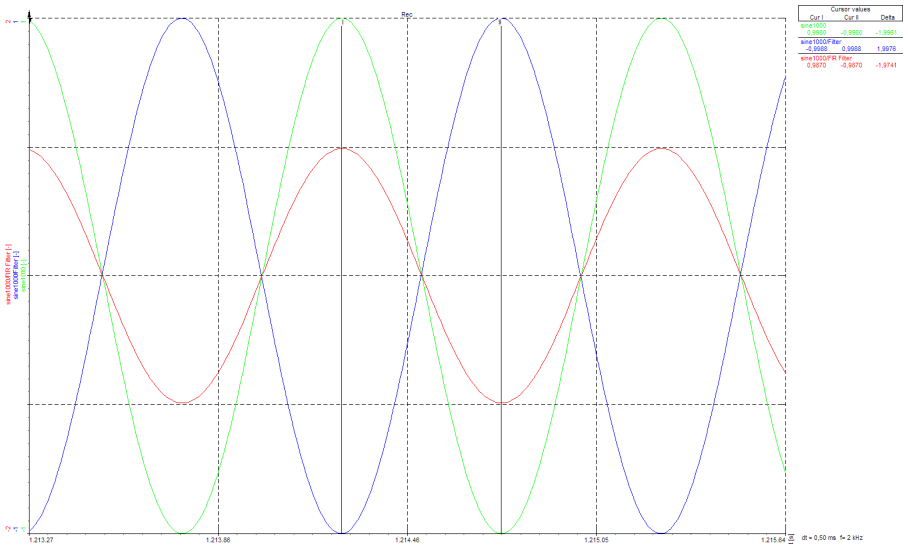
Phase response

In the filter setup screen, an amplitude and phase diagram is shown, where the filter characteristics for the actual filter parameters can be previewed.

IIR

Below a butterworth lowpass filter is shown with upper frequency of 2100 Hz, 10th order. The phase diagram shows -180° for 1 kHz, which can also be verified in a simple measurement.

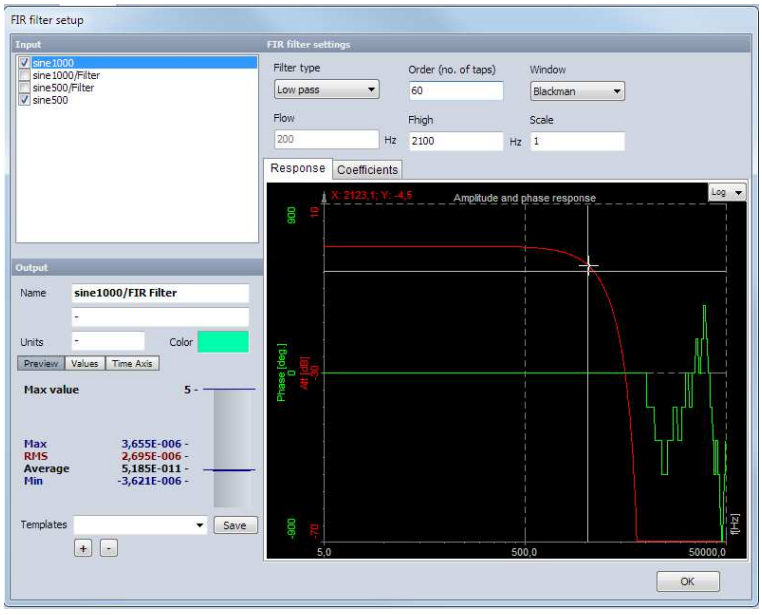




green curve: original signal, sine wave 1 kHz
blue curve: IIR low-pass filtered signal, phase response of -180 ° at 1 kHz (in pass-band)
red curve: FIR low-pass filtered signal, linear phase response in pass-band (amplitude scaled differently for presentation)

FIR

Although the phase is linear in transmission band, the amplitude is already -4 dB lower at cut-off frequency (at order 60, see below).



Hint:

If signal timing is critical and filtered and unfiltered channels are compared, use FIR filters. The overall filter-delay is compensated inside DEWESoft and because of linear phase in transmission band no frequency dependant additional delay is created. If each timing-critical channel is filtered with the same filter and then compared, also IIR filters can be used, since each channel has the exact same total phase shift (still frequency dependant, but the same for all channels).

IIR Filter types – overview

Butterworth:

The frequency response is maximally flat (has no ripples) in the pass-band and rolls off towards zero in the stop-band; −20 dB per decade per order.

It has a slower roll-off, and thus will require a higher order to implement a particular stop-band specification compared with a Chebyshev filter, but it has a more linear phase response in the pass-band.

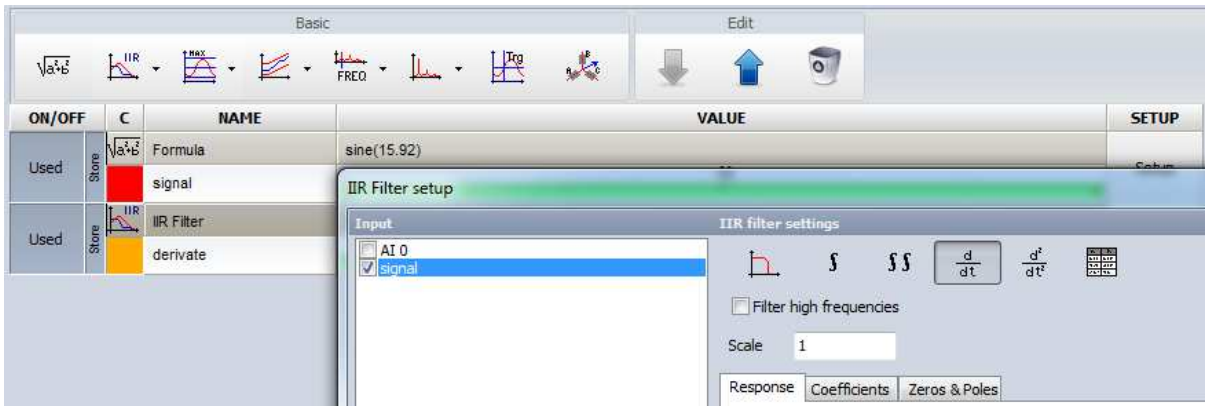
Chebyshev:

Chebyshev filters have a steeper roll-off but more pass-band ripple than Butterworth filters.

Bessel:

This filter is a type of linear filter with a maximally linear phase response across the entire pass-band, thus preserving the wave shape of filtered signals in the pass-band.

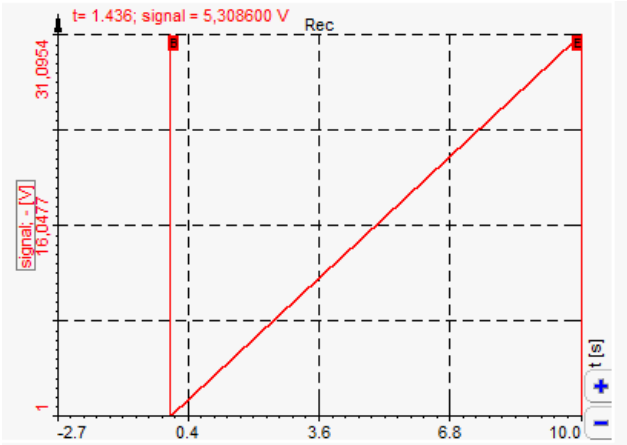
IIR filter - Derivation and dY derivation of a signal



Derivation describes the maximum voltage change of a signal.

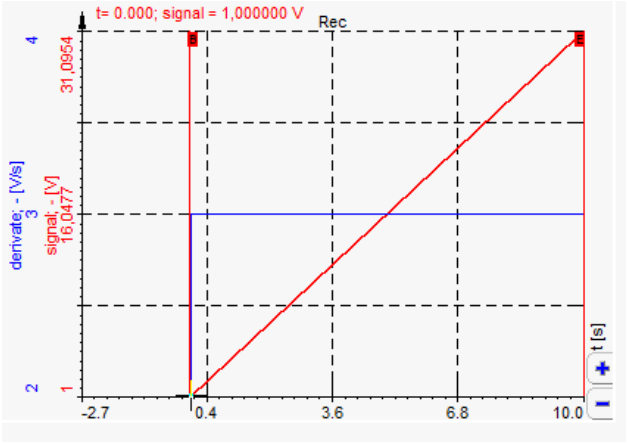
If we have a function:

s(t)= 3*t + 1 this function will create a signal shown in picture below.

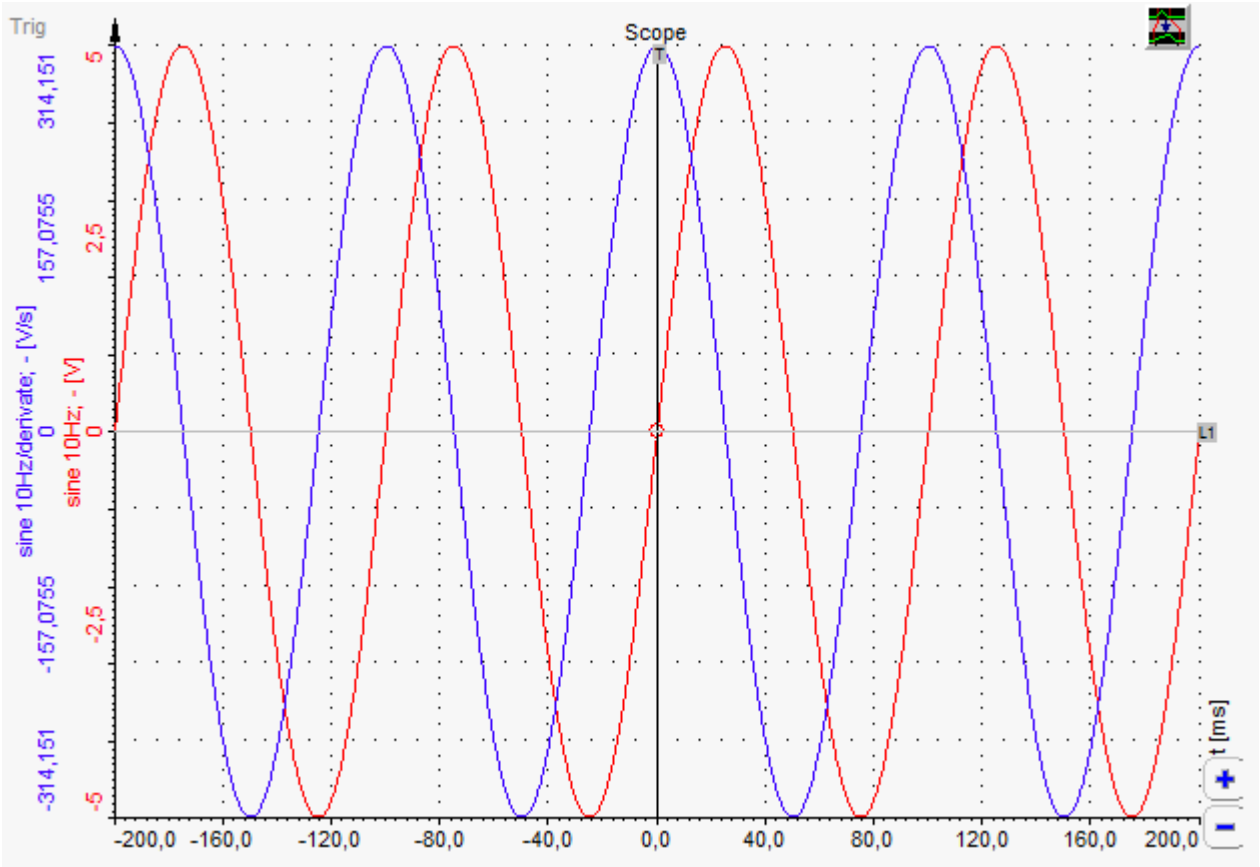


So the signal will start @ 1 and will count up 3 units after 1 second.

The derivation of this signal will be 3V/s.

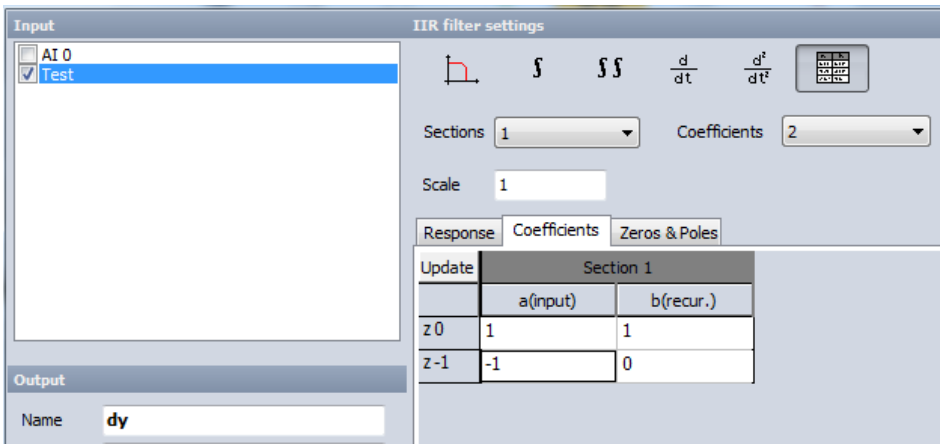


The next example shows the derivation of a sine signal.
A sine signal which is described as 5 * sine(10) will create a sine wave with 5 V amplitude and 10 Hz.
Derivation will yield: -5 * 2 * pi * 10 * cos(10) -- therefore a cosine with an amplitude of 314.15 V/s.



The filters can be also used to calculate the dY of a signal.

Example: a signal with voltage 3,5,7,8,12,15,…. will result in (5-3)=2,(7-5)=2,(8-7)=1,4,3 dY……



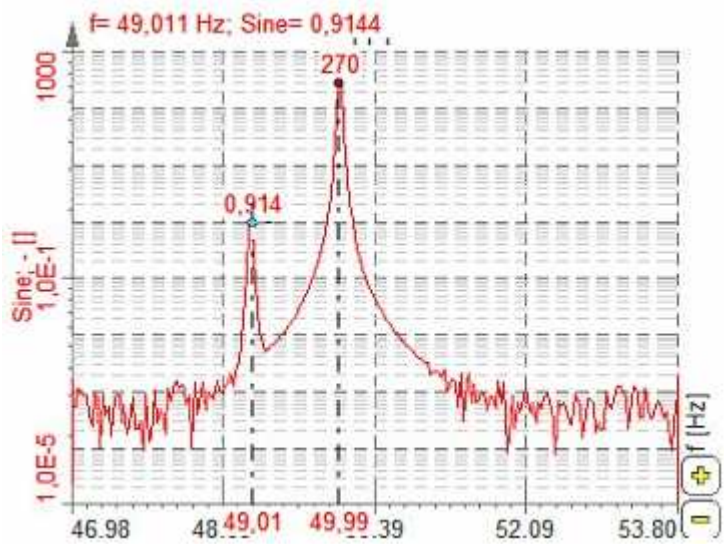
For this we have to manually enter coefficients into a filter, like shown in the upper picture. In comparison to Derivation, here only dY is calculated where at derivation dY/dt is calculated. So in simply word, dY is divided with dt = sample rate.

FFT filter

FFT filter is quite different from other types of filters. While IIR and FIR filters are time domain filters, FFT filter calculates the spectrum of the signal with a specific number of lines and overlap and then extracts the RMS value of certain range of this signal. Therefore the result is not the full curve, but only one value per frequency spectrum.

The usage of this filter is to extract low peaks of signals where there are big harmonics nearby. It wouldn't be possible to use IIR filter to extract amplitudes this low.

The example below shows the electro motor winding failure which can be seen as low values at the rotation frequency where the line frequency is very high:



If we choose Tracking Frequency source, we need to enter the Frequency channel and Number of harmonics instead of center frequency.

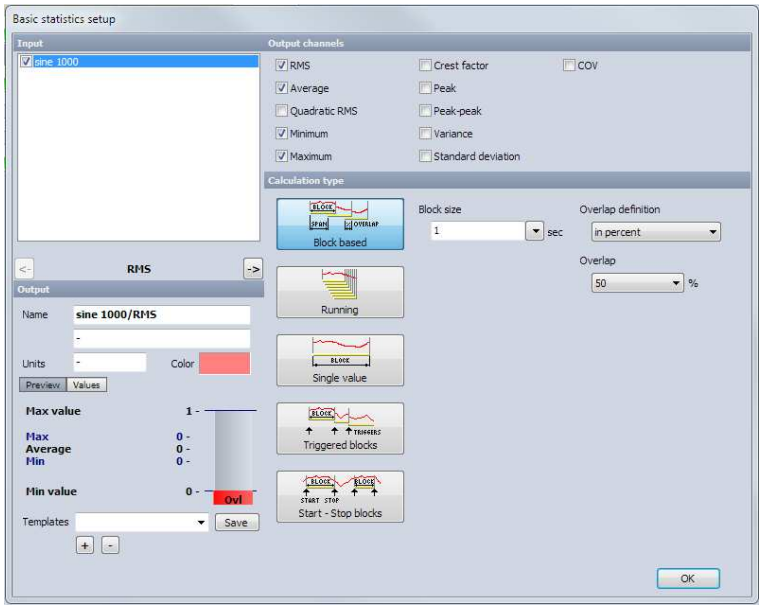
Statistics

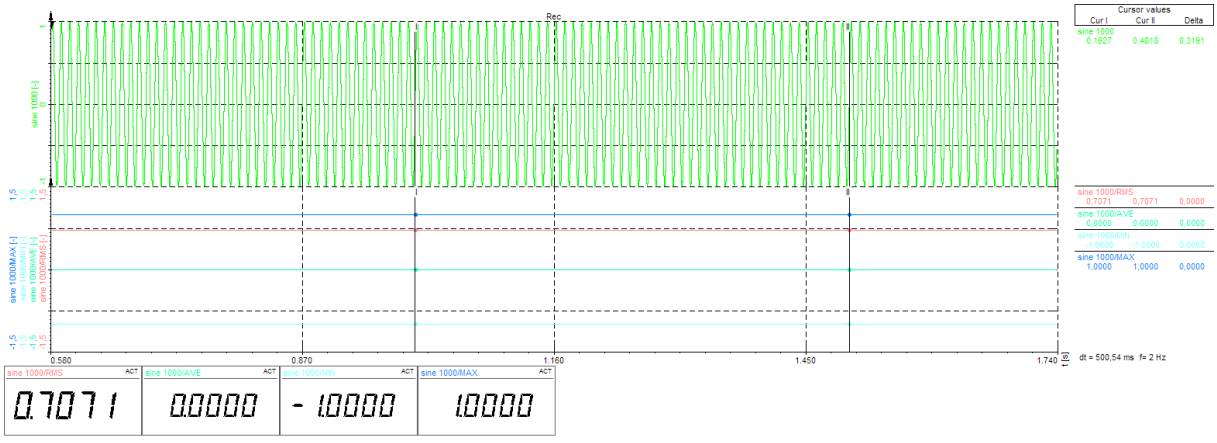
Basic statistics

There are several modes how the statistic values (RMS, average, quadratic RMS, minimum, maximum, crest factor, peak, peak-peak, variance, standard deviation, covariance) can be calculated.

Block based average

Blocks of defined size (block size in seconds) with defined overlap are used to calculate average values. Here, one second block size with 50 % overlap gives one statistic value every 0.5 s, as seen in the screenshot below.



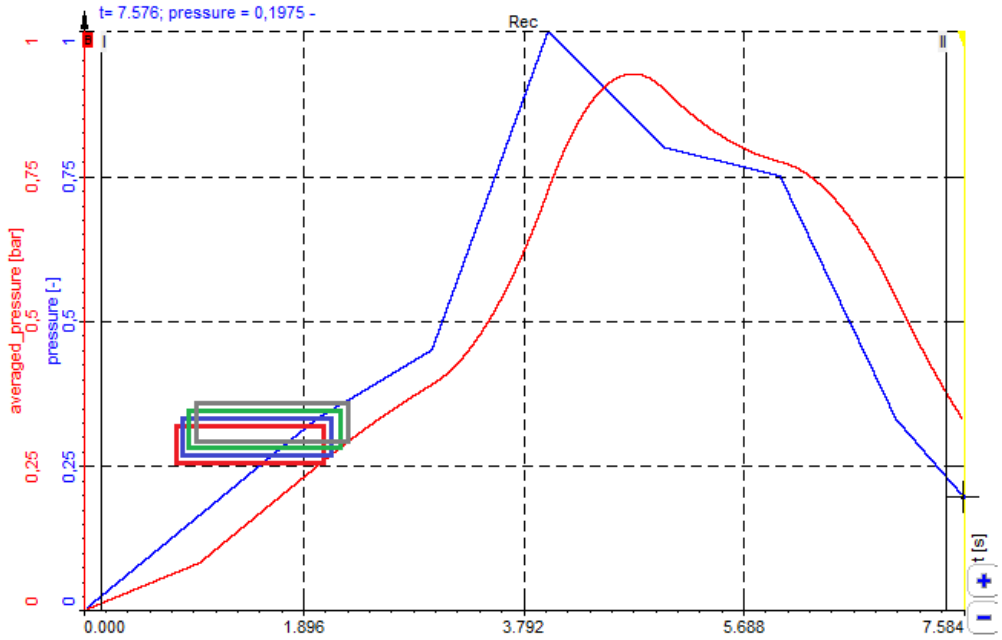


Running average

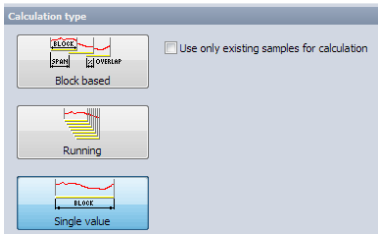
Here in this example the RUNNING mode over a block size of 1s is chosen.

You can visualize the calculation like this:

A window with a length of 1s is moved over the channel to average. All the data in this window will be averaged and a single average value will be the result (indicated by the red, blue, green, and gray boxes in the picture below). After that, the window will be moved for one sample to the right and the average will be calculated again. So the output will be a synchronous channel with the full sample rate. The picture below will illustrate the function in more detail.



Single value



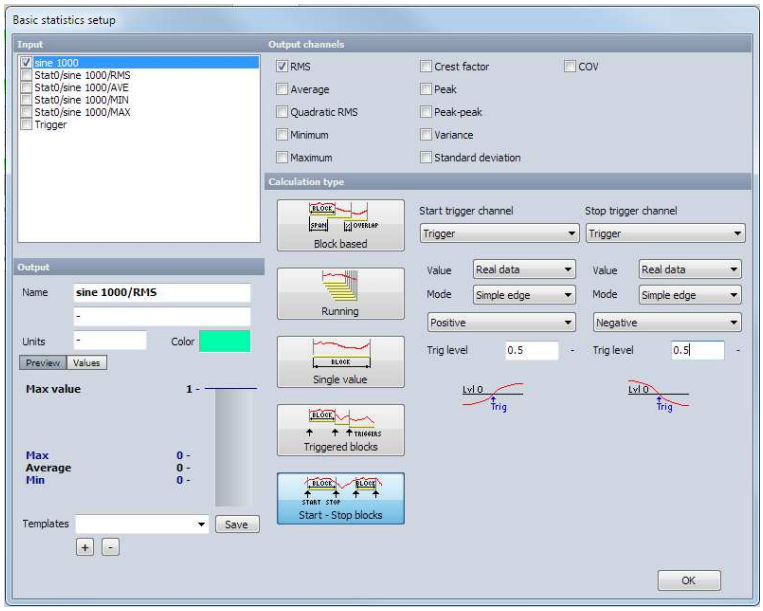
This setting gives one overall statistical value for all channel data. If the target channel is asynchronous, and only existing values (without interpolation) should be used for statistics, then make sure to check the”use only existing settings” check-box.

Triggered blocks

Triggered blocks option calculates the statistical value based on specific trigger event. When event is recognized, statistics calculation is started. When a second event is recognized, the calculation is stopped and the statistical value is written with its time stamp. After that the next value is calculated.

Start – Stop blocks

This feature works the same as “triggered blocks”, except a different channel is specified as stop trigger.



Array statistics

The array statistics can calculate the statistical value from an array.

There are several options which can be chosen:

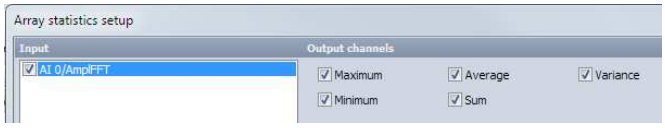
- **minimum** finds minimum value from the array.

There are two output channels created: class and value. Class will describe which index of the array holds the parameter and the value will be the minimum value itself.

- **maximum** finds maximum value from the array.

There are two output channels created: class and value. Class will describe which index of the array holds the parameter and the value will be the maximum value itself.

- **average** calculates average value of all elements from the array.
- **sum** calculates sum of all elements from the array.
- **variance** calculates the variance of all elements from the array.



Classification

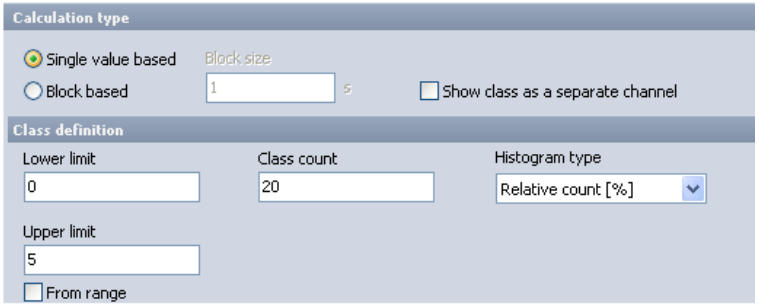
Classification is a procedure to count the values from the channel and assign them to classes. A classical classification from the primary school is to create the classes and count number of student with specific weight or height.

Classification in measurement field is used for various applications, for example to find the distribution of power grid frequencies with time or to find the distribution of sound levels to which certain area or working place is exposed to.

First of all we need to define what will be the result of classification. There are two options:

- **single value based**: the result will be one array holding the result of the entire run;
- **block based**: the result will be set of array each one added at the end of defined block size. If we have for example 2 seconds block size and acquire data for 10 seconds, we will get 5 arrays of classification values, each valid for 2 seconds of data.

Show class as a separate channel option will create single value channels for each of the class element. This is a nice option to display the values in the multi meter.



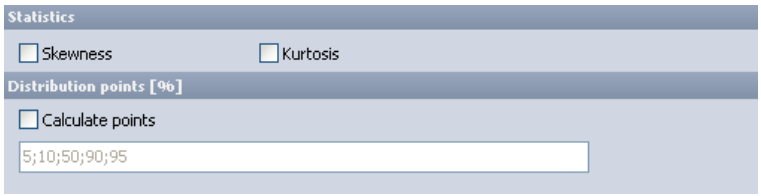
For class definition we need to set the:

- **lower limit**: this will set the lower limit for start counting - all values below this level will be counted in the first class;
- **upper limit**: this will set the upper limit for end of counting all values above this level will be counted in the last class;
- **class count**: defines the number of classes. In the example above the width of each class will be 5/20=0,25. First and last class will have half width, so it will go from from 0 to 0.125. Second class has a middle value of 0.25 and it goes from 0.125 to 0.375 and so on.

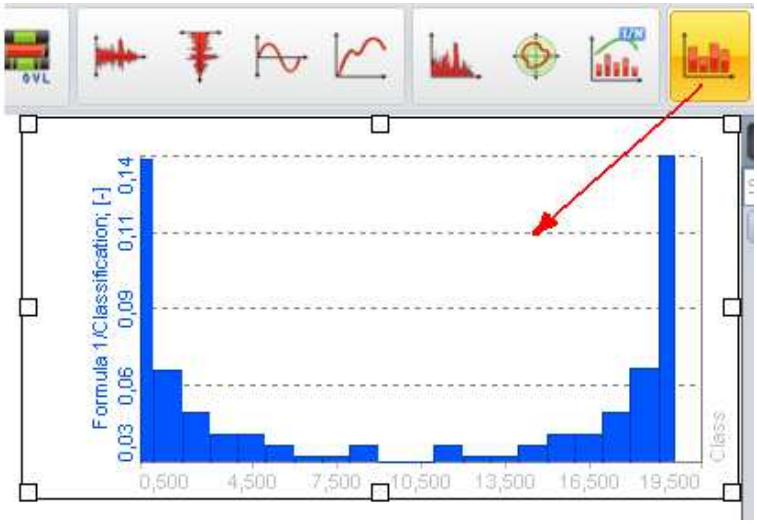
Histogram type defines what will be the output of the data (amplitude):

- **absolute count**: each class value has the number of samples within the class (value will always count up);
- **relative count**:each class value has the value of samples with the class normalized to total number of counted samples (sum of all classes will be always 1);
- **relative count [%]**: same as relative count, but expressed in percent (sum of all classes will be always 100);
- **density**: provides empirical probability density each class value has the number of samples normalized to total number of samples and divided by class width. In this case the value is not depending on number of classes within a range;
- **density [%]**: same as density, but multiplied with 100;

- **distribution**: provides empirical probability distribution, each class value has the sum of all lower classes and the number of current samples, normalized to total number of samples. The highest class has the value of 1.
- **distribution [%]**: same as distribution, but expressed in percentage. The highest class has the value of 100.



There are also several special output channels available. Two of them are skewness (asymmetry of probability distribution) and kurtosis (measure of "peakedness" of distribution). Additionally we can output a list of distribution point values. Distribution points are the class values at which distribution reaches entered value.
For the moment the distribution points works only if distribution is chosen as the histogram type.
Histograms can be seen in the 2D graph during measurement and analysis. If we choose block based calculation, we can use also 3D graph to display the history of classifications.

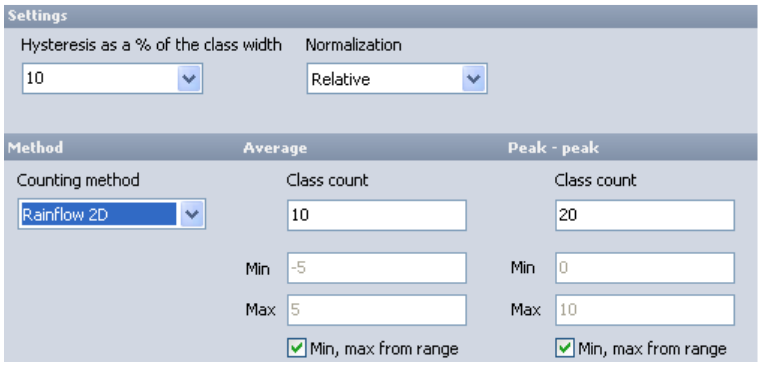


Counting

New Counting module can be added on the DEWESoft Math setup screen by selecting add Counting math button from Statistic group:

Counting is the standard procedure to reduce amount of data for analysis. It is for example used in application of road load data collection where we have some static load and on top dynamic load.

The only interesting values are the height of load cycles and the average static load of that cycle. For that purpose the rainflow analysis is made.



The counting procedures counts the peaks and valley of the signal. The Hysteresis is defined in percentage of class width. This prevents too much false counts if the signal is noisy.

There are three possible output values (normalization:

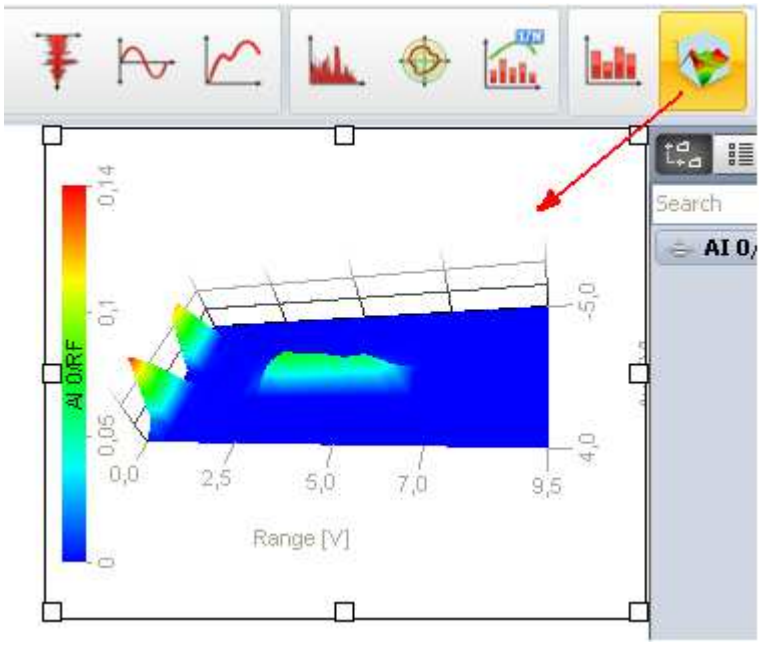
- **absolute:** it outputs number of cycles as a value - values will increase with time;
- **relative:** it outputs the number of cycles normalized to absolute number of cycles - sum of all values will be always 1;
- **relative [%]**: it outputs the number of cycles normalized to absolute number of cycles multiplied with 100 - sum of all values will be always 100;

There are several counting methods to choose from:

- **peak counting**:counts the number of peaks in the signal in certain classes (for this option we can define to count peaks, valleys or both);
- **range counting**: counts the range between successive peaks and valley pairs. Ranges are positive when slope between peaks and valleys is positive. We can choose either to count positive, negative directions or both).
- **level crossing**: counts number of times when that signal crosses various levels.
- **rainflow 1D**: counts range pairs. Rather than counting the peaks it splits the signal variations to smaller and larger pairs of values and counts all of them independently.
- **rainflow 2D**: same procedure as above, but it also calculates the average of each cycle and creates matrix of average vs. peak-peak value vs. number of cycles.

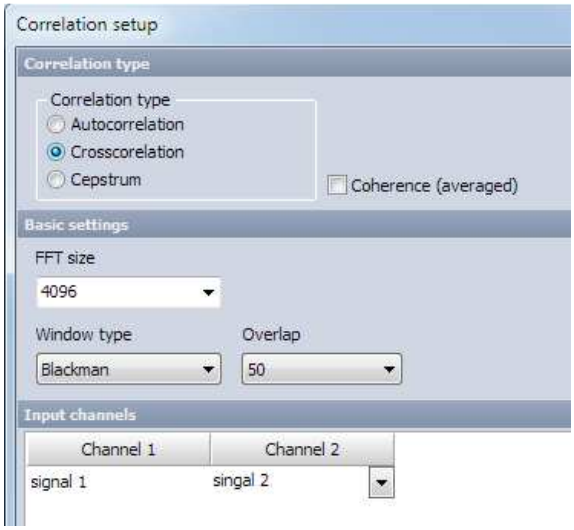
First four options outputs 2D matrix while the last one outputs 3D matrix.

For all options we have to define the number of classes for average value, minimum and the maximum or we choose to define the min and max value from the range of input parameters.
For 2D rainflow we have to define these parameters also for the third axis which counts peak-peak value.
The peak, range, level and 1D rainflow can be best seen in 2D graph while we need to use 3D graph for rainflow.



Correlation

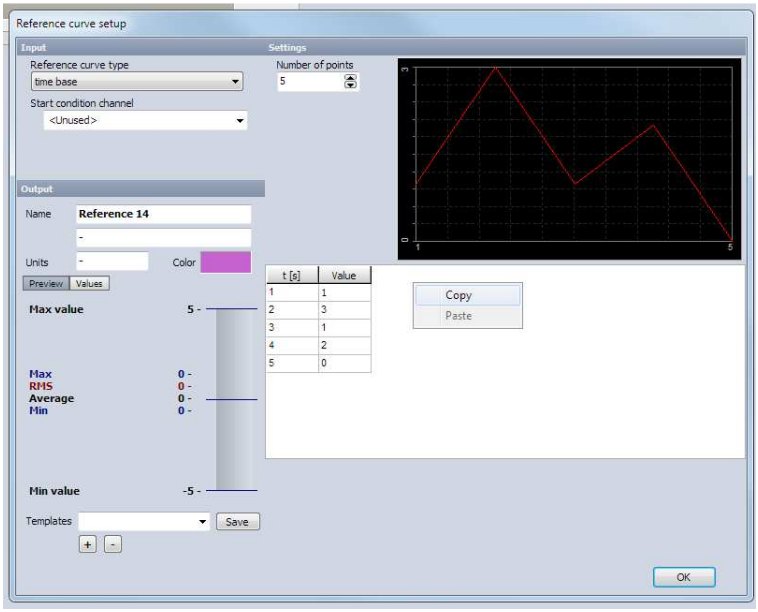
Correlation is used to calculate autocorrelation function, cross-correlation function and also the cepstrum of a signal. As the calculation is done in frequency domain, also FFT parameters must be setup.



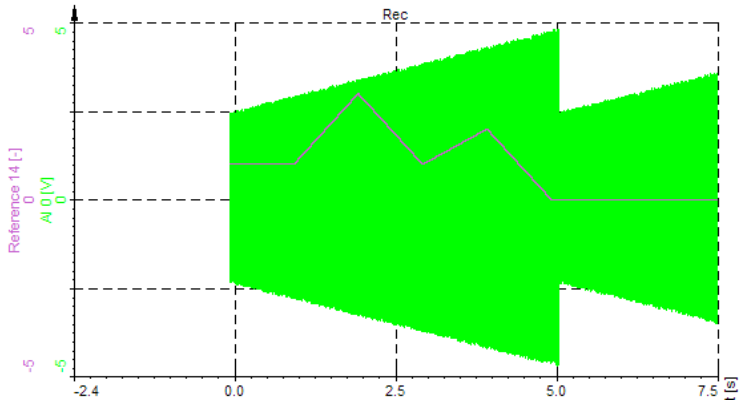
Reference curves

Reference curve

This curve can simply be setup by entering values into the table shown, the time in seconds into the first column, the corresponding value into the second.



The resulting data can then be used for calculation or display (see screenshot below).

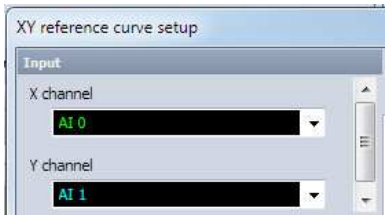


Hint: These values can also be entered by copy and paste, but the header row (first row) must be identical to the original version. Right click onto the title row or anywhere in the white field and copy the value table, then paste it into Excel, for example, add or edit the values, copy to clipboard and now paste into the reference curve table (be sure to leave the header row untouched and also to include it in the selection to be copied).

t [s]	Value
1	1
2	3
3	1
4	2
5	0

XY reference curve

For this curve x and corresponding Y values are entered into the table (or pasted of course), but additionally X and Y channels must be defined to compare the reference curve with. As output we get value 0, depending if the reference limit is exceeded by the input channel or not.



Hint:

There is another advantage of the reference curve. The output channel gives the value of 1 when the xy curve crosses the reference curve. This can be used for trigger criteria or for counting of events (with ECNT function in formula editor).

FFT reference curve

This is very similar to the above function, but here the frequency value and the corresponding amplitude are entered for reference. Again, the output delivers 0 and 1, depending on the channel value exceeding the reference limit or not. Additionally FFT parameters must be entered, as the calculation is done in frequency domain.

Constant

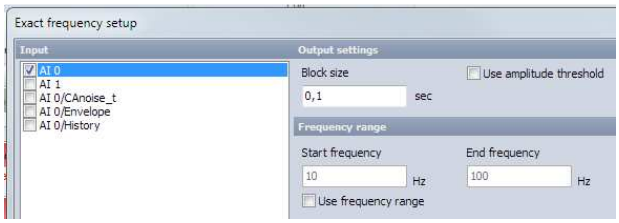
Also here a value-table is entered, which can be used for calculation or display.

FREQ Other

Exact frequency

As the name suggests, this function is used to calculate the exact frequency of a signal. Even with lower sampling rates which doesn't allow methods like period measurement and even with signals with not that clear level crossing this method will work very nicely.

The method is based on finding best fit to the theoretical sine wave. It can measure millihertz accurate with sampling rates only few kHz and a low block size. The method works best on the signals close to the pure sine wave. It will search and lock to the highest harmonic component in the signal.



CA noise

CA noise is a special calculation which can judge the noise resulting from the engine, based on the measurement of the cylinder pressure.

Envelope

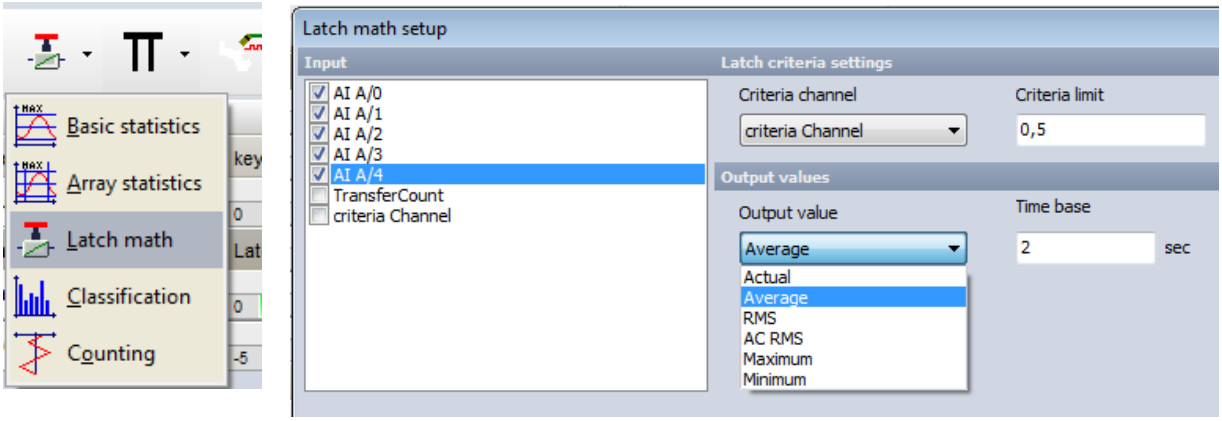
Envelope detection is a procedure for early detecting of faults on ball bearings (but of course any signal can be input for envelope detection for different use).

Delay channel

With this function, a signal channel can be delayed by a specified number of samples or time in milliseconds.

Triggering

Latch math



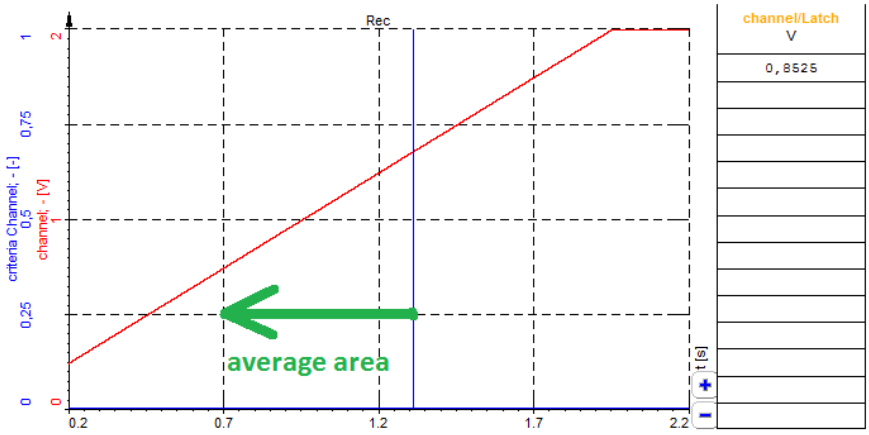
The latch function can hold a value if a trigger condition is met.
So if the criteria channel exceeds the set criteria limit the input channels will be latched.
Here the latch can be done in various modes.

Actual: If criteria is met the ACTUAL value at that time is taken.
Average: If criteria is met the AVERAGE value is taken. The calc. time can be set (Time Base) in sec.
RMS: If criteria is met the RMS value is taken. The calc time can be set (Time Base) in sec.
AC RMS: If criteria is met the AC-RMS value is taken. The calc. time can be set (Time Base) in sec.

AC-RMS will calculate the RMS only on the AC part of the signal. DC components are not

taken in account.

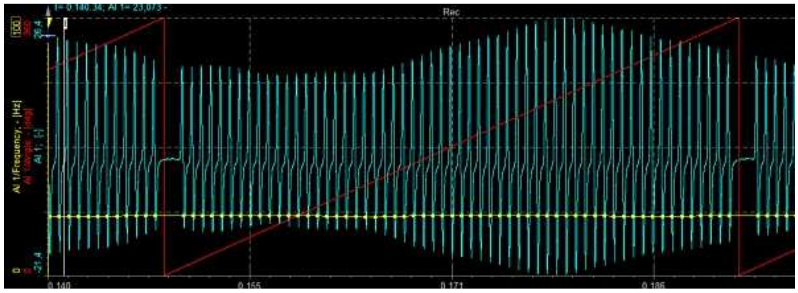
Maximum: If criteria is met the Maximum value is taken. The average time can be set (Time Base) in sec.
Minimum: If criteria is met the Minimum value is taken. The average time can be set (Time Base) in sec.



Info: The average is taken always the average time span before the event.

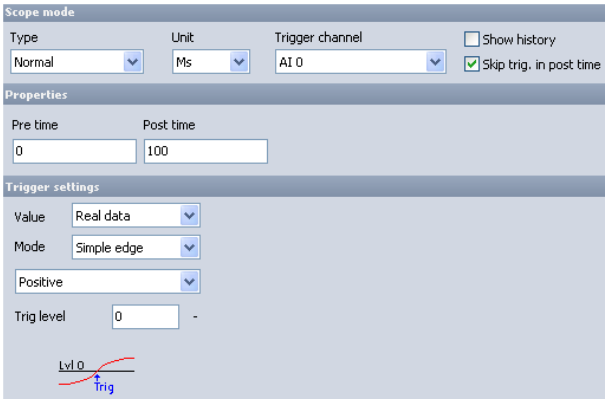
Angle sensor

The input for the angle sensor math can be any analog trigger signal (like tacho - 1 time per revolution or automotive 60-2 sensor). In both cases we need some mathematic to calculate the current angle and the frequency of incoming signal. In both cases the signal needs to be connected to analog input of the instrument. Then we need to select this signal, select the sensor type from the list of available sensor or by defining a new sensor type in the counter sensor form. Next we can simply press “Find” function to determine the trigger levels. In most cases this function works already, but in some cases we need to define the trigger levels manually by settings the “Trigger level” and “Retrigger level” input field. Next we can define the Retrigger time. This is a hold off in which the signal is not checked for triggers. This is useful to prevent trigger glitches.
The graph below shows a 60-2 signal and correctly detected angle and trigger.



Scope trigger

Scope math is intended to extract matrix channel from the scope trigger shots on which the math functions can be performed. We will get either a last shot or the history of scope triggered shots.



For that we need to define the trigger Type which has the same meaning as on the scope instrument:

normal - only output a scope shot when a real trigger occurs,
auto - if trigger is not found, output current values;
free run - outputs scope pictures regardless of the trigger;
single shot - outputs only at first trigger

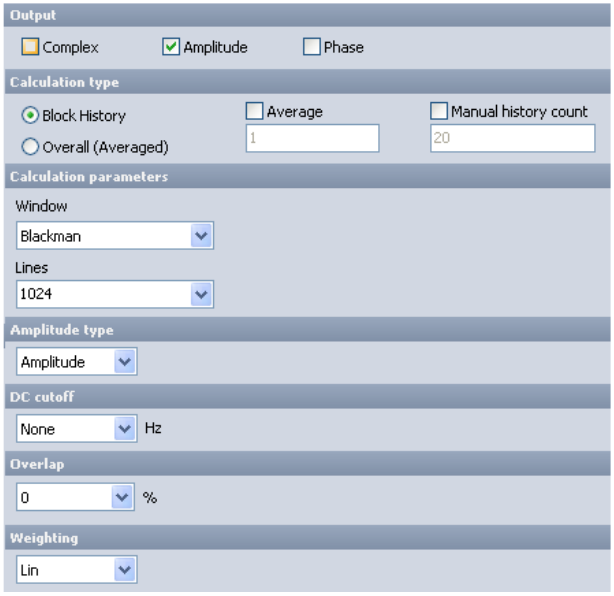
Unit defines the x scale units either in miliseconds or samples.

Trigger channel defines the channel for triggering the shots. We can output many different channels (defined on the left upper side, but trigger channel can be only one for all).
Show history defines if the output is single channel (always overwritten with latest data) or if the entire history is put into the channels.
Skip trig. in post time defines if the triggers are searched also when post time is running. If yes, data could overlap.
Then we define pre and post time in the chosen units to cut out a time window from the original signal.
Trigger settings are the same as in normal scope, alarms or storing triggers.

Spectral analysis

FFT – Fast Fourier Transformation

When you press the Setup button on FFT analysis math item, the following setup window will open:



Output of the FFT analysis could be complex (real, imaginary), amplitude or phase or any combination of that.
Calculation type can be overall (averaged), where the result is one spectrum for entire record. Second option is block history, where the FFTs are acquired shot by shot and put into the buffer. In this case they can be observed on 3D graph. Number of averages can be also set. When we enter a value of n, then it will calculate average of n spectrum and put result in the channel.
Manual history count can override the normal settings for the number of shots to be kept in the memory during measurement.
Several calculation parameters can be also set. One of them is window type, which describes the FFT window. There is a good description of usage of different window functions in the tutorial. By default we use Blackman, because it is a good compromise between amplitude error and width of side bands.

Number of lines defines the size of the FFT. More lines we choose, more accurate the frequency will be, but also longer time is needed for calculation. Number of samples needed is always twice of the size of resulting FFT (number which is set).

Important note: ANY number of lines can be entered, not only 2^n. We can manually enter 2000 or 10000 FFT lines. The only drawback is that the calculation will get slower if we choose strange numbers (like 12431).

Amplitude type defines what kind of amplitude is put in the channel. We have several options:

Amplitude type	**Units**	**Description**
Amplitude (Auto)	V	is the pure signal amplitude
RMS	V rms	is the RMS amplitude, calculated as Amplitude/sqrt(2)
Power	V * V	calculated as RMS value squared
PSD	V * V / Hz	calculated as RMS squared, divided by the line resolution and sqrt(2) - used for checking the noise
RMS SD	V / sqrt(Hz)	calculated as RMS value, divided by the square root of line resolution - also used for checking the noise

DC cutoff option will remove the static offset from the FFT. If it is set to none, it will output FFT as calculated, if not, it will cut the number of lines until frequency which is defined is reached.
Overlap defines how much two FFT shots will overlap between each other. 50% is enough that all the samples will have same weight on the result independant of the window which is used.
Weighting defines which sound weighting should be used on the resulting FFT.

Hint:
As all matrix channels also FFT can be best seen on 2D graph. Block based FFT can be also put in the 3D graph for viewing the history.

STFT – Short Time Fourier Transformation

Short time Fourier transformation is the procedure which calculates more lines than the normal FFT. This is achieved by having smaller real block size of data and larger FFT size. Real data is windowed and zero padded and then FFT is calculated. With this procedure we can calculate more FFTs for the same time base. It is nice to be used with fast transients.

CPB – Constant Percentage Bandwidth

A band-pass filter,whose bandwidth is a constant percentage of center frequency. 1/3 octave filters, including those synthesized in DSAs, are constant percentage bandwidth.
Here we can select two options:

Synthesized

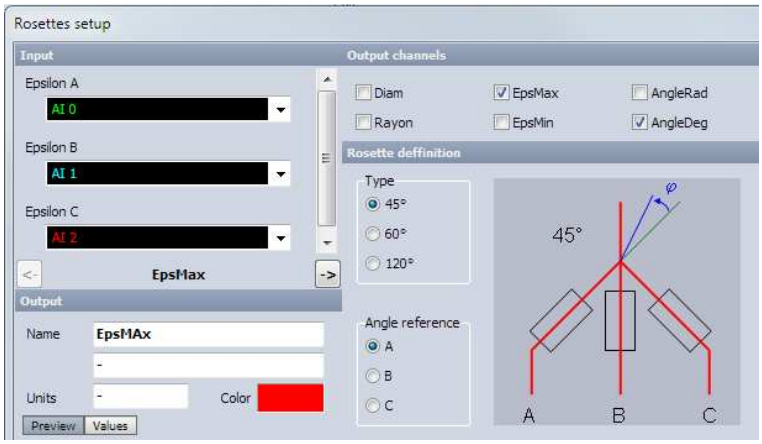
Here the filters are calculated in frequency domain, which gives very exact filters, but at the same time poor update rates, due to windowing and the big calculation cost behind.

True Octave (ANSI, IEC)

This is a time-domain multiband filter, as is demanded by the standards, which is very fast, but not as exact as the above frequency domain filter.

Rosettes

This functionality is used for strain gauges in special geometric arrangement.

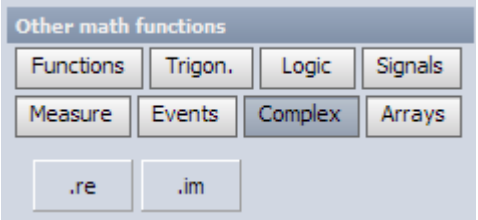


The output channels represent the following measurement data:

Diam: average stress
Rayon: max shear stress
EpsMax: max stress (angle direction)
EpsMin: min stress (angle + 90° direction)
Angle: angle of max stress

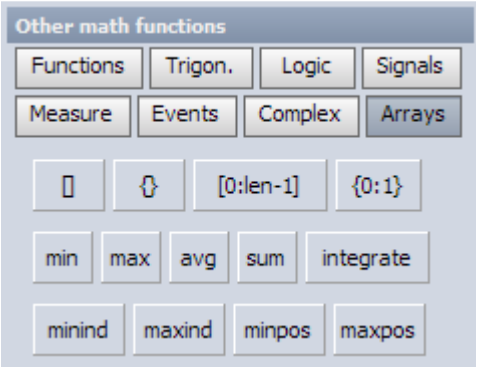
DEWESoft 7.1 – new functions

Complex



Value.re …real part of complex data set
Value.im … imaginary part of complex data set

Arrays



[]:'Data'[Idx] returns one value from array channel Data at index position Idx
{}: 'Data'[Pos] returns one value from array channel Data at position Pos in axis units
[0:1]: 'Data'[N:M] returns a cut-out array of array channel Data, from index position N to index position M, where 0 is the first value and len-1 is the last possible value.
{N:M}: 'Data'[N:M] returns a cut-out array of array channel Data, from position N to position M, according to axis units.
min: min('Data') returns minimum value of array Data
max: max('Data') returns maximum value of array Data
avg: avg('Data') returns average value of array Data
sum: sum('Data') returns the sum of all values of array Data
integrate: integrate('Data') returns integrated array of array Data
minind: minind('Data') returns index of minimum value of array Data
maxind: maxind('Data') returns index of maximum value of array Data
minpos: minpos('Data') return the position in axis units of minimum value of array Data
maxpos: maxpos('Data') return the position in axis units of maximum value of array Data

To cut out an array from the first value to index 100 or to cut out from index 150 to the end use the following syntax:

'Data'[:100]
'Data'[150:]

Matrices – two-dimensional array

The same operators as for arrays can be used, but must be used twice here, one time for each dimension. To pick the whole dimension another operator is introduced, see example:

To pick the 10th line of a matrix use the following command:
Line X10: **'Matrix'[10][:]**

To choose an area from a matrix and calculate the average value:
Area: **'Matrix'[10:20][40:60]**
Average: **avg('Area')**

Also here shorthand notation can be used. To pick an area from index zero to 100 in the first dimension and from index 200 to end in the second use the following command:

'Matrix'[:100][200:]

MATH – EXAMPLES

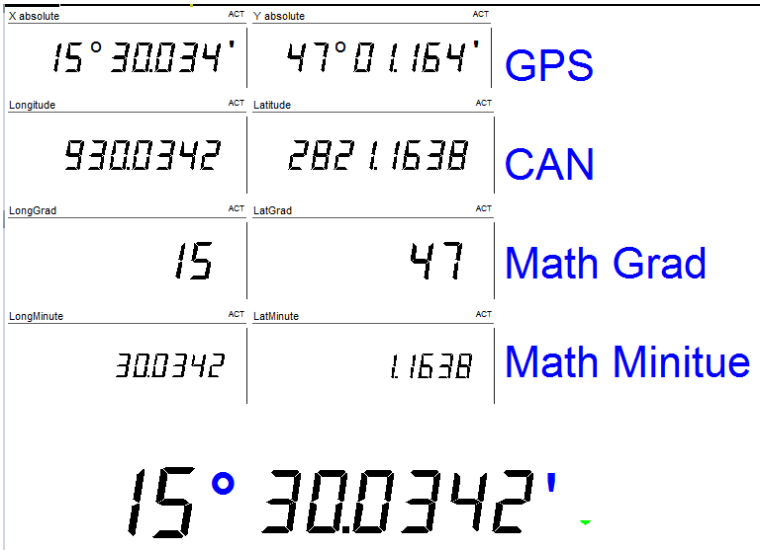
Formula

Function: MOD / TRUNC – Example: Separate CAN GPS Signal to DEG:MIN,xxx

TRUNC: converts a number into integer

Trunc(1452.457) = 1452

Example: The VGPS Longitude and Latitude signals received over the CAN Bus should be separated into DEG:MIN,xxx. The CAN data is received in MIN like it is shown below. With math functions the regular GPS display should be produced. Below the result is shown.



MATH: trunc('Latitude'/60) dividing by 60 will convert it from minutes to deg, where the trunc function will eliminate the digits after the comma. → result [°]

(('Latitude'*100000) mod 6000000) / 100000 will provide the minutes and the remainder.

Because the MOD function only delivers the remainder as an integer, we have to multiply the latitude with 100000 and use MOD with 60 *100 000 = 6000 000, then divide the result again with 100,000 to get the result.

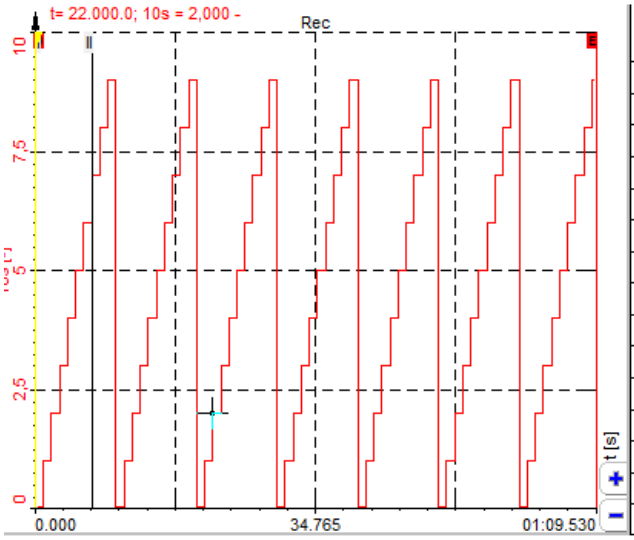
Function: MOD / TIME – Example: Show actual value averaged every 10s in a list and export it to Excel.

TIME: is providing the elapsed time of the measurement.

MATH: **(**Time MOD 60)

This will create a saw tooth with a period time of 60s. Look to the picture below. This is channel will be used as event channel in the LATCH math to average the actual channel and show it in a list. The

averaged values could be exported to Excel or TXT asynchronously. To export it asynchronously @ Export only the asynchrony channels have to be selected.



Time	pressrue/Latch bar
06.09.2010 18:00:22.537	0,9997
06.09.2010 18:00:12.537	0,9997
06.09.2010 18:00:02.537	0,9998
06.09.2010 17:59:52.537	0,9998
06.09.2010 17:59:42.537	0,9998
06.09.2010 17:59:32.537	0,9998
06.09.2010 17:59:22.537	0,9998

Exported	Index	Type	Acq. rate	Dimension	Name
No	0	AI 0	10000	Scalar	pressrue
No	1	Math 0 (Formula)	single	Scalar	Frm0/Formula 0
No	2	Math 1 (Formula)	10000	Scalar	Frm1/10s
Yes	3	Math 2 (Latch math)	0,1	Scalar	Latch0/Latch index
Yes	4	Math 2 (Latch math)	0,1	Scalar	Latch0/pressrue/Latch

Function: SCNT Sample Counter - Example: Create Angle Signal

SCNT: delivers the samples acquired from start of the measurement.
The counter will be reset at the start of storing.

MOD: delivers the remainder of a division
420 MOD 720 = 420
740 MOD 720 = 20

Example: If external clocking is used and the signal will be shown in an angle-based XY diagram, with the MOD function we can create an angle signal.

Let's assume we are using an encoder with 720 pulses/rev.

MATH: SCNT MOD 720 will deliver a sawtooth which runs from 0 to 720.
To get the angle we have to multiply it by 0.5 deg, so at the end we get this formula:
SCNT MOD 720 *0.5 this channel can be used in a XY diagram to show the result angle based.

The only disadvantage will be, if we get wrong pulses from the encoder (i.e., noise or spikes), the angle signal will shift.

If a TRG pulse is also available, CLK and TRG, this signals could be also routed to an Orion Counter, where this is eliminated. Because the TRG PULSE will reset the counter every revolution. So a hardware counter on an ORION A/D card will typically yield better results than software functions.

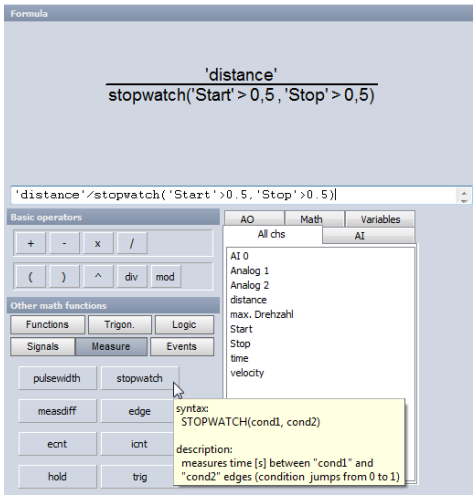
Function: Stopwatch – Example: Velocity with two light barriers

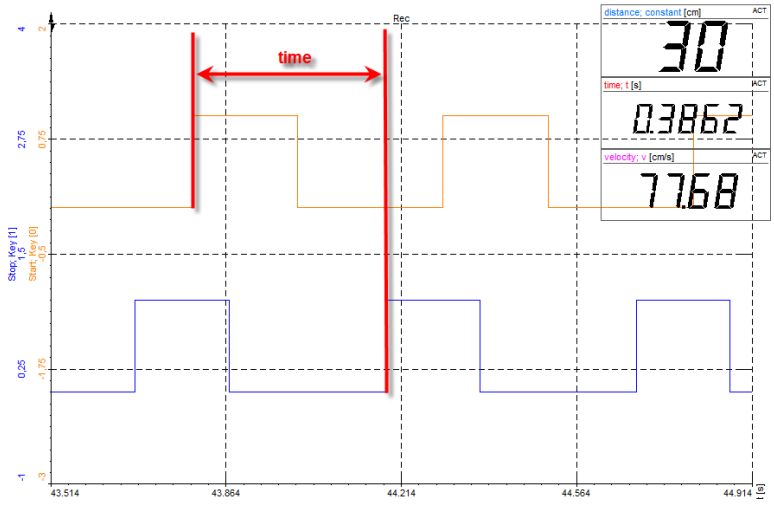
STOPWATCH: **STOPWATCH(Condition1,Condition2)** measures the time in [s] between condition1 and condition2. Condition is a logic value that jumps from 0 to 1 (edge sensitive)

Example: If the distance between the two light barriers is known as 'Distance' the average speed of an object passing the light barriers could be calculated.

MATH: **'Distance' / stopwatch('Start'>0.5,'Stop'>0.5)**

Start and Stop could be analog or digital signal. The logic comparison '>' turns the condition into a logic value anyway. Instead of 0.5 there could be used any logical triggerlevel. Also for digital signals the comparison is recommended.



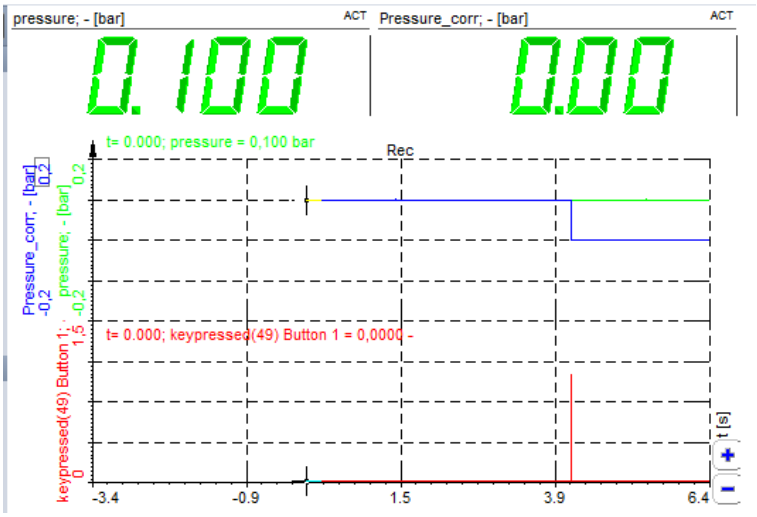


Function: HOLD – Example: Remove offset from a Channel

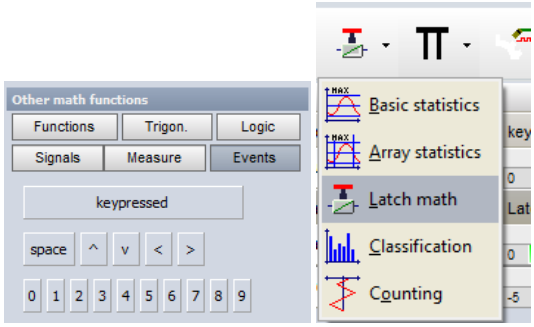
HOLD: HOLD(channel, condition) Will latch the actual value of the channel if condition become true.

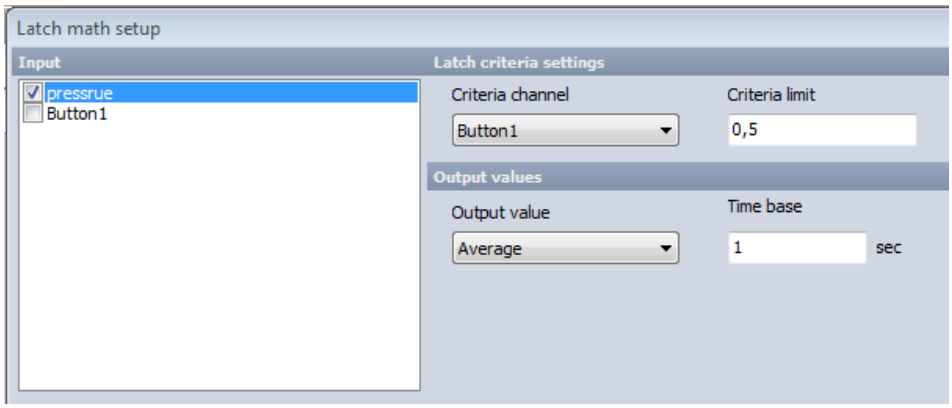
MATH: 'pressure'-hold('pressure',keypressed(49)>0.5)
The actual pressure channel is subtracted with the value latched in the hold function. The hold function will latch the actual pressure if the second statement (keypressed(49)>0.5) become true.

So even during the measurement a offset compensation could be done by pressing a specific key. The picture below is showing an example. The keypressed(49) channel is indicating the pressed key in an additional math channel to make it more abvious.

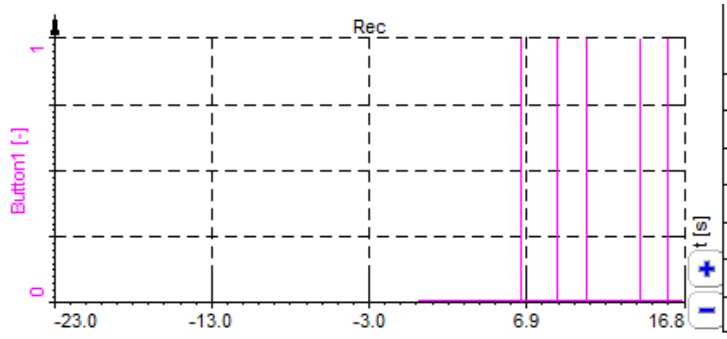


Function: KEYPRESSED / LATCH – Example: Latch Value into List





MATH: keypressed(49) produces signal from 0 to 1 with a duration of 1 sample if [1] Button is pressed. This could be used in the latch Math to latch the actual or average value of another channel(s) in a list.

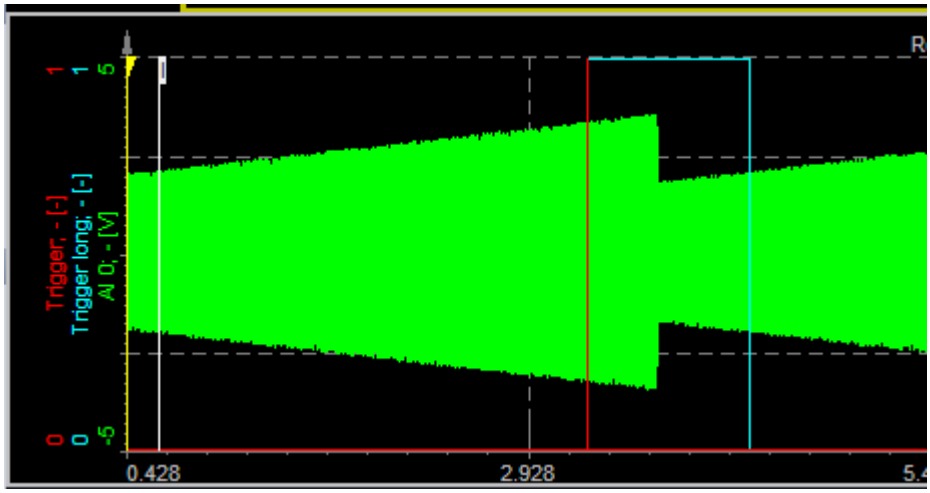


Time	pressrue/Latch bar
06.09.2010 18:42:44.999	0,9998
06.09.2010 18:42:43.234	0,9998
06.09.2010 18:42:39.904	0,9998
06.09.2010 18:42:38.040	0,9998
06.09.2010 18:42:35.642	0,9998

Trigger event is high for specified duration

If we have a trigger pulse of any kind with a short duration, we can create a signal out of it that is high for a specified duration (here 1s) with the following formula:

edge('Trigger', time – hold(time, 'Trigger') >= 1)



Decode logical signal for positive and negative sign in mathematics

A logical signal (0 or 1) is used to code the positive or negative sign of a measurement channel.

SigSignLogical = 0	…	negative number
SigSignLogical = 1	…	positive number

To recreate the signal together with the correct sign use the following formula:

SigSign = 2 * SigSignLogical – 1

SigSignLogical = 0	…	SigSign = -1
SigSignLogical = 1	…	SigSign = +1

This new signal can now be used as a factor for calculation to get the correct signed channel values.

Duty cycle calculation – slow signals (pulsewidth, stopwatch function)

If the analog signal cycle duration is very long, the above calculation will not work anymore. So here is another possibility to calculate duty cycle percentage.

First we define a trigger value for the start of a period and create a pulse out of it.

edge('AI 0' > 12) (trig cycle)

We also need a trigger for the start of the negative half-cycle.

edge('AI 0' < 12) (trig negative half-wave)

Next we measure the time of the last whole cycle (between start of period triggers).

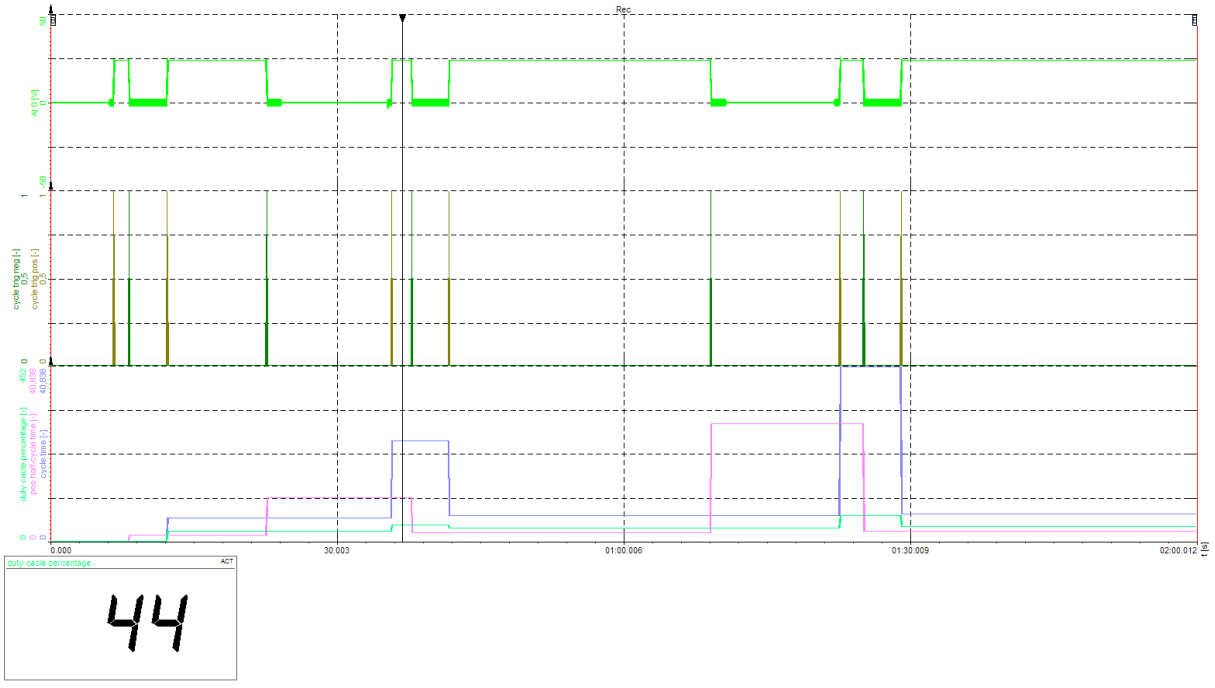
pulsewidth('trig cycle') (time whole period)

Another time is measured, here it is from start of period to beginning of negative half-wave, which gives the time of the positive half-wave..

stopwatch('trig cycle', 'trig negative half-wave') (time positive half-wave)

Finally, the ratio between the positive half-cycle and the whole cycle is calculated and multiplied by 100 to get percentage, also the hold function is used for the value to only be updated after each whole period.

hold('time positive half-wave' / 'time whole period' * 100, 'trig cycle')



Duty cycle signal generator (mod, scnt)

For testing lots of times we need a square signal with a duty cycle, which can be setup.
We begin by entering parameters (of course this can also be assigned to visual controls):
frequency, duty cycle percentage of the first half-period, amplitude and a switch (0 or 1) to define signal to ground or peak-peak amplitude.
The first calculation gives samples per 1 rotation.

sr / 'input_freq' (samples per 1 rotation)

Now we need an angle signal, which is reset after one full period.

scnt mod 'samples per 1 rotation' (angle)

Next we calculate the angle value limit, where the negative half-cycle starts out of given duty cycle percentage.

'input_first duty cycle percentage' / 100 * 'samples per 1 rotation' (limit)

We then create two output signals, first the one to ground.

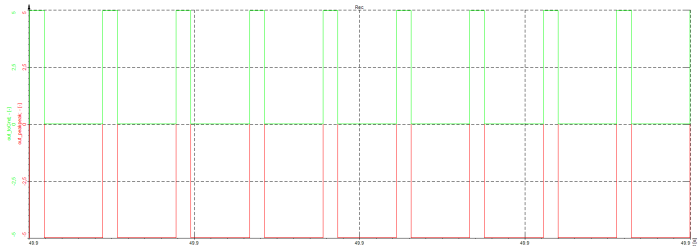
if('angle' < 'limit', 'input_amplitude', 0) (out_toGnd)

And the peak-peak signal.

if('angle'<'limit', 'input_amplitude', -'input_amplitude') (out_peakpeak)

Finally we check the input parameter and select one of these as output.

if ('input_IsPeakPeak' = 1, 'out_peakpeak', 'out_toGnd')



Stopwatch triggered by event (keypressed)

Create a stopwatch, which is started and stopped via keypress during measurement.

We need the keypressed event for this, but together with the edge function, to suppress retriggering of start event and to hold the value until the stop button is pressed.
For the start stopwatch key:

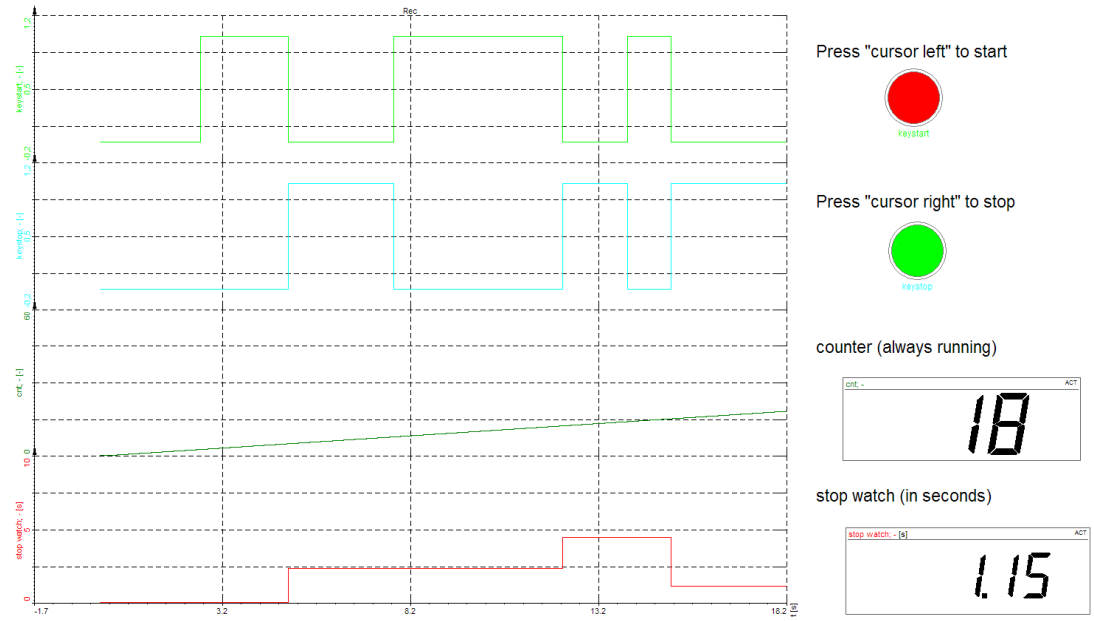
edge(keypressed(37), keypressed(39)) (keystart)

And of course also for the stop key:

edge(keypressed(39), keypressed(37)) (keystop)

The math function “stopwatch” can measure time between two different channel edges and this is exactly what is needed here.

stopwatch('keystart', 'keystop')



Count pulses that are longer than a specified duration

To only count the pulses, which are longer than 30ms we create a digital signal with a threshold of about half the peak signal, to get a nice edge.

'pressure'>=500 (pressure_digital)

We also want to specify a limit for the pulsewidth, and only if this limit is exceeded a logical signal is high.

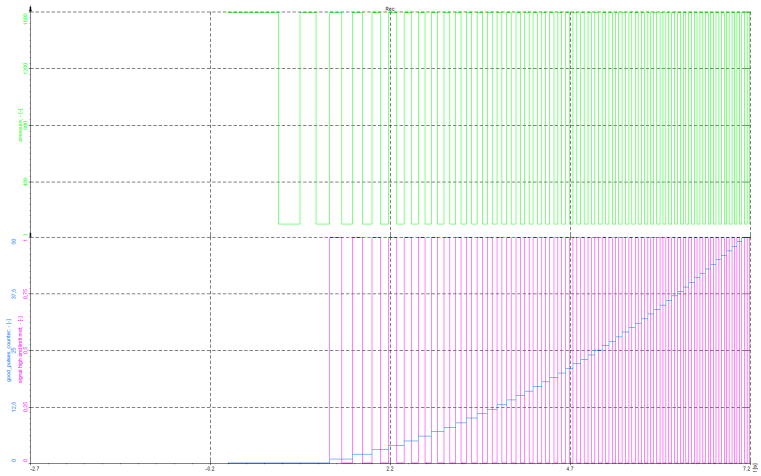
pulsewidth('pressure_digital')>=0.03 (pulsewidth_ok)

What we actually need is the signal to be high and the limit condition should be met.

'pressure_digital' AND 'pulsewidth_ok' (signal high and limit met)

All that is left to do now is to count the “good” pulses.

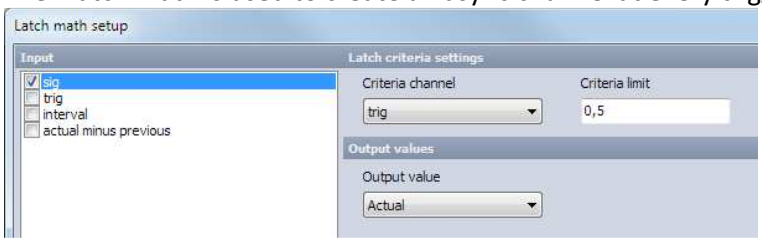
ecnt('signal high and limit met')



Measurement(t_n)- Measurement(t_{n-1}) – prev function

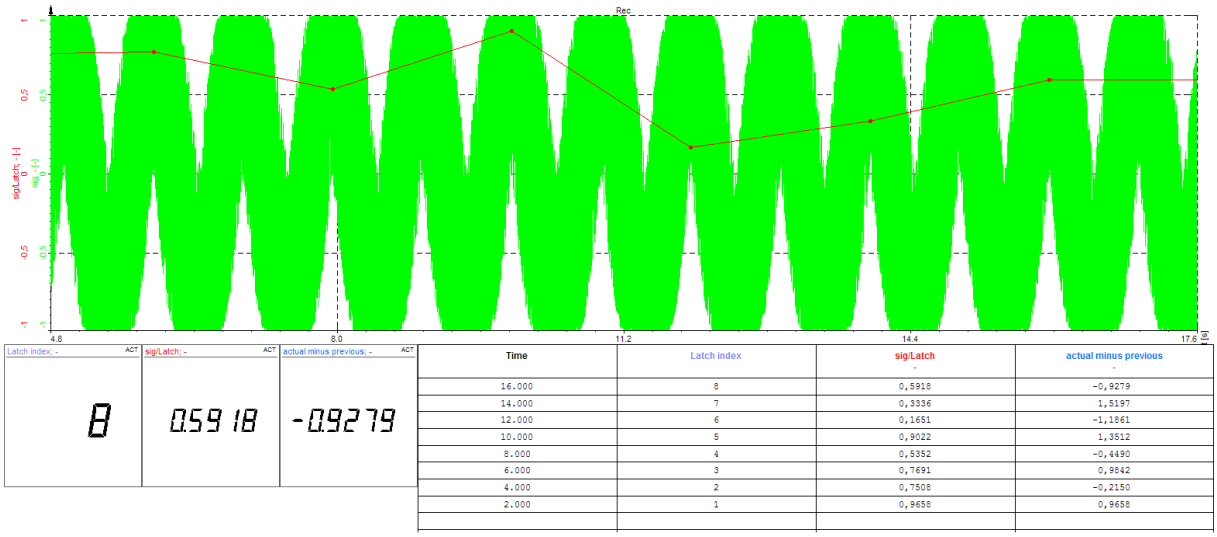
Here we take an analog test signal and we create a trigger every 2 seconds:
time mod 2 = 0 (trig)

Then latch math is used to create an async channel at every trigger position.



To calculate the actual value minus the previous (async) value, use the following formula
'sig/Latch' – prev

Notice that we choose async timebase and the reference channel is the latched signal now (see screenshot below).



Time	Latch index	sig/Latch	actual minus previous
		-	-
16.000	8	0,5918	-0,9279
14.000	7	0,3336	1,5197
12.000	6	0,1651	-1,1861
10.000	5	0,9022	1,3512
8.000	4	0,5352	-0,4490
6.000	3	0,7691	0,9842
4.000	2	0,7508	-0,2150
2.000	1	0,9658	0,9658

Logical event counter with two reset conditions – logical, keypressed, storing function

As test signal we use

(trian(1))/2 (Rect)

The logical condition is

'Rect' > 0 (Condition)

This is the event counter how often the logical condition is met

ecnt('Rect' > 0) (Zero crossing count)

Also samples are counted while condition is met

icnt('Rect' > 0) (Samples above zero)

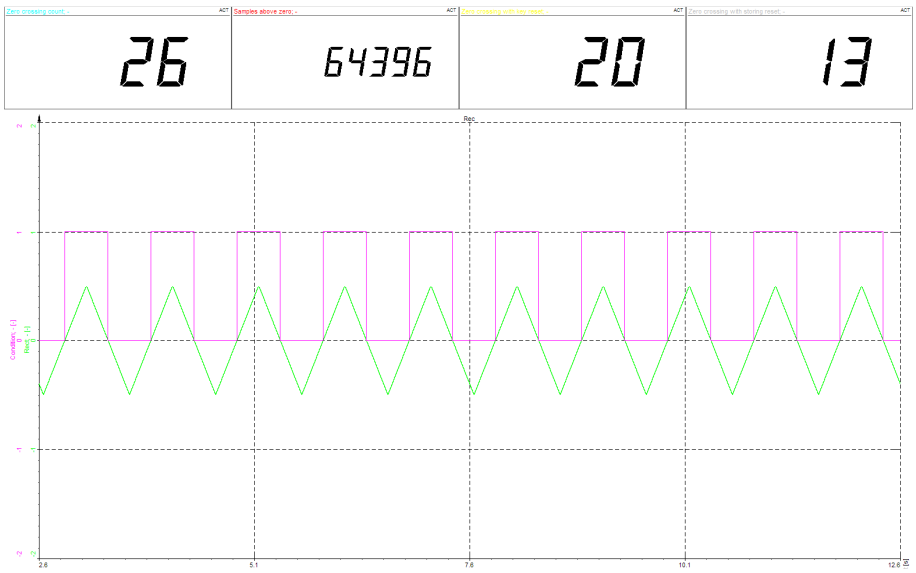
Zero crossing with key reset:

'Zero crossing count' - hold('Zero crossing count', keypressed(32) > 0.5)

Zero crossing with storing reset:

'Zero crossing count' - hold('Zero crossing count', storing > 0.5)

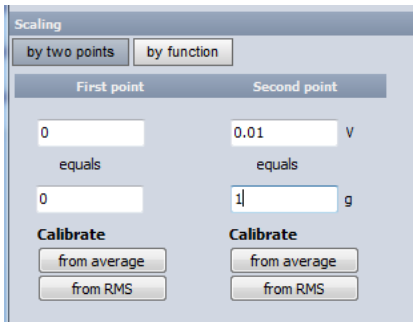
Note: "storing" is 1 as long as data is being stored, 0 otherwise
"trig" gives just one pulse at the beginning of storing



Filters

Function: Acceleration → velocity → displacement

Very often, acceleration must be converted to velocity or displacement. This is done with integration or double integration. But to do it right, we need to start at the beginning: the scaling.

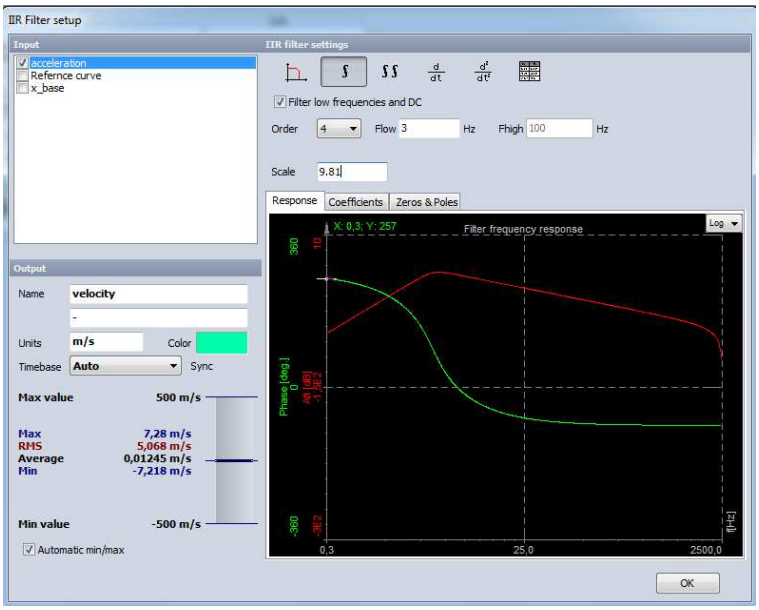


1.) So the scaling is done either in g or m/s² in the analog setup of DEWESoft, where the measurement chain starts.

Velocity

2.) We have to apply the acceleration channel to a single integration to get the velocity. This will be found in the IIR filters MATH function.

If an integration is done on low frequencies, or signals with a static offset, this will produce incorrect results. Therefore the integration function has an the option to filter low frequencies and zero out DC components (offsets). In this case, a 3 Hz Fourth order high pass filter is used which gives reliable results on most of the applications.



The unit should be m/s, and because the acceleration is scaled in g we have to add an additional scaling factor 9.81.

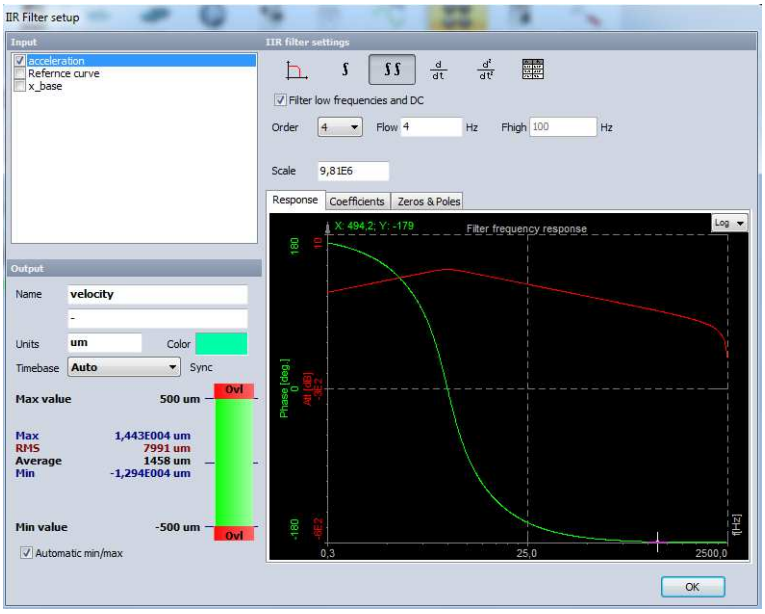
If the acceleration would have been scaled in m/s² in the analog setup the scaling factor is not needed.

In case mm/s is needed, either 9810 or 1000 has to be entered as scaling factor.

Displacement

3.) To get displacement we have to us the double integration. Also there the high pass filter will be needed, to avoid wrong results caused on low frequencies and dc offset.

4Hz 4 Order will give reliable results on most case. If not the filter has to be set higher (5Hz).



Most of the time the displacement is needed in um. So the scale factor has to be set depending on the analog acceleration scaling.

In this example the input was scaled in g, take a look to the picture above, so to get um we have to enter a scale value of 9810000 or 1000 000 if the input would have scaling in m/s².

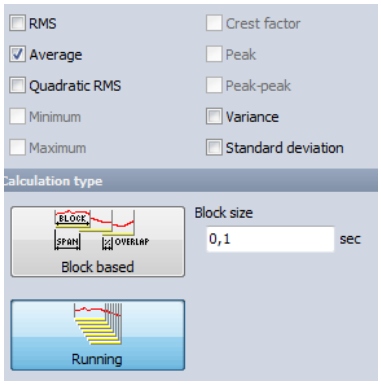
Note: As described above we have to use a high-pass filter to avoid incorrect results.

Also be aware that below the high-pass filter frequency, the output will be attenuated by the filter.
So exact readings will be only available @ frequencies higher than the high-pass filter.
This is the case for both displacement and velocity.

If displacement is needed below this frequency, it is preferable to use a displacement sensor in the first place and not use software functions.

Statistics

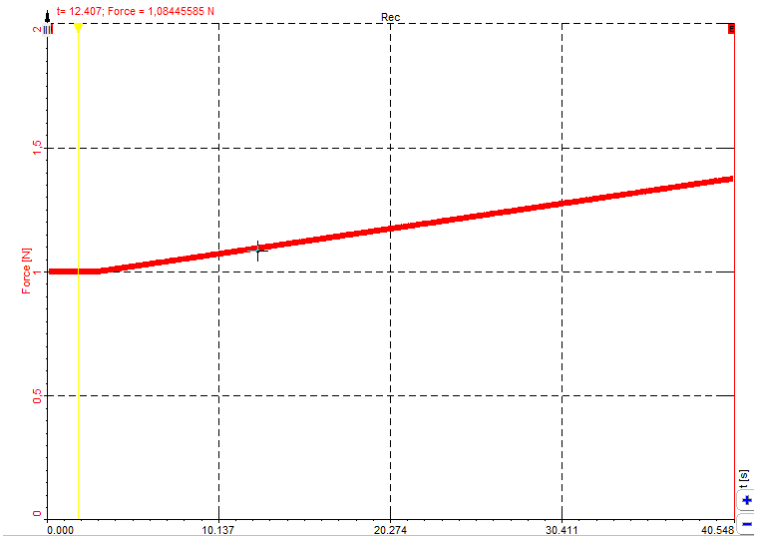
Function: Average – Example: Remove offset from channel POST Processing



Above, “running” is selected.

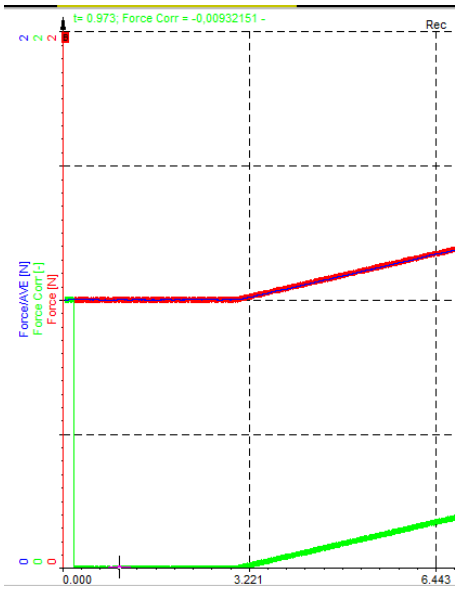
Average: Calculates the running average over a Block size of 0,1s.

Example: An analog channel with an offset should be recalculated in Math. The first 0,2 seconds of the signal should be averaged and used for the offset correction. Below the raw signal from the signal is shown.



MATH: 'Force'-hold('Force/AVE',time>0.2)

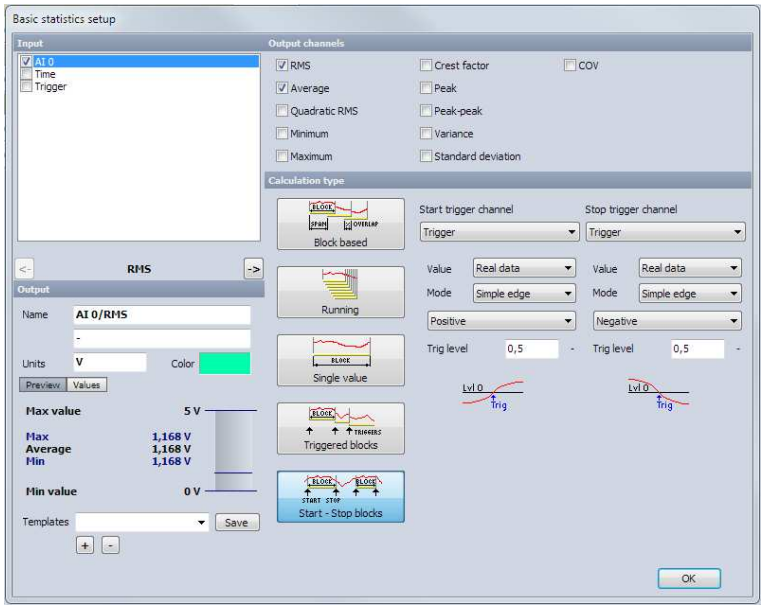
Force/AVE is the running average calculated out of the statistics. This will be latched if the time (reflecting the actual time in the dataset) is higher than 0.2 seconds, the hold function will latch the average and subtract it from the force. The result is shown in the picture below.



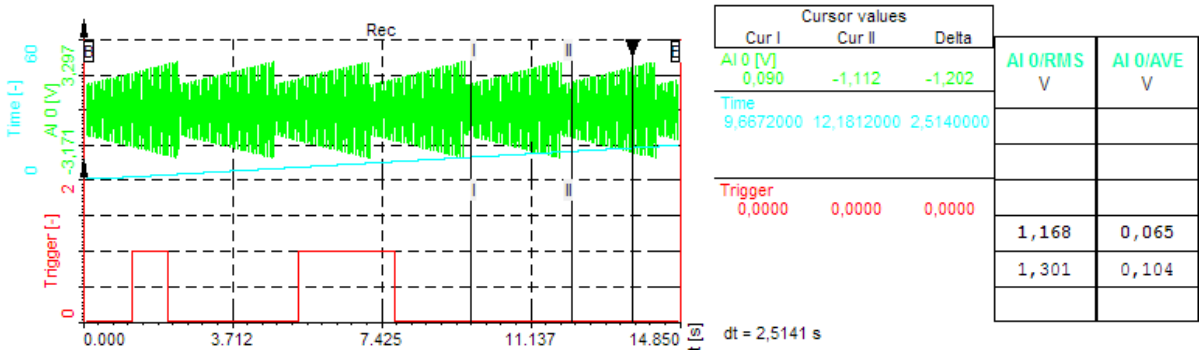
RMS and average value for specific parts of a signal chosen by cursor

Let's say we would like to cut out specific parts of a signal and calculate statistical values (RMS, average) for these cut-out parts.
It is possible to export the zoomed-in part as a new DEWESoft data file and do the statistics for the whole new file. But here is a much nicer approach:
In Math we create the following formulas and a basic statistics channel:

Time channel: **time**
Trigger channel: empty for now
Basic statistics (Start – Stop blocks): As a start trigger we select the newly created trigger channel with positive edge and 0.5 trigger level, the same with stop-trigger, except here we use negative trigger edge.



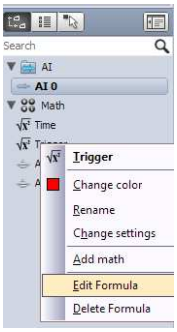
In Analysis mode we add a recorder with the signal we want to analyse and the math time channel. We enable “show cursor values” for the visual control and are npw able to select a region with cursor 1 and 2. The time in seconds at these two cursor positions is displayed now so the trigger channel can now be defined.



Trigger channel: **(time > 1.2) and (time < 2.1) or**
(time > 5.34) and (time < 7.76)

To add another trigger for the actual time range, just add the following to the trigger channel math:

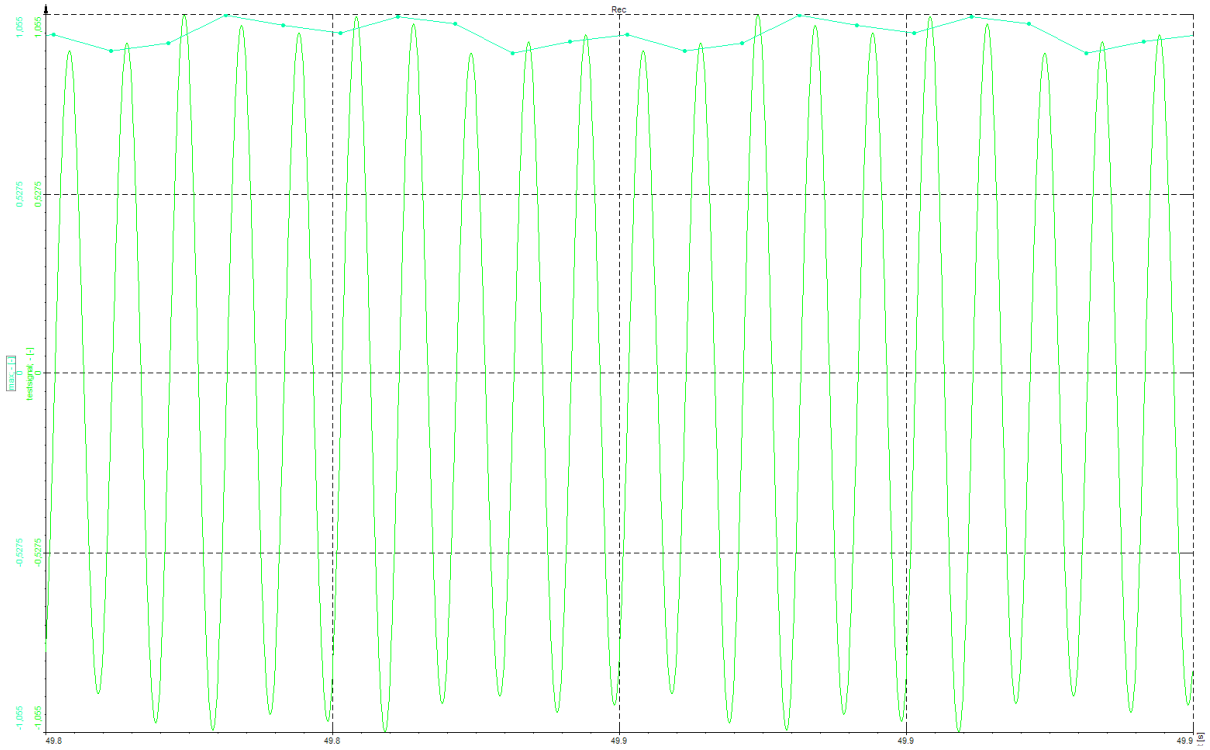
or (time > 9.66) and (time < 12.18)



For this you don’t have to switch back to setup and math, just use the context menu (right-click) of the trigger math channel in analysis mode and pick “Edit Formula” (see screenshot).

Peak hold function

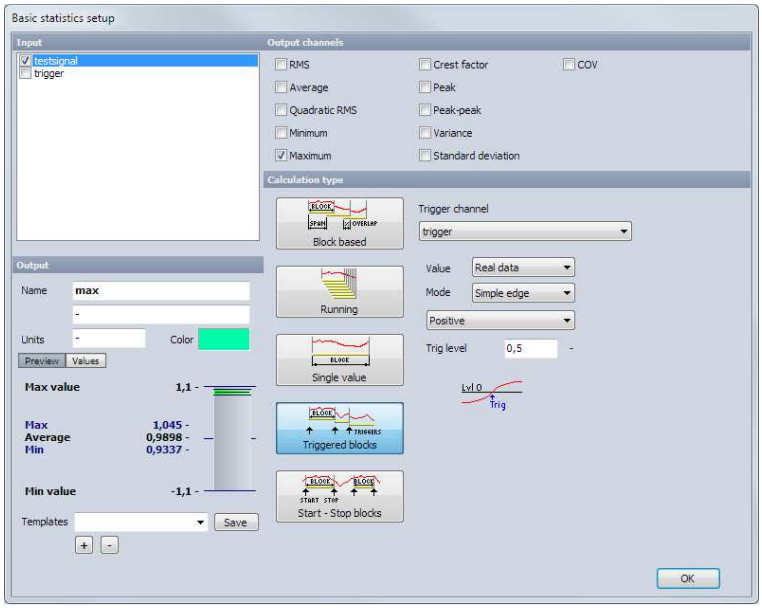
To display the peak value of the last period(see picture, notice the small delay between actual peak and peak function) the following math functions can be used.



First we need a trigger signal every signal period.

'testsignal' > 0

Then we use the **basic statistics** with the “triggered blocks” setting and the above function as trigger channel, with trigger level set to 0.5.



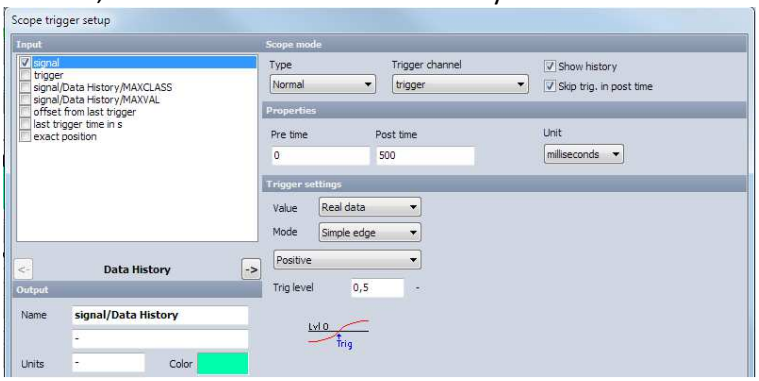
Peak hold function – at exact peak position

We have seen in the previous example, that the peak value is held after one full period of the signal. For now we would like to get a more exact result, exactly at the position of the maximum (peak).

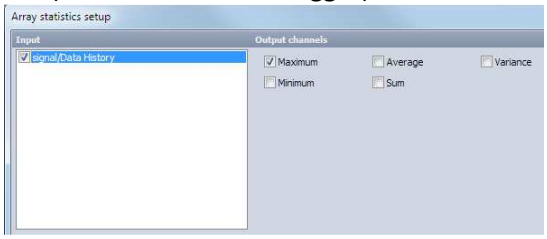
We would like to trigger every 0.5 seconds.

scnt mod (sr/2) = 0 (trigger)

Now we add the **scope trigger** function to trigger on the input signal with the above trigger as trigger channel, 0.5 as threshold level and history checked.



Array statistics delivers the maximum value of an array, but also the position, where this maximum lies (relative to the last trigger).



As we get a trigger every half second, we can calculate elapsed total time with this formula.

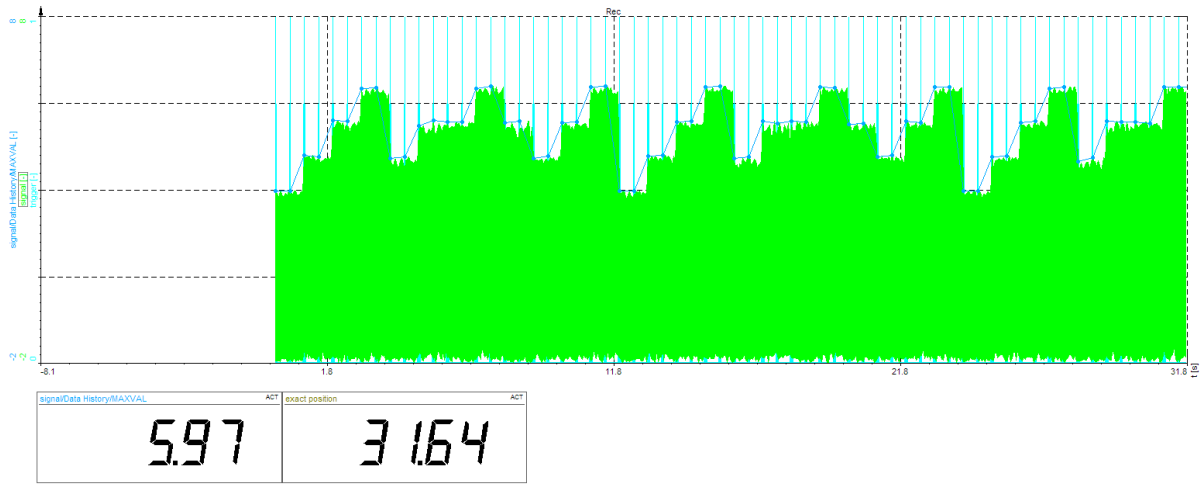
ecnt('trigger')*0.5 (elapsed time triggers)

To get the offset from last trigger point in seconds, we use the MAXCLASS result from aray statistics to get the sample based index and divide by sample rate.

'signal/Data History/MAXCLASS' / sr (time position in scope trigger)

For the exact position of the peak value, we then simply add those two times.

'elapsed time triggers' + 'time position in scope trigger'



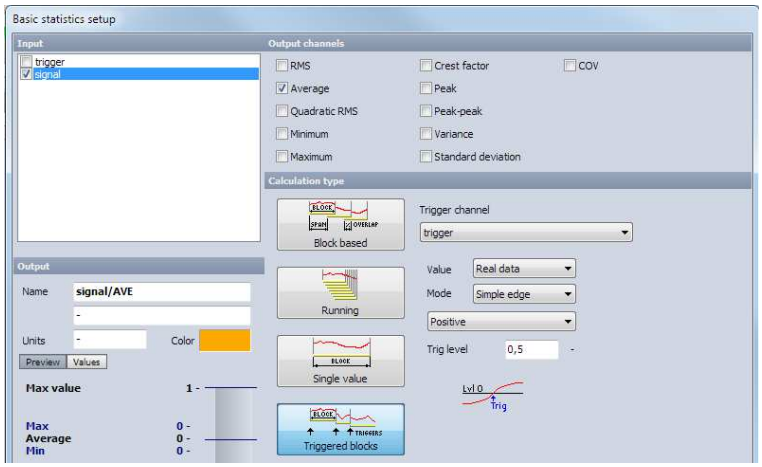
Hint: The display still shows the peak value at the end of one period, but from array statistics we get not only the peak value, but also the exact position (shown at the bottom right).

How to start Average block exactly from trigger event

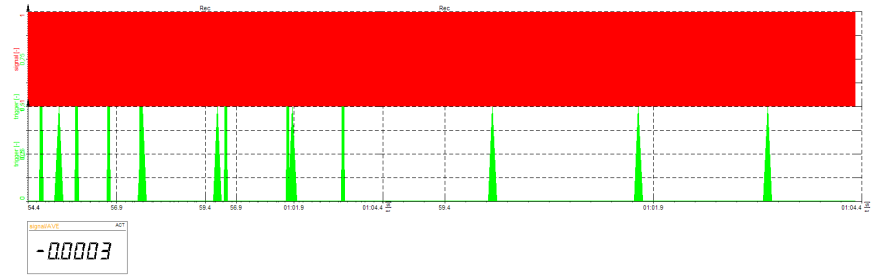
For this we choose the math function basic statistics, average with option “triggered blocks”, but first we need a trigger signal, which can be anything we can get an edge from; here we just use

edge(keypressed(32))

Now we can setup the statistics with the trigger signal, simple edge and trigger level at 0,5, as it is a logical (high-low) signal.



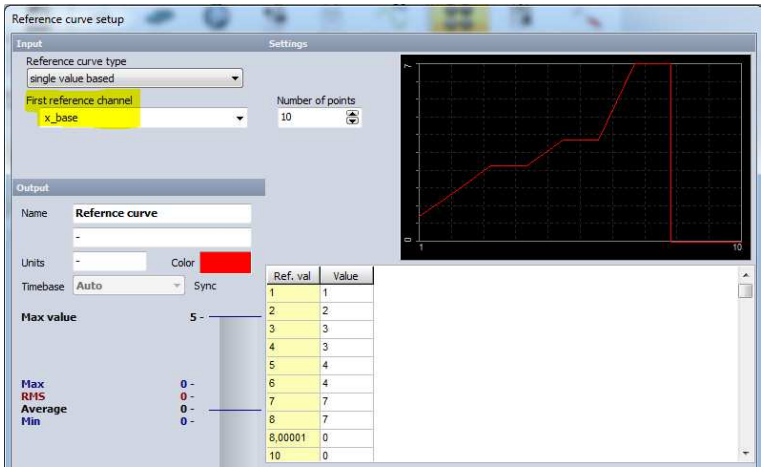
Statistics calculation is now done exactly between the last two trigger events.



Reference Curve

Function: Create a reference curve, with restart possibility.

Example: The target is to create a free definable reference curve, with the possibility to restart.
Here the reference curve function with reference channel is used.
Below you will see a reference curve with is related to a second channel called x_base.
So if the x_base channel delivers 2 the output will be 2, if time delivers 6 the output will be 4, and so on...



Now let's have a closer look at the x_base channel, which is created in a math formula.

Formula: time+100-hold(time+100,keypressed(49))

One of the target was to start and restart the reference curve on an event... which can be a keyboard event
or any other event, like pressure < 12bar.

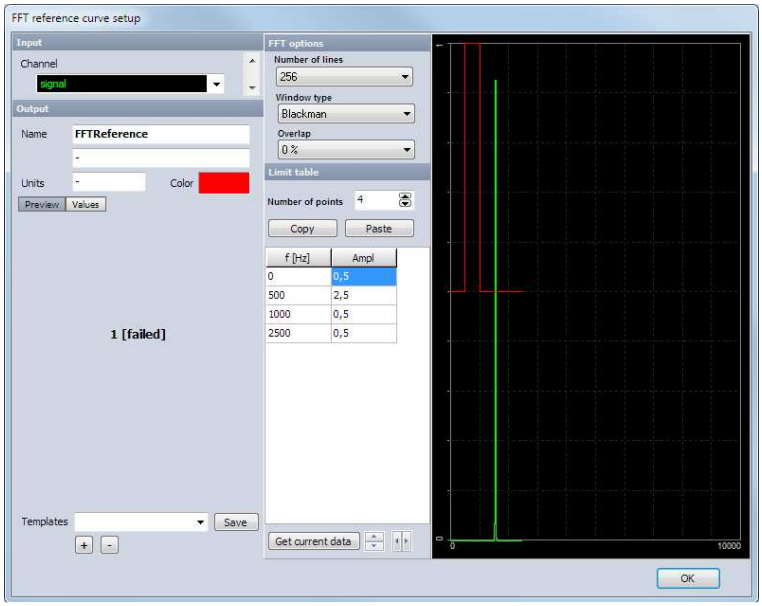
So let's have a closer look to the reference curve, after 8s it will become 0.
So at the beginning the x_base channel should start with a value higher than 8, so that the output of the reference curve is 0. That is solved by adding 100 to the time function of the formula (red part).
So at the beginning the formula will output 100 and count up ,… the reference curve will deliver 0.

The next thing would be to start the refence curve by an event. This is realized in the second part of the formula(green). If an event, (key pressed) will occur, (time+100) will be subtracted from time+100.
So the x_base channel will start from 0 and the reference curve will be output.

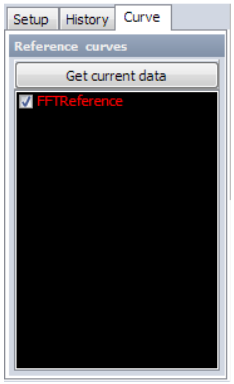
How to display FFT reference curve in FFT display

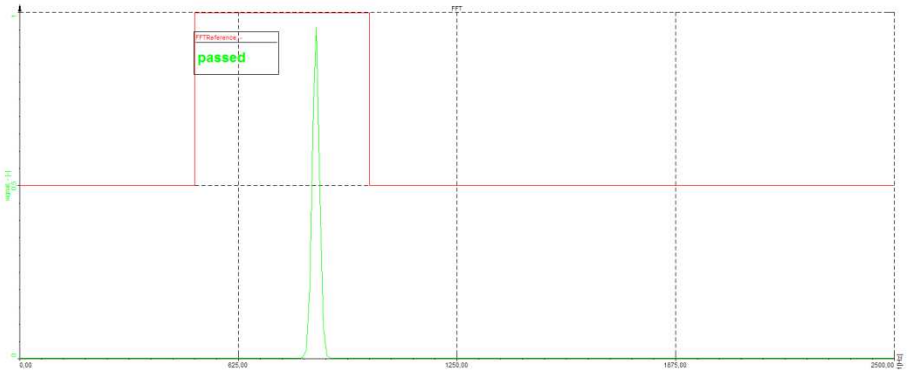
For this to work we need to create a math FFT reference curve and add it into the FFT display.

In the **FFT reference curve** setup, we define our signal channel as input. The table in the center is used to enter the reference curve, by entering the frequency in the first column and the corresponding amplitude in the second (copy and paste can also be used here).
The output of this math function is 0 if all signal FFT peaks are below the reference threshold and 1 if a peak is over the limit.



To display the reference curve in the FFT visual control together with the current FFT, use the “curve” tab in visual control properties and select the reference signal to be displayed.





The output values 0 and 1 can be used to display information in a “discrete display” (add indicator lamp and change properties), as described in an earlier example.

Other

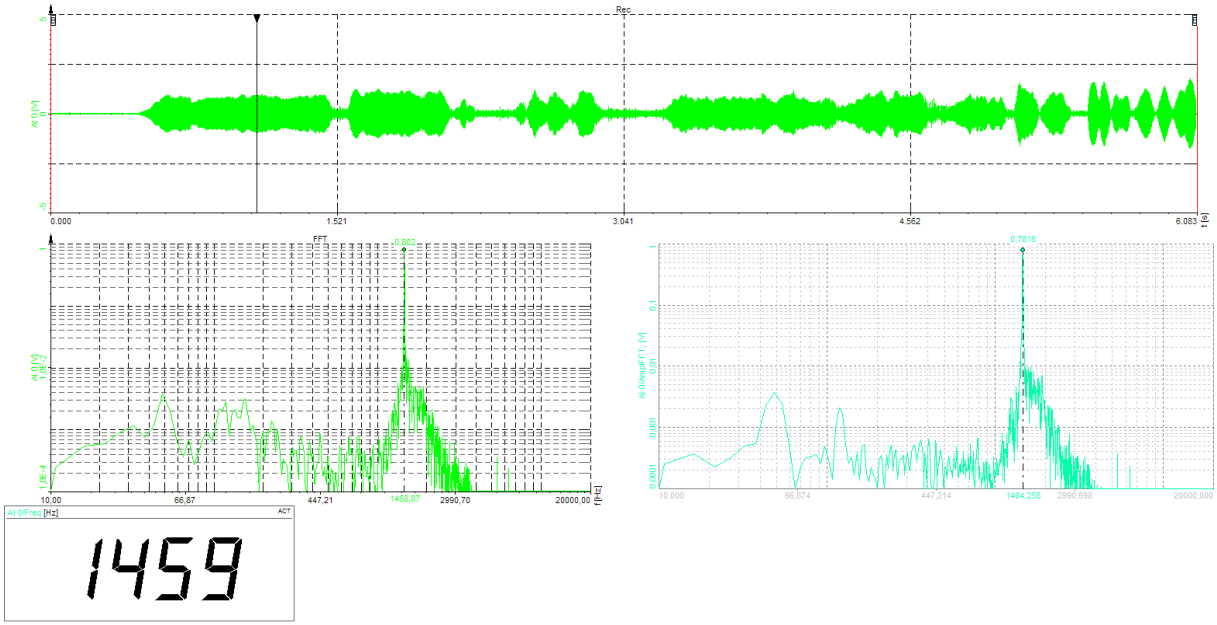
Sound card demo – detect peak frequency with exact frequency

We no use the on-board soundcard for demonstration of the FFT spectral analysis and the exact frequency algorithm. The test signal is a human whistle.

We have two possibilities to view a FFT analysis. The first is the display instrument FFT (visual control) where we have the advantage to change FFT parameters during measurement or analysis and to find a good balance of frequency resolution and update rate. Additional features here are the display of RMS value, a list of maximum peaks and marked peaks. However, it is not possible to do further calculations or to use the FFT channel in math, as it is a display instrument only.

But we can also add a math-channel FFT, which is then displayed in a 2D diagram and can be fully used in math, even array mathematics and array statistics.

The picture below shows the FFT visual control to the left and the FFT math to the right with the same settings, and a digital instrument displaying the exact frequency of a near sine-wave (human whistle).



Hint:

A slight difference between the two FFT displays comes from the way block averaging is done. The display instrument tries to update as fast as possible and to reach the display update rate setup in global settings, therefore a block average is done as often as CPU power allows (overlap is very high), while FFT math does the block averaging as setup in the control with exactly the overlap that is set up.

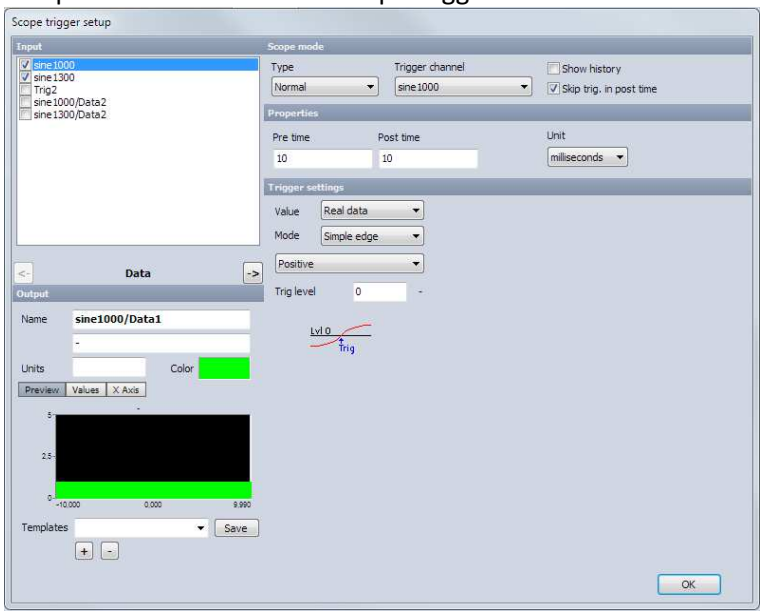
Triggering

Multiple scope triggers

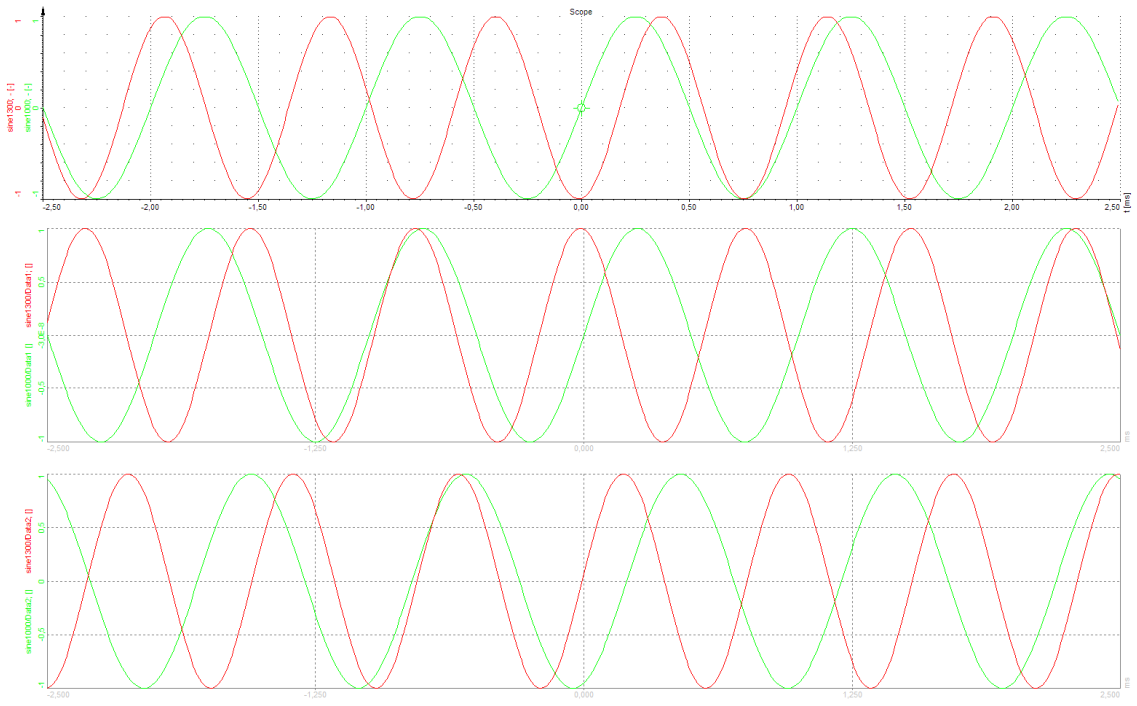
We would like to watch two signals in two scopes, each triggering to one of them. Due to a DEWESoft limitation only one trigger channel can be defined for scopes, no matter how many scope displays are shown on screen.

This can be solved with scope trigger math.

In scope trigger setup select all the channels to display and choose the first trigger-channel with pre and post time. Add another scope trigger module and do the same for the second trigger-channel.

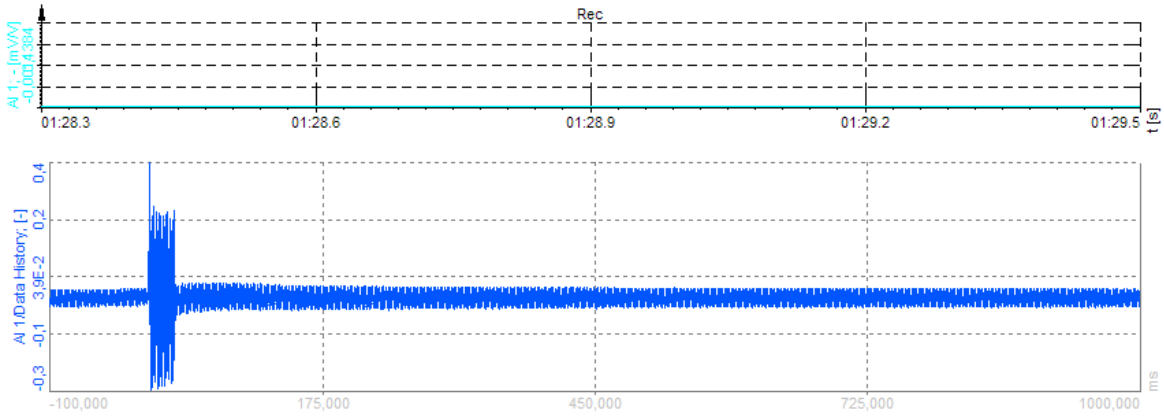


The first display shows the scope visual control, which can only have one trigger channel in total. The second and third controls are 2D displays, triggering to the first and second channel, respectively.



Scope Trigger, Array statistics

The scope trigger math can also be used to show a triggered event every time the trigger level is reached. Like in the example before the trigger level, pre and post time must be set correctly, then the triggered signal of specified duration is saved in the history buffer. Statistical values are then calculated with array statistics, like the maximum value and maximum position in ms or samples.



Trig, - ACT

5

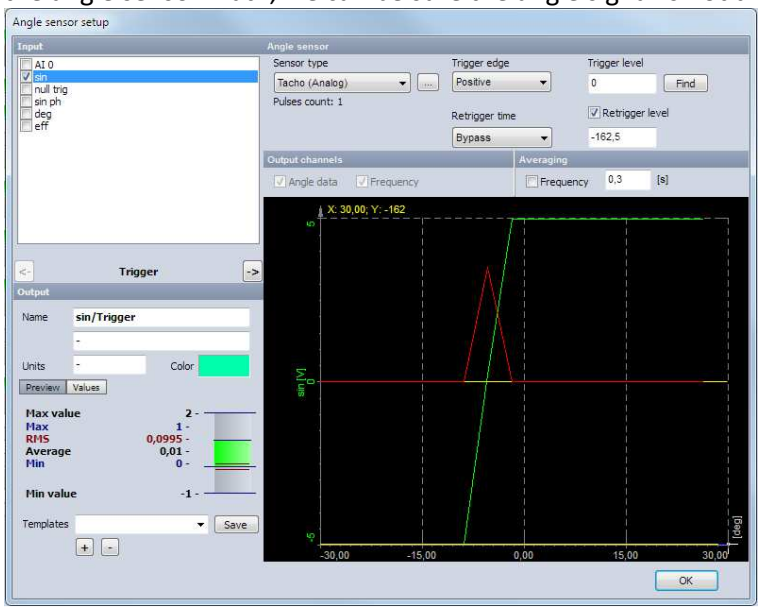
Trig -	AI 1/Data History/MAXVAL -	AI 1/Data History/MAXCLASS
5,000	0,4	ms=0,200
4,000	0,3	ms=0,200
3,000	0,3	ms=0,100
2,000	0,2	ms=0,200
1,000	0,3	ms=0,200

Phase angle firing simulation

First we need a signal to work so we create a sine wave with 325 V peak.

325 * sine(50) (sin)

Then we need angle sensor math to get one trigger per period. When we push “Find” the zero crossing is correctly detected and retrigger level is set to -162.5 (325/2) of the negative wave. Using the angle sensor math, we can be sure the angle signal is not drifting away from the signal.



Now we need a logical signal that depends on the angle we will select later on.

if('sin/Angle' mod 180 < 'angle', 1, 0) (null trig)

Starting from every half-period, all values smaller than angle are zero.

Another if condition sets the output signal to zero if above condition is true or to the original signal otherwise.

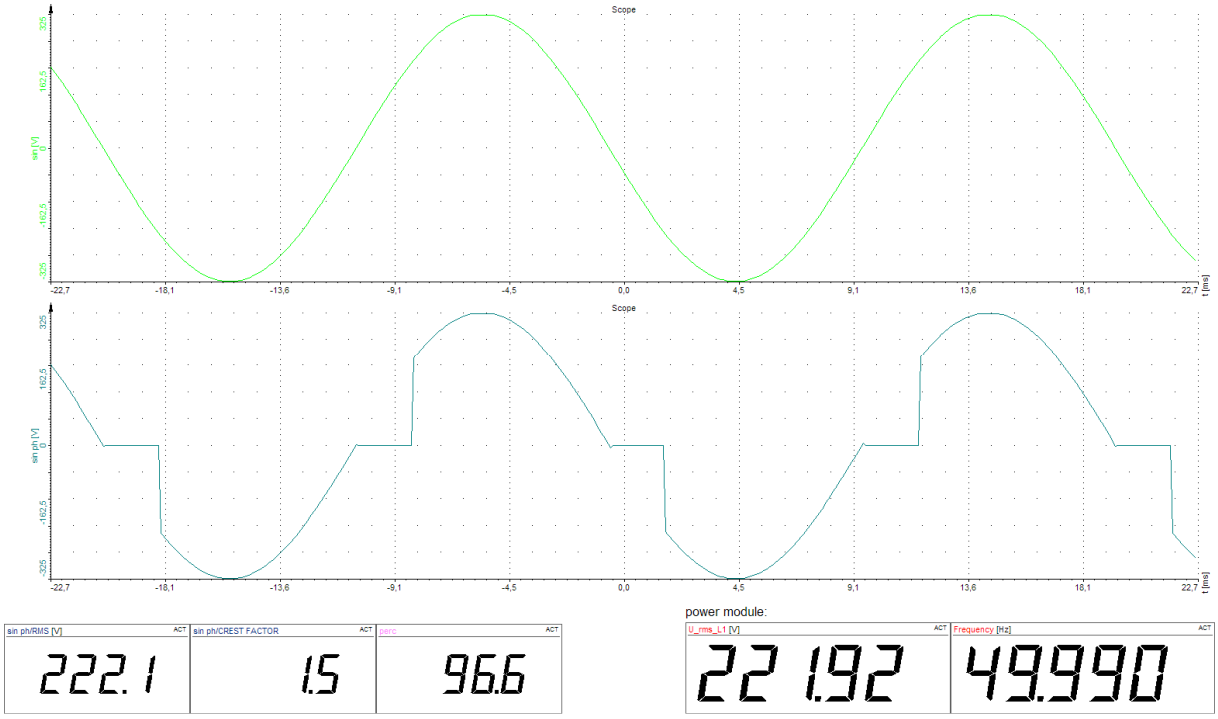
if('null trig'>0, 0, 'sin') (sin ph)

As a last step the RMS value for this signal is then calculated by multiplying with square-root of 2:

'sin ph' * sqrt(2)

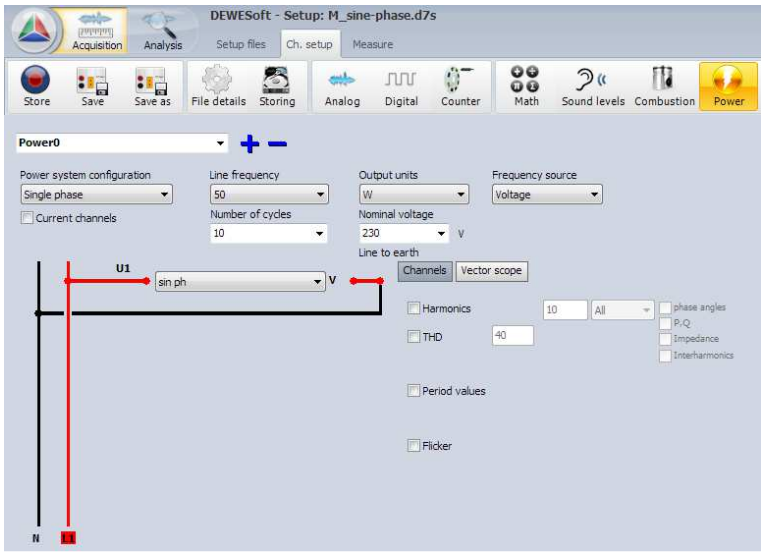
The percentage of full power (230 V RMS) is then calculated by

'sin ph/RMS'/230*100



For RMS value and crest factor calculation we use basic statistics (displayed at the bottom left corner) and for comparison the DEWESoft power module (displayed at the botoom right corner).

The DEWESoft power-module is simply set up with our phase angle firing signal (without RMS factor) as a single phase voltage input, as shown below:



Hint:
A small difference in RMS value is OK here, because the angle sensor detection is not perfect and the power module in general is more exact, as it uses internal resampling to optimize FFT for period lenghth.

Counter game – time, hold, keypressed, latch math, if, round, mod, basic statistics

Here is a counter game done with DEWESoft math, just for fun.
The goal is to press the space key when the counter number is dividable by 10. In the display we see the running counter, the last value when the key was pressed, number of values, an info box with a text message, error of the last triggered value (how far from 10 is it) and the total error average, finally a table overview of all tries.

COUNTER GAME
Try to press space key when counter can be divided
by 10 - press space to start

Counter	last value (hold)	last value (latch)	no of values
450	201	201	8

Info	error	error average
Game over	1	2

Table of values

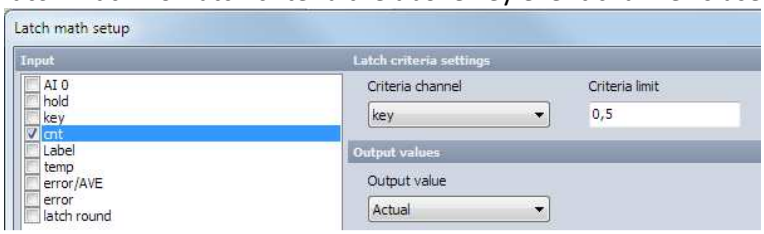
Time	latch round	error
20.113	201,0000	1,000
18.934	189,0000	1,000
17.642	176,0000	4,000
10.045	100,0000	0,000
8.163	82,0000	2,000
6.032	60,0000	0,000

Formulas used:

running counter: **time * 10**
hold value by formula: **hold('cnt', keypressed(32))**
To hold just the last value the hold function could be used, but here we need a value-history, so we need latch math.

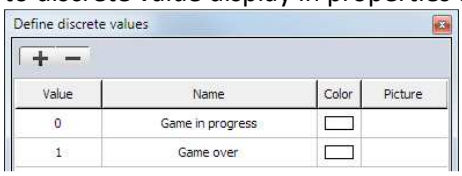
key event (for latch math) **keypressed(32)**

latch math: for latch criteria the above key event channel is used



label value channel: **if('Latch index'>5,1,0)**
Only 5 tries per game, info shows “Game in progress or game over, depending on label value.

Here is the corresponding visual control**, discrete value display** (indicator lamp from menu, change to discrete value display in properties dropdown):



temp value **round('cnt/Latch')mod 10**
Mod delivers the remainder of a division by 10 here.

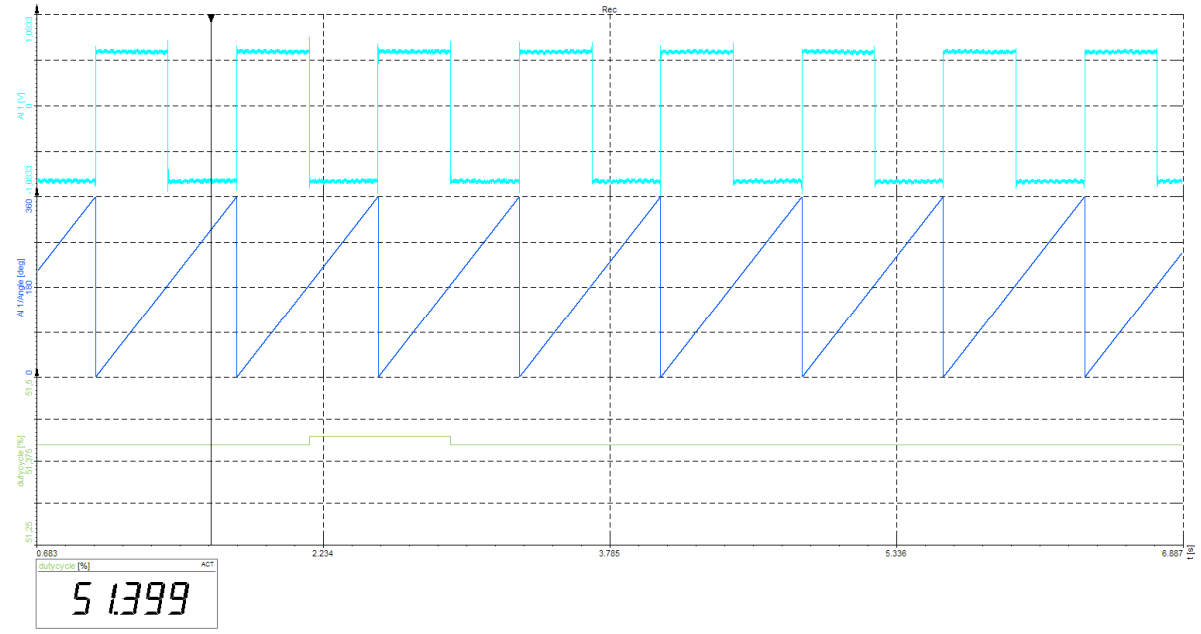
if condition **if('temp'>5,10 - 'temp', 'temp')**
We would like to have the error to the nearest number dividable by 10, so the maximum error is 4 to the next highest multiple of 10 or 5 to the next lowest multiple of 10. The corresponding math function does the following: if the remainder is bigger than 5 the result is 5 – remainder, else it is the remainder

Add **basic statistics** with average checked, type is single value.

remove decimals **round('cnt/Latch')**

Add visual control “tabular values display”, uncheck time in properties and add the rounded latched channel.

Duty cycle calculation – fast signals (with angle sensor)



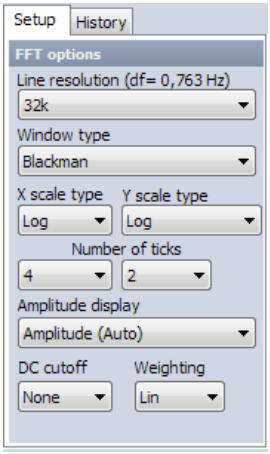
The analog input signal (cyan) should be analysed in terms of duty-cycle calculation.
For this we need to find the period length, or more exactly the 360 °angle over one whole period with the **angle sensor math**, set to “Tacho(analog)”, positive edge and trigger level 0.
At the start of the negative half-cycle we need to take the actual angle and divide it by 100 % which is here 360°, finally multiply with factor 100 to get the percentage for the duty cycle.

hold('AI 1/Angle','AI 1' < 0) / 360 * 100

The hold value is triggered every time the trigger condition is met, so every time a new negative half-cycle starts a new angle value is calculated.

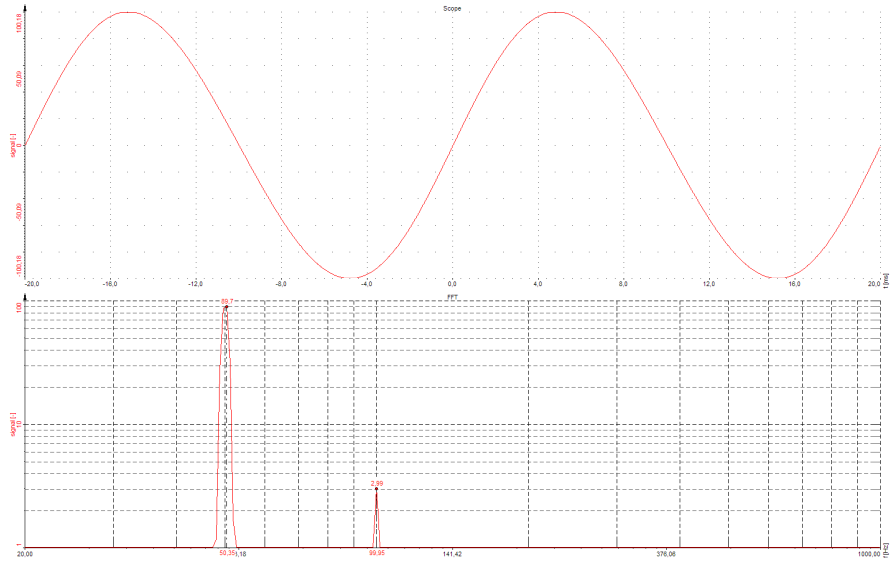
Spectral analysis

Find harmonic components of periodic signal



If we have an unknown periodic signal, in time domain it is not always possible to distinguish overlaid frequencies with different amplitudes, but in frequency domain these signals can be viewed separately.

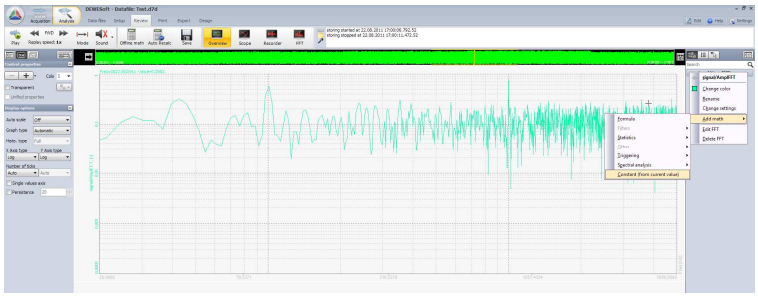
In the example below we see an unknown periodic signal on a scope display, which resembles a sine signal. If we have a look at it in frequency domain with the FFT visual control, we see two different signals overlaid, with different amplitudes (see FFT properties on the left).



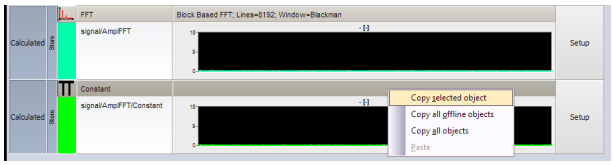
Copy FFT spectrum to math array constant

If we need to use a spectrum for calculation or we would like to compare it to actual data it is possible to add a math constant in analysis mode.

The spectrum from FFT math (displayed with a 2D visual control) is copied by just right-clicking on the input channel and choosing “Add math” and “Constant”.



Now we get a new math constant, which can also be used for calculation (array maths) or reference in visual controls.

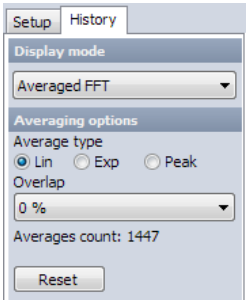


To be able to copy this constant from math channel to Excel for example, right click somewhere in the channel and choose “copy selected object”.

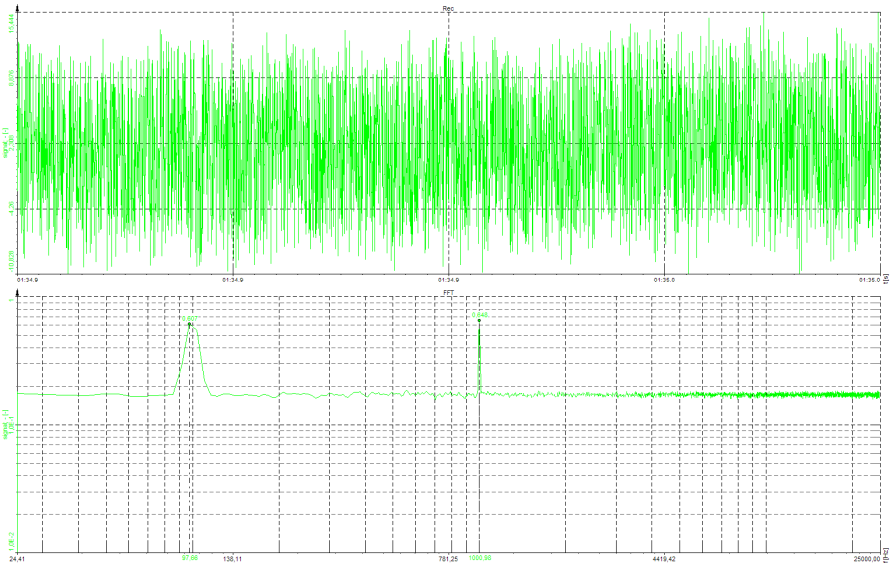
Hint: From the FFT visual control no data can be copied.

Find harmonic components of noise signal (if present)

If there is a lot of noise, in scope or recorder (time domain) we see nothing but noise.
Nevertheless a periodic signal may be present, which can be visualized and measured with spectral analysis.



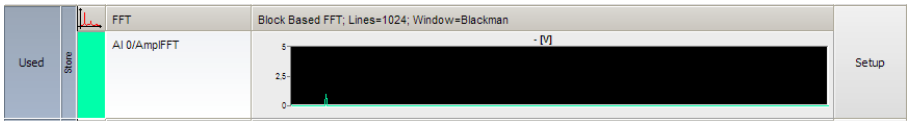
To improve signal to noise ratio we use “Averaged FFT” here.



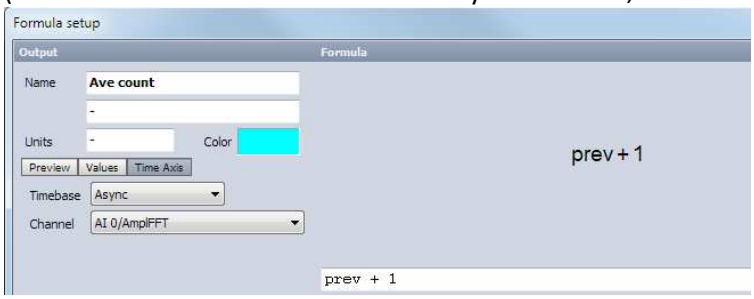
The lower graph shows the two periodic signals hidden under lots of noise in time domain, but clearly visible after same averaging.

FFT average in math– prev function

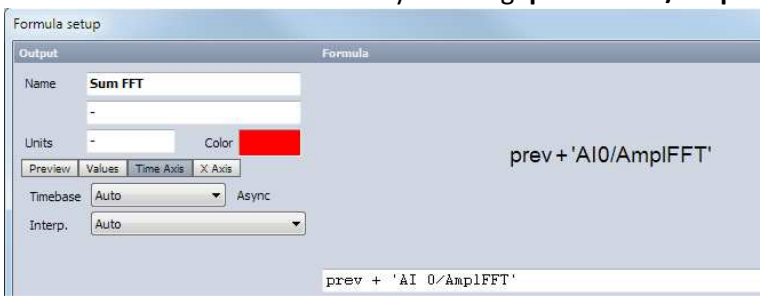
We use an analog test signal and block based FFT mathematics.



Next, we create an average counter (Ave count) with formula: **prev + 1**
(be sure to select the FFT channel as async timebase, see below)



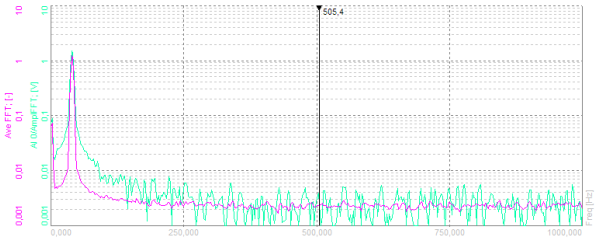
The SumFFT channel is created by entering: **prev + 'AI 0/AmplFFT'**



(notice the timebase is set to auto, which is async here)

Finally, the AveFFT formula is entered as **'Sum FFT'/'Ave count'**

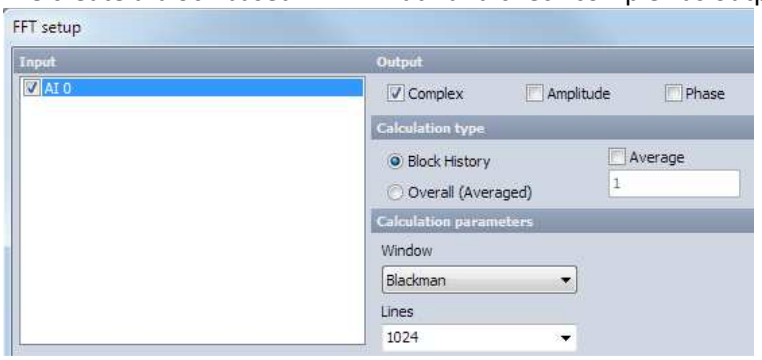
This average FFT channel can now be used for further calculations, such as maximum frequency: **maxpos('Ave FFT')**



Complex FFT – extract real and imaginary part

When we have a complex signal, we use the math functions **.re** and .**im** to extract real and imaginary parts of the signal for further calculation.

We create a block based FFT in math and check complex as output type.



From this signal we can now extract real and imaginary parts, respectively.

'AI 0/CplxFFT'.re

'AI 0/CplxFFT'.im

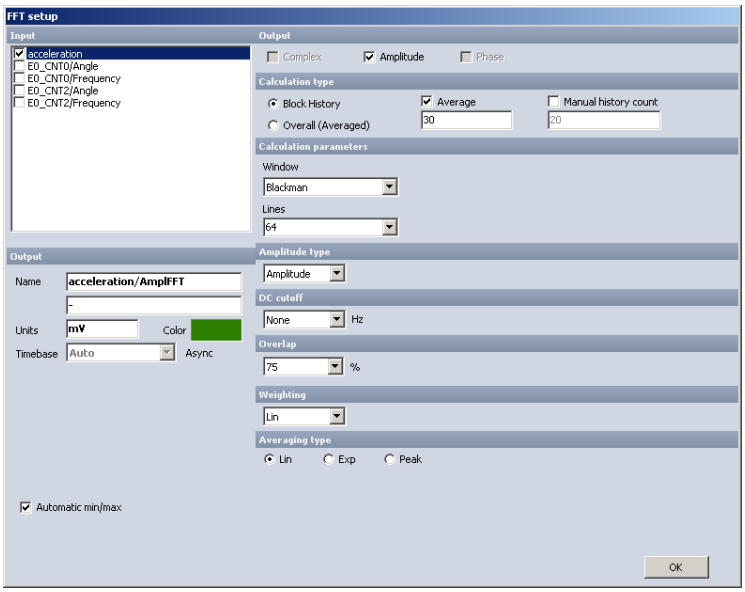
DEWESoft – General hints

Hint: How to add an average FFT to a datafile

If you did not add an average FFT during measurement, it is a little tricky to achieve this later (because you cannot simply change the FFT instrument to Average afterwards).

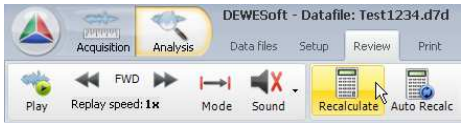
But it can be done with MATH –
in post-processing!

Open the datafile (*.d7d) and add in the ofline math a FFT function. This is listed under the 6th menu point.

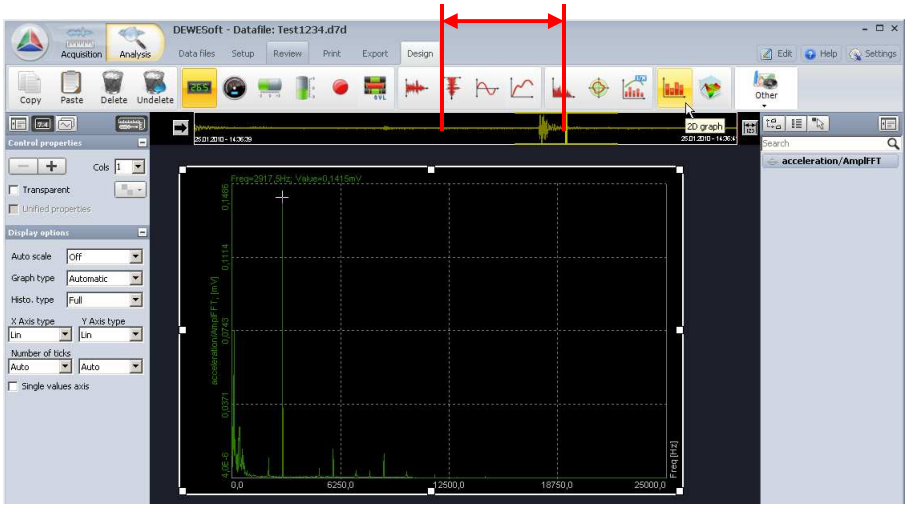


Choose the channel, set the average cycles and the resolution (lines).
We suggest a window overlap of 75%.
Quit with OK and go back to „Review“ to the overview.

Now the label of the button “offline math” has changed to “recalculate”. Start the recalculation.



Depending on the channel count, resolution and data amount this can take a little while.

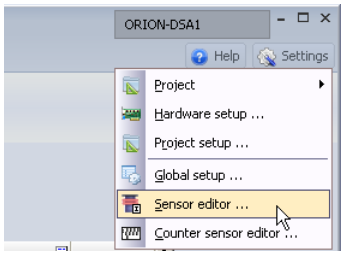


For the calculation results in a matrix, the instrument to be used is the „2D graph“.

Hint when scrolling through: in the overview-instrument on top the yellow T-bar shows the FFT window-size.

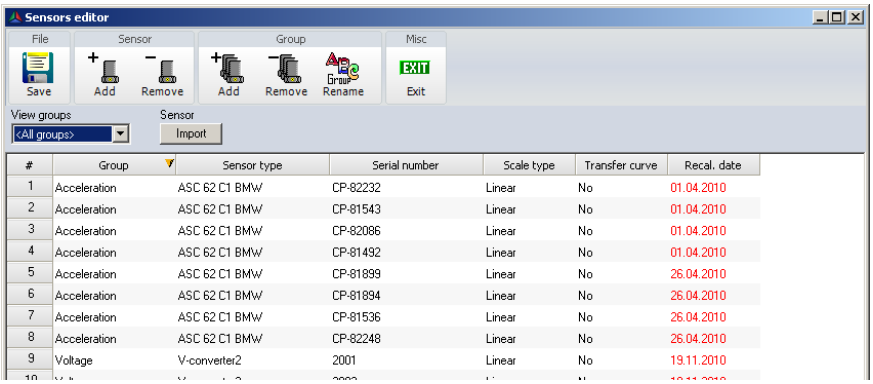
Hint: Non-linear scaling of sensors

A linear sensor scaling can easily be done in the channel setup. How about correcting non-linear sensors?



This can be done in the Sensor editor.

Sensors are aligned in groups (e.g. Acceleration, Voltage, Current…).

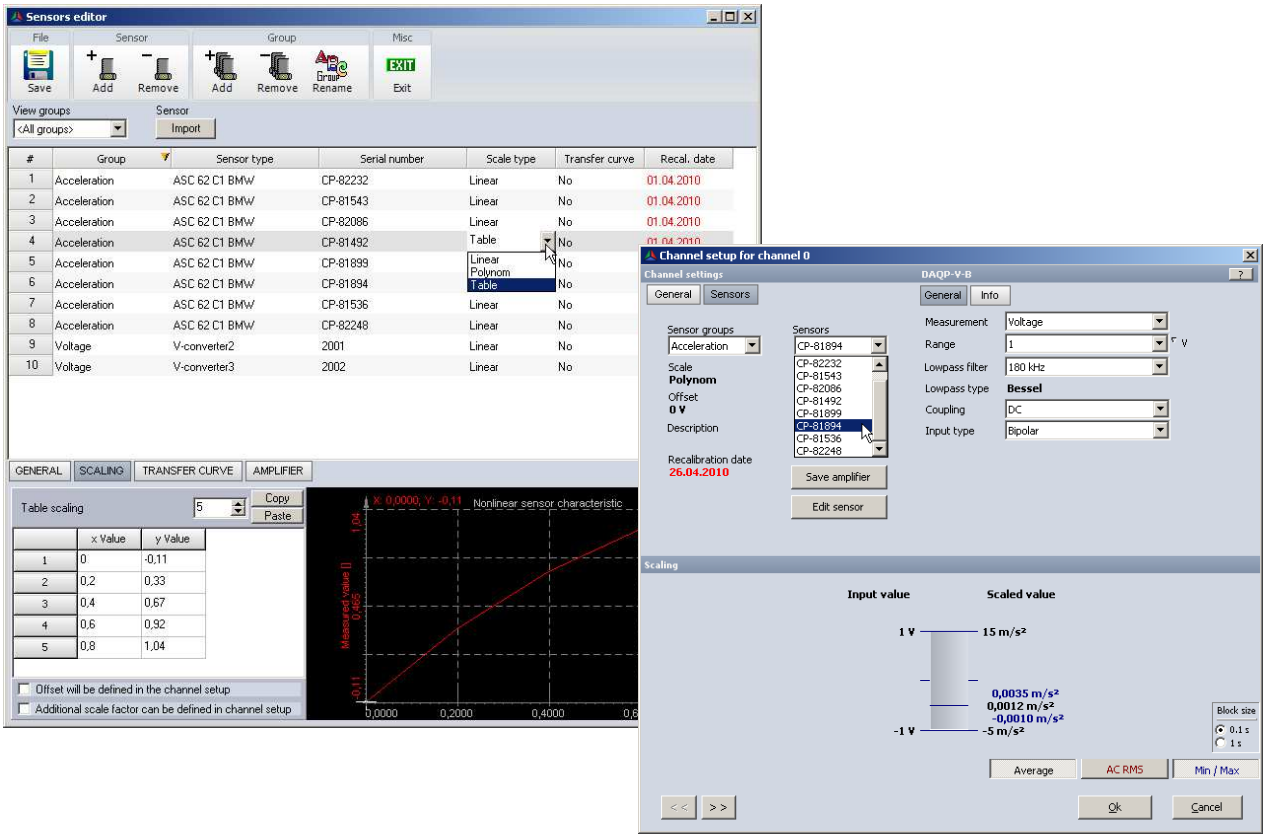


#	Group	Sensor type	Serial number	Scale type	Transfer curve	Recal. date
1	Acceleration	ASC 62 C1 BMW	CP-82232	Linear	No	01.04.2010
2	Acceleration	ASC 62 C1 BMW	CP-81543	Linear	No	01.04.2010
3	Acceleration	ASC 62 C1 BMW	CP-82086	Linear	No	01.04.2010
4	Acceleration	ASC 62 C1 BMW	CP-81492	Linear	No	01.04.2010
5	Acceleration	ASC 62 C1 BMW	CP-81899	Linear	No	26.04.2010
6	Acceleration	ASC 62 C1 BMW	CP-81894	Linear	No	26.04.2010
7	Acceleration	ASC 62 C1 BMW	CP-81536	Linear	No	26.04.2010
8	Acceleration	ASC 62 C1 BMW	CP-82248	Linear	No	26.04.2010
9	Voltage	V-converter2	2001	Linear	No	19.11.2010

In the column „Recal. Date“ you will be remembered for the annual calibration.

We suggest to store the sensors with the serial number for the correct identifying.

Change the scale type to Linear / Polynom / Table and enter your values below.
The sensor is then easily selected in the channel setup.

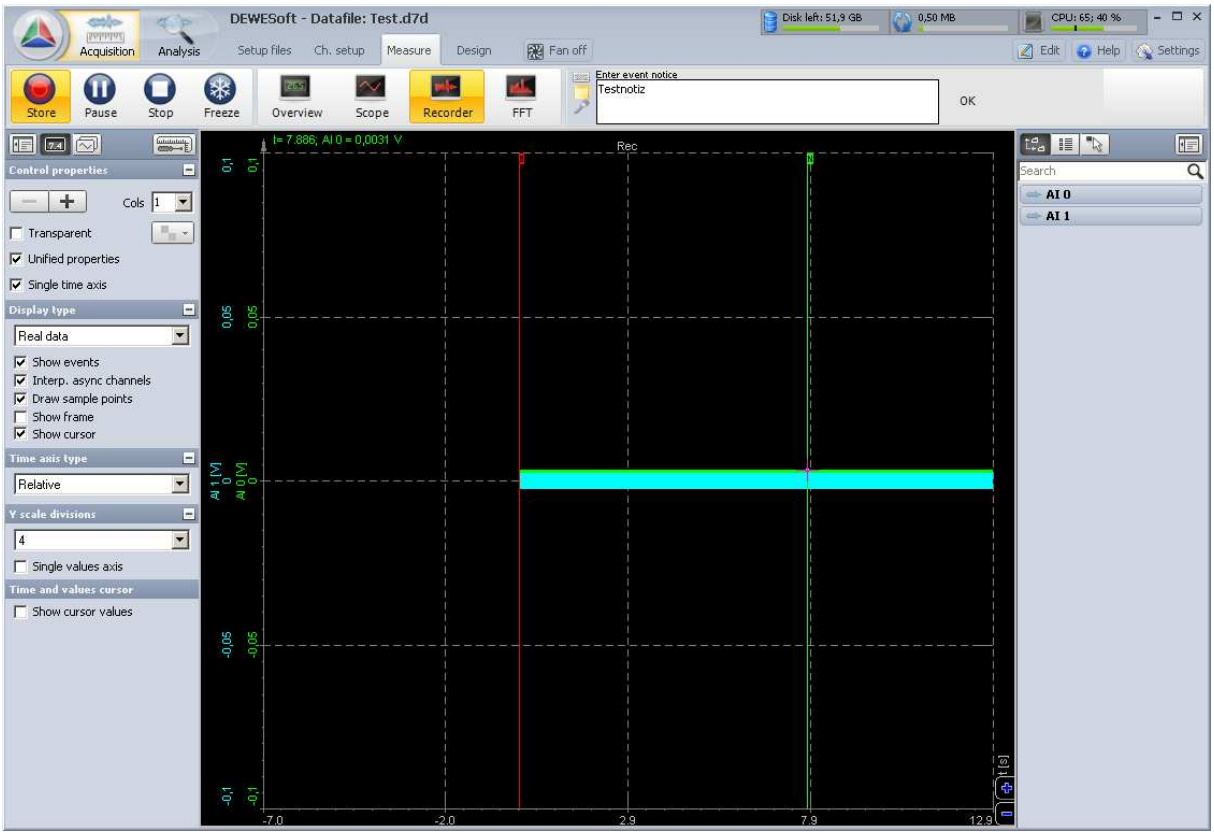


	x Value	y Value
1	0	-0,11
2	0,2	0,33
3	0,4	0,67
4	0,6	0,92
5	0,8	1,04

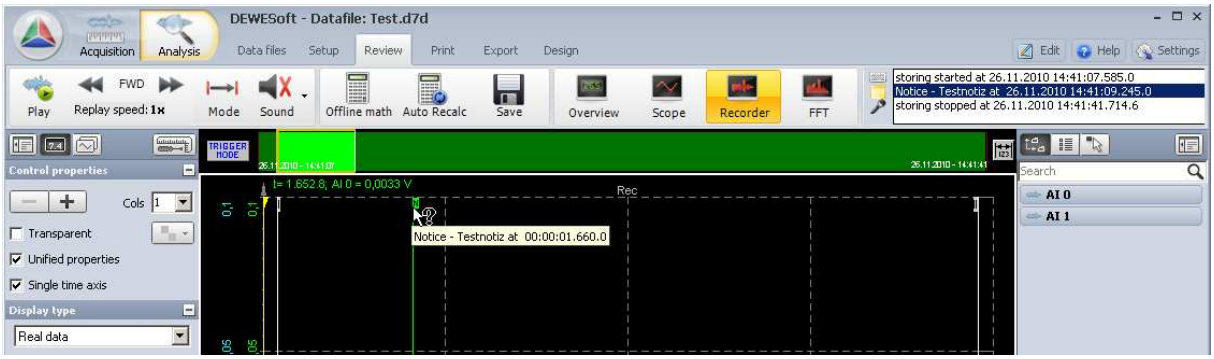
Hint: Add notes during measurement

If you press the button ‘n’ during storing a green marker occurs.
At the same time the Event window opens and you can enter your note.

You don’t have to hurry because for the note the time you pressed the ‘n’ key is stored.



This is then saved to the datafile. Place the mouse pointer over the event and you see the note.



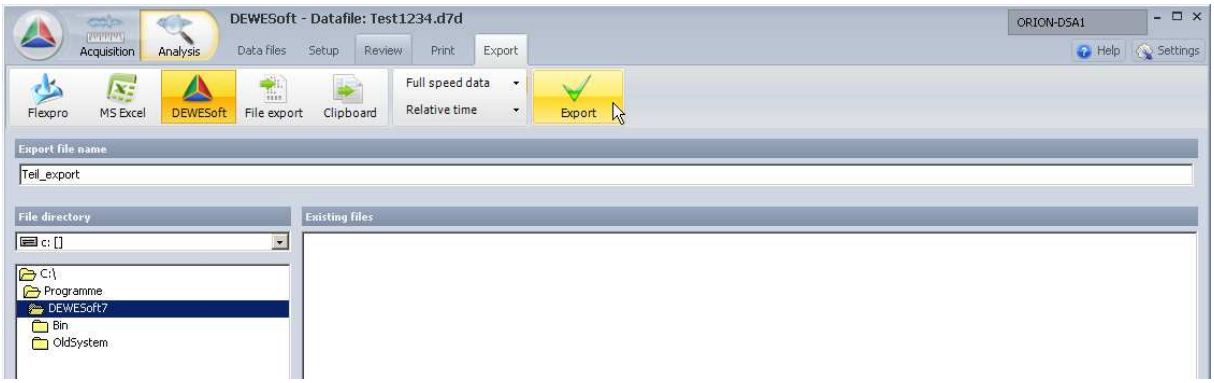
Hint: Downsize DEWESoft datafiles

DEWESoft export -> DEWESoft

To reduce the size of a stored DEWESoft datafile you can for example export only the interesting part to a new DEWESoft datafile.
Therefore take a recorder instrument and select the area you want to export with the cursors. On the overview instrument on top you see the selection in relation to the whole file.

Then go to Export and select the option DEWESoft.

All channels are exported.



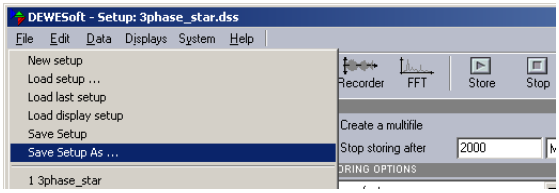
The new file is much smaller and has all the data you are interested in.

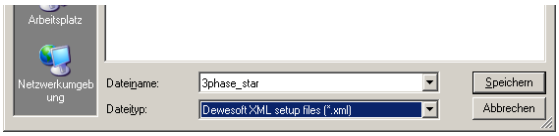
Hint: Porting DEWESoft6 setupfiles (*.dss) to DEWESoft7 (*.d7s)

One of the big advantages of DEWESoft7 is the open file structure. Let's have a look at a practical example.

Open the old setup in DEWESoft 6.

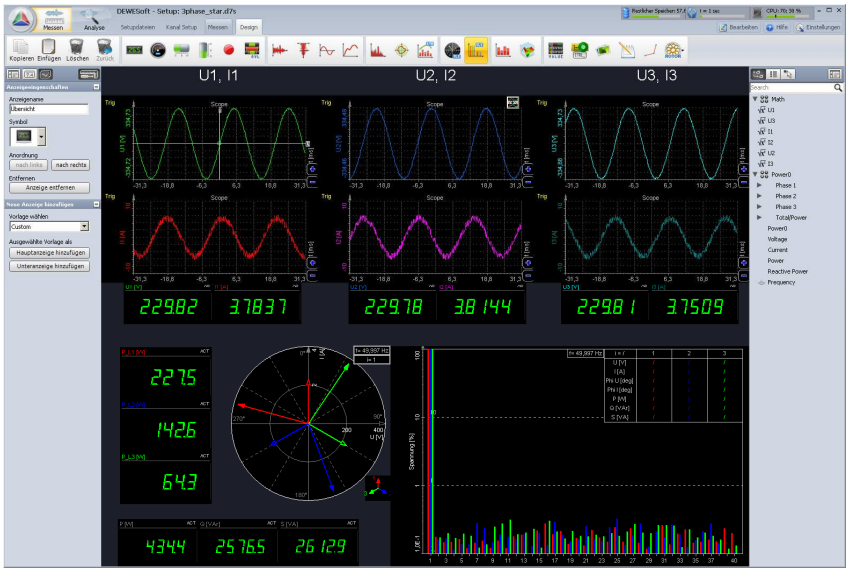
Next save the setup as an XML file.





Close DEWESoft6 and open DEWESoft7.

Load the XML setup file in DEWESoft7, chose XML as filetype.

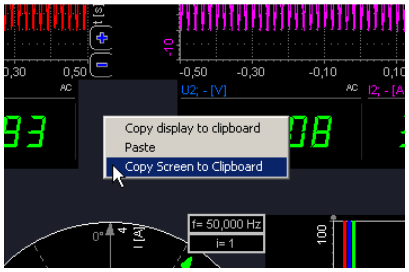


Hint:

You can also **copy a whole screen** of instruments.

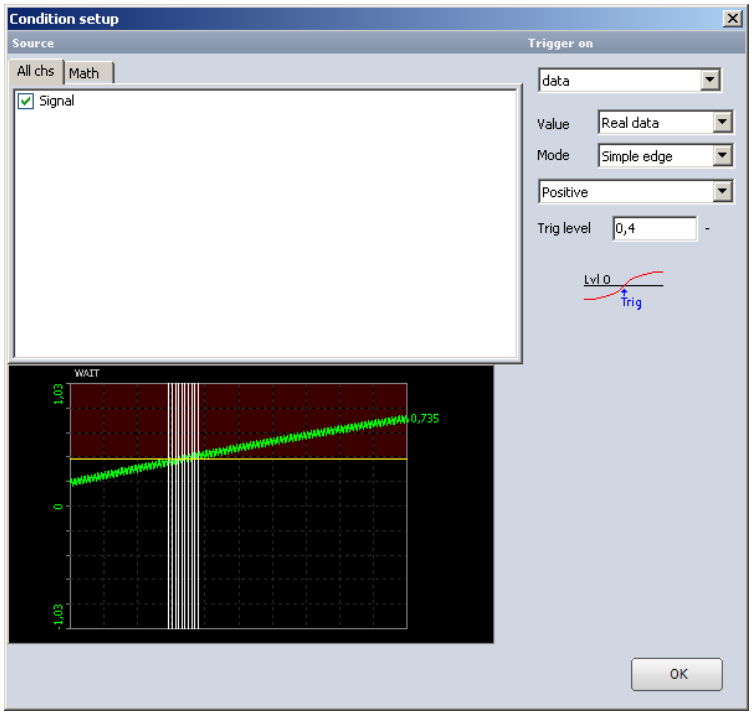
Just right click on the background and chose "Copy Screen to Clipboard".

Now you can Paste all the instruments to another Screen such as Recorder, Scope, … feel free to make your own layout.



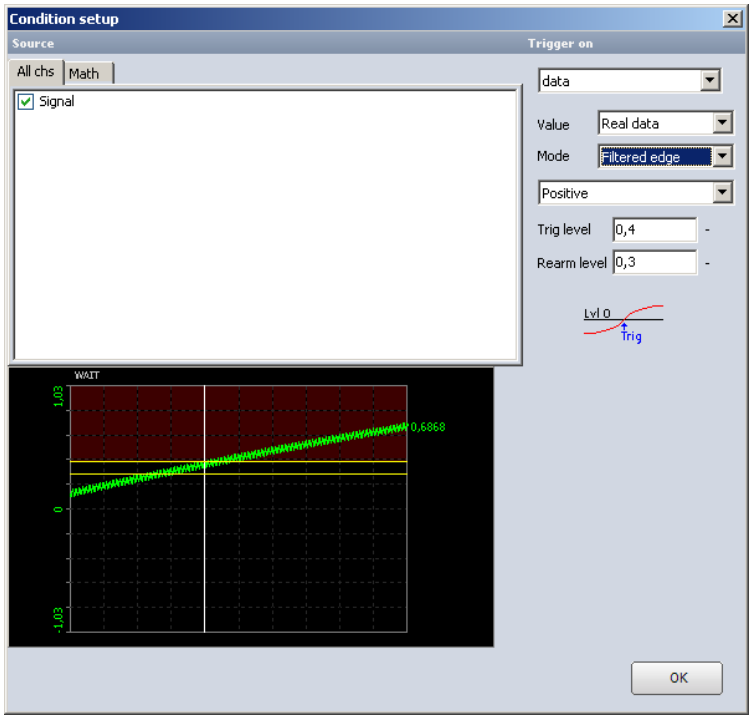
Hint: Trigger on a noisy signal (filtered edge)

If you have noise on your signal and use a trigger level, it will trigger multiple times.



To prevent this there is the option “Filtered edge”.

Specify a re-arm level which should be a little bit below your trigger level. The trigger will only work again, if the input signal falls below the rearm level.



Hint: Keypressed event as manual storing-trigger

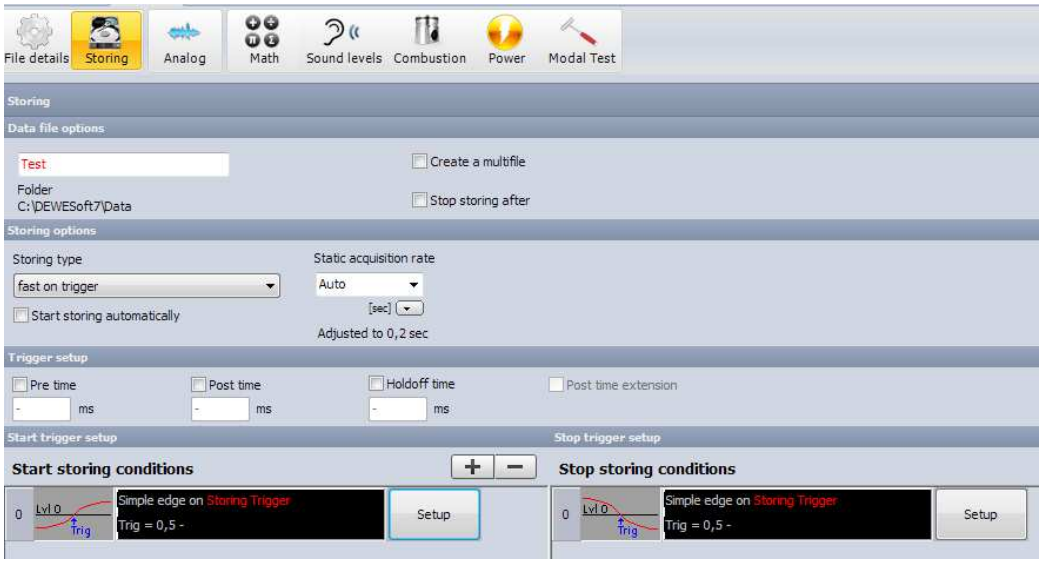
We would like to use the spacebar to start and stop storing of measurement.

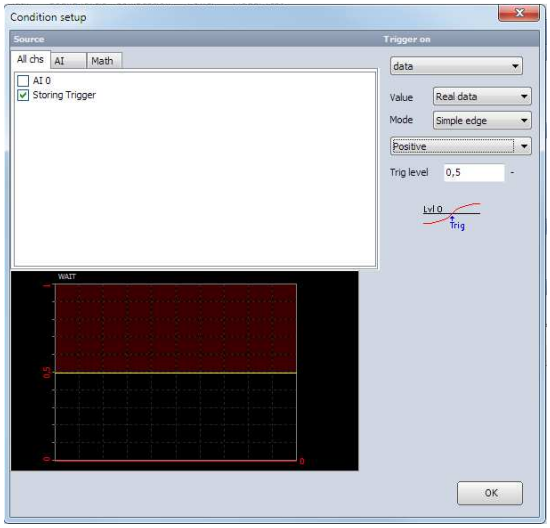
For this we need to create a trigger channel from the keypressed event, which is a simple math formula: **edge(keypressed(32), keypressed(32))**

"keypressed" gives a pulse when the specified key is pressed, edge triggers on the flank and since the same input is used as a rearm condition, only every second keypress triggers the signal to high, every other resets the input.

Now all we have to do is define a start and stop storing condition in the menu. We pick our math formula as a trigger channel and define trigger level as 0.5 with a positive edge for start storing and the same with a negative edge for stop storing.

This way we can start and stop our measurement by pressing spacebar.



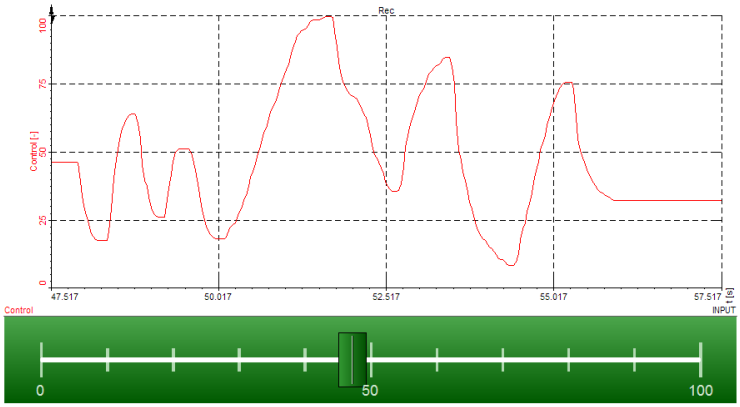


Control channel – variable for control button , slider input, …

Control channel is a free plugin (DLL into Addon directory).
The channel is added with the plus sign, then channel name and description is given and the min and max values are defined.

In design mode the visual control “Input control display” is added, with the following properties:

Control channel is selected and also here the min and max values are set.
Now the slider can be used to enter parameters for a control channel, which is also addressible in math and can be used for further calculations.

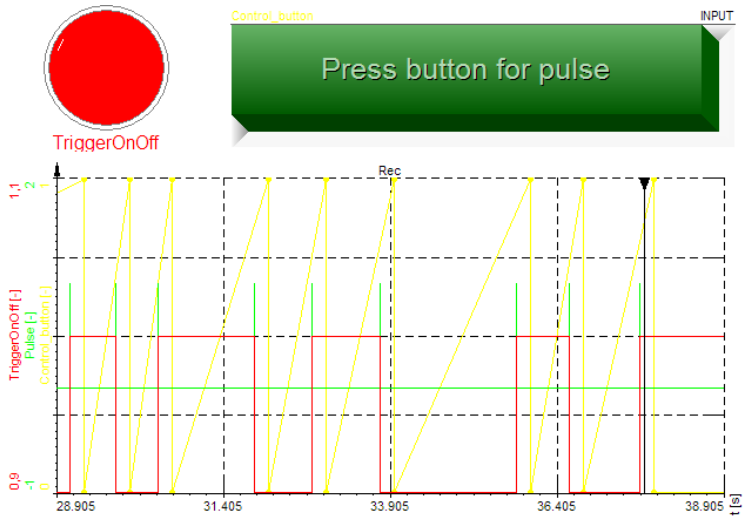


Another application is to use a control button as a trigger signal, which can also be addressed in other math channels.
The control channel assigned to the button gives 1 when the button is pushed (yellow signal in screenshot). This signal can be used to generate a trigger pulse with the formula (green signal):

edge('Control_button')

To get an on-off signal out of this pulse signal, which is here used for the indicator lamp the following formula is used:

ecnt('Pulse') mod 2


Control_button
INPUT
Press button for pulse
TriggerOnOff
Rec
1,1
2
1
0,9
-1
0
TriggerOnOff [-]
Pulse [-]
Control_button [-]
28.905
31.405
33.905
36.405
38.905
t [s]

DEWESoft 7.1 – array math

DEWESoft 7.1 – array calculation used for FLIR infrared camera

The FLIR infrared camera image is saved as a matrix (2 dimensional array) in DEWESoft. Depending on the mounting of the camera X and Y axis can be swapped, so here the data X axis is the vertical axis in the 3D diagram.

We would like to see two temperature-lines now; one horizontal, which shows all 8 hot ICs and one vertical. For this we need to extract the right index start position and show the whole array for that dimension, which can be done with the [:] command.

MATH:

Line 213 X: **'FLIR A3X0'[:][213]**

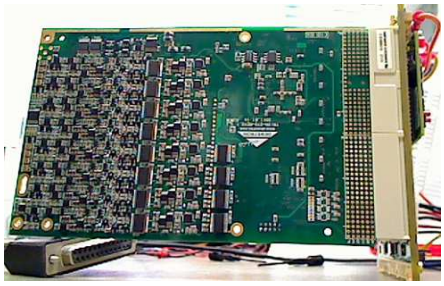
Line 182 Y: **'FLIR A3X0'[182][:]**

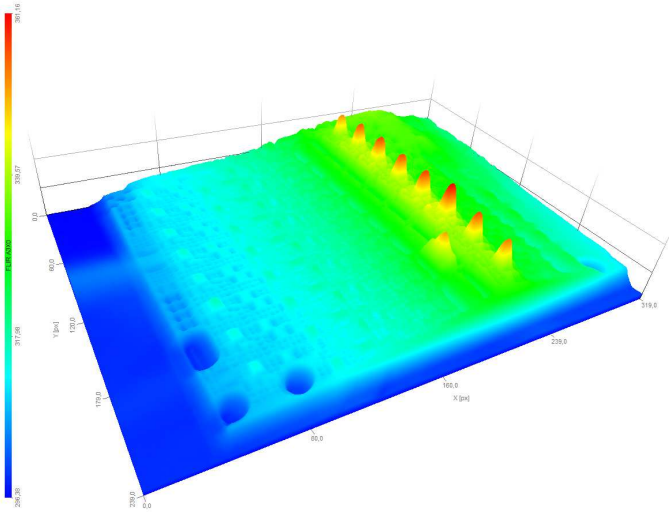
Another application here would be to define an area for temperature averaging to smooth the temperature response close to temperature gradients, or to average over larger areas. This can be done like this:

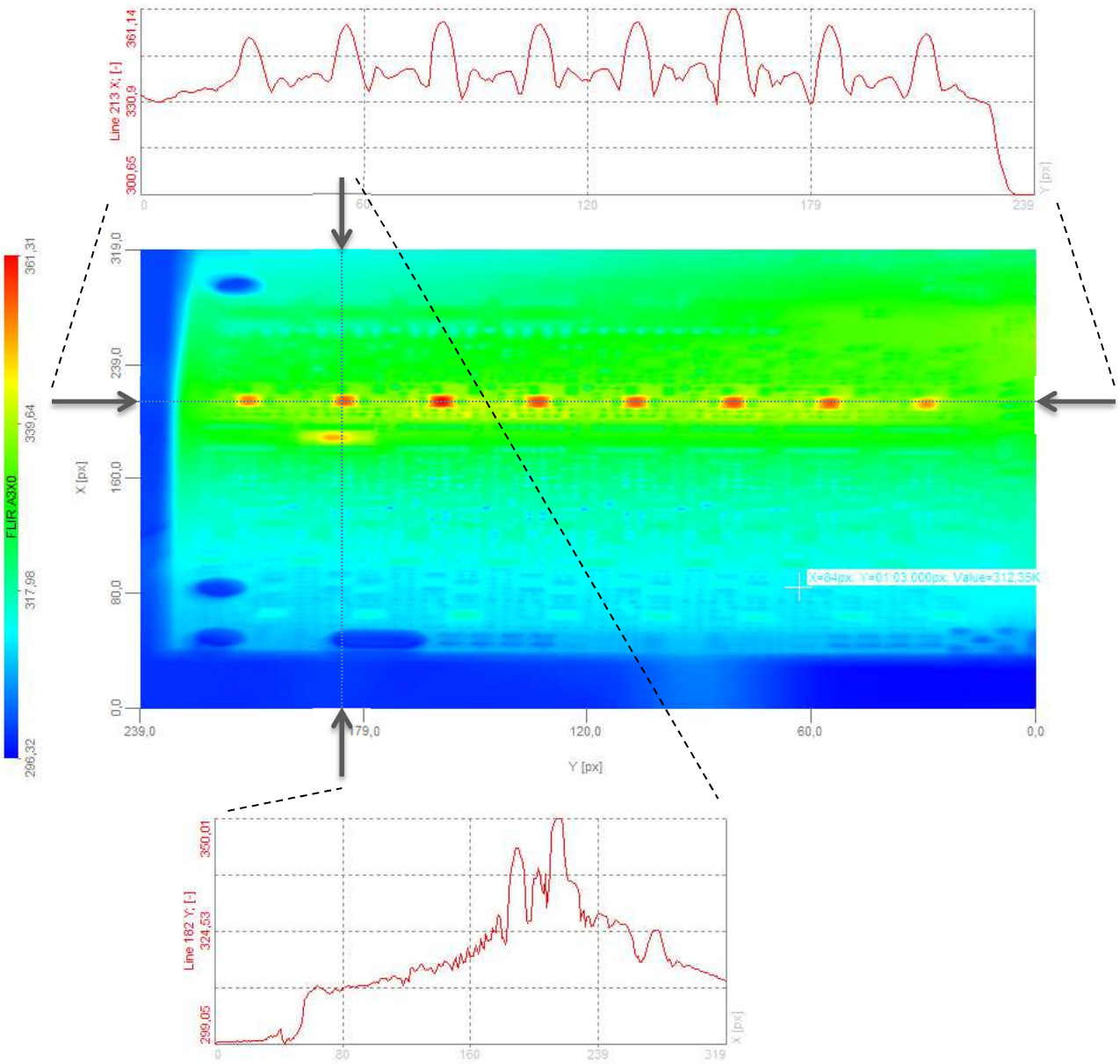
Area: **'FLIR A3X0'[170:190][200:230]**

Average: **avg('Area')**

Maximum: **max('Area')**

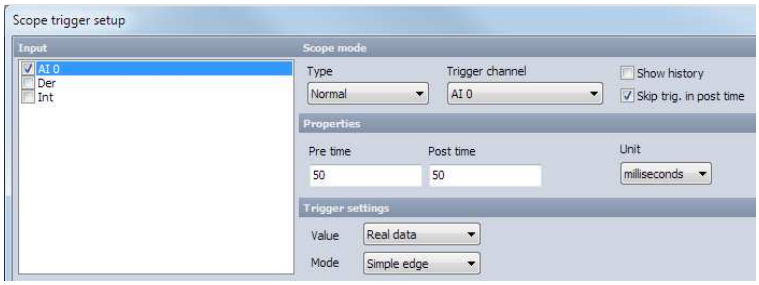





Line 213 X; [-]
361,14
330,9
300,65
0
60
120
179
239
Y [px]
FLIR A3X0
361,31
339,64
317,98
296,32
X [px]
319,0
239,0
160,0
80,0
0,0
X=64px; Y=01:03.000px; Value=312,35K
239,0
179,0
120,0
60,0
0,0
Y [px]
Line 182 Y; [-]
350,01
324,53
299,05
0
80
160
239
319
X [px]

DEWESoft 7.1 – derivation and integration of an array (vector)

To create an array we use the scope trigger function on an analog test signal (see below).



Discrete derivation:

x[k] – x[k-1]

To calculate derivation for an array the following formula is used:

'AI 0/Data'[1:len-1] - 'AI 0/Data'[0:len-2]

Discrete integration:

$\sum_{i=-\infty}^{k} x[i]$

To calculate integration for an array we use the new integrate function:

integrate('AI 0/Data')