ibaAnalyzer

Ausdruckseditor

Handbuch Teil 3

Ausgabe 8.2

Messsysteme für Industrie und Energie

www.iba-ag.com

Hersteller

iba AG

Königswarterstraße 44

90762 Fürth

Deutschland

Kontakte

Zentrale	+49 911 97282-0
Support	+49 911 97282-14
Technik	+49 911 97282-13
E-Mail	iba@iba-ag.com
Web	www.iba-ag.com

Version	Datum	Revision	Autor	Version SW
8.2	04-2023	Syntax der Ausdrücke	TS	8.2.0

Inhalt

1 Zu dieser Dokumentation

Diese Dokumentation beschreibt die Funktion und die Anwendung der Software *ibaAnalyzer*.

1.1 Zielgruppe

Diese Dokumentation wendet sich an ausgebildete Fachkräfte, die mit dem Umgang mit elektrischen und elektronischen Baugruppen sowie der Kommunikations- und Messtechnik vertraut sind. Als Fachkraft gilt, wer auf Grund der fachlichen Ausbildung, Kenntnisse und Erfahrungen sowie Kenntnis der einschlägigen Bestimmungen die übertragenen Arbeiten beurteilen und mögliche Gefahren erkennen kann.

Diese Dokumentation wendet sich insbesondere an Personen, die mit der Auswertung von Mess- und Prozessdaten befasst sind. Da die Bereitstellung der Daten mit anderen iba-Produkten erfolgt, sind für die Arbeit mit *ibaAnalyzer* folgende Vorkenntnisse erforderlich bzw. hilfreich:

- Betriebssystem Windows
- *ibaPDA* (Entstehung und Struktur der Messdateien)

1.2 Schreibweisen

In dieser Dokumentation werden folgende Schreibweisen verwendet:

Aktion	Schreibweise
Menübefehle	Menü *Funktionsplan*
Aufruf von Menübefehlen	*Schritt 1 – Schritt 2 – Schritt 3 – Schritt x* Beispiel: Wählen Sie Menü *Funktionsplan – Hinzufügen – Neuer Funktionsblock*
Tastaturtasten	<Tastenname> Beispiel: <Alt>; <F1>
Tastaturtasten gleichzeitig drücken	<Tastenname> + <Tastenname> Beispiel: <Alt> + <Strg>
Grafische Tasten (Buttons)	<Tastenname> Beispiel: <OK>; <Abbrechen>
Dateinamen, Pfade	`Dateiname`, `Pfad` Beispiel: `Test.docx`

1.3 Verwendete Symbole

Wenn in dieser Dokumentation Sicherheitshinweise oder andere Hinweise verwendet werden, dann bedeuten diese:

Gefahr!

Wenn Sie diesen Sicherheitshinweis nicht beachten, dann droht die unmittelbare Gefahr des Todes oder der schweren Körperverletzung!

- Beachten Sie die angegebenen Maßnahmen.

Warnung!

Wenn Sie diesen Sicherheitshinweis nicht beachten, dann droht die mögliche Gefahr des Todes oder schwerer Körperverletzung!

- Beachten Sie die angegebenen Maßnahmen.

Vorsicht!

Wenn Sie diesen Sicherheitshinweis nicht beachten, dann droht die mögliche Gefahr der Körperverletzung oder des Sachschadens!

- Beachten Sie die angegebenen Maßnahmen.

Hinweis

Hinweis, wenn es etwas Besonderes zu beachten gibt, wie z. B. Ausnahmen von der Regel usw.

Tipp

Tipp oder Beispiel als hilfreicher Hinweis oder Griff in die Trickkiste, um sich die Arbeit ein wenig zu erleichtern.

Andere Dokumentation

Verweis auf ergänzende Dokumentation oder weiterführende Literatur.

1.4 Aufbau der Dokumentation

In dieser Dokumentation wird umfassend die Funktionalität der Software *ibaAnalyzer* beschrieben. Sie ist als Leitfaden zur Einarbeitung wie auch als Nachschlagedokument angelegt.

Ergänzend zu dieser Dokumentation können Sie für aktuellste Informationen zur installierten Programmversion die Versionshistorie im Hauptmenü *Hilfe – Versionshistorie* heranziehen (Datei `versions.htm`). In dieser Datei wird neben der Aufzählung behobener Programmfehler auch auf Erweiterungen und Verbesserungen der Software stichwortartig hingewiesen.

Außerdem wird mit jedem Software-Update, das nennenswerte neue Features enthält, eine spezielle Dokumentation "NewFeatures…" ausgeliefert, die eine ausführliche Beschreibung der neuen Funktionen bietet.

Der Stand der Software, auf den sich der jeweilige Teil dieser Dokumentation bezieht, ist in der Revisionstabelle auf Seite 2 aufgeführt.

Die Dokumentation von *ibaAnalyzer* (PDF- und gedruckte Ausgabe) ist in sechs separate Teile gegliedert. Jeder Teil hat seine eigene bei 1 beginnende Kapitel- und Seitennummerierung und wird unabhängig aktualisiert.

Teil	Titel	Inhalt
Teil 1	Einführung und Installation	Allgemeine Hinweise, Lizenzen und Add-ons Installation und Programmstart Benutzeroberfläche
Teil 2	Arbeiten mit *ibaAnalyzer*	Arbeiten mit Messdatei und Analyse, Darstellungsfunktionen, Makrokonfiguration, Filterdesign, Voreinstellungen, Drucken, Export, Schnittstellen zu *ibaHD-Server*, *ibaCapture* und Reportgenerator
Teil 3	Ausdruckseditor	Verzeichnis aller Berechnungsfunktionen im Ausdruckseditor, inkl. Erklärung
Teil 4	Datenbank-Schnittstelle	Arbeiten mit Daten aus Datenbanken, Verbindung zur Datenbank, Schreiben von iba-Messdaten in Datenbanken, Extraktion der Daten aus der Datenbank und Analyse der Daten
Teil 5	Schnittstelle für Datei-Extraktion	Funktionen und Einstellungen zur Extraktion von Daten aus iba-Messdateien in externe Dateiformate
Teil 6	Anwendungsbeispiele	*In Vorbereitung*

2 Funktion und Bedienung

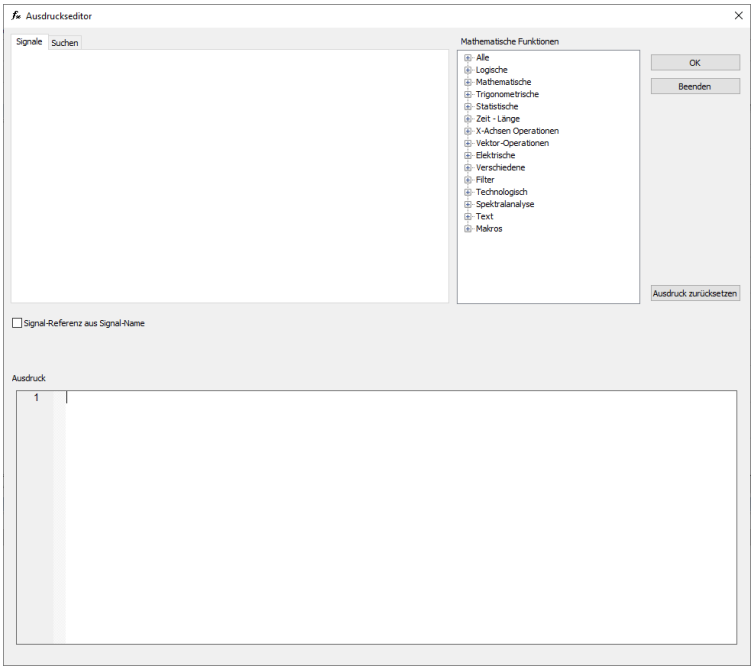
Der Ausdruckseditor ist ein Hilfsmittel zur Eingabe von (mathematischen) Formeln oder Ausdrücken, die in den folgenden Abschnitten ausführlich beschrieben sind. Prinzipiell können diese Ausdrücke in die Zeilen der Signaltabelle, im Register Signaldefinitionen, von Hand eingegeben werden.

Um diese Eingaben zu erleichtern und auch um eine ausführliche Liste der möglichen Operationen und ihrer Syntax zur Verfügung zu stellen, gibt es den Ausdruckseditor, der in jeder Zeile, in der ein Signal eingegeben werden kann, verfügbar ist. Starten Sie den Ausdruckseditor mit einem Klick auf das Symbol in der Signalliste.

Hinweis

Die Symboltaste f_x in der Symbolleiste öffnet nicht den Ausdruckseditor, sondern den Dialog für die logischen Signaldefinitionen, siehe *Logische Signaldefinitionen* in Teil 2 des Handbuchs.

2.1 Aufbau



Der Ausdruckseditor besteht aus drei Bereichen.

Im linken Teil befindet sich ein Signalbaum, der dem im Signalbaumfenster sehr ähnlich ist. Im Unterschied zum Signalbaumfenster sind hier jedoch neben den Originalsignalen auch alle Aus-

drücke zu sehen, die bereits mit dem Ausdruckseditor erstellt wurden. Aus diesem Signalbaum wählt man die gewünschten Signale oder Ausdrücke aus, mit denen gerechnet werden soll.

Im rechten Teil des Dialogfensters befindet sich in einer Funktionsbaum-Darstellung die Sammlung der verfügbaren mathematischen Operationen und anderer Funktionen, geordnet nach Themen.

Unterhalb dieser beiden Auswahlfelder befindet sich die Befehlseingabe, in die der gewünschte Ausdruck mehrzeilig eingetragen wird. Oberhalb erscheint in dem grauen Bereich ein kurzer Hinweis zur Syntax einer Operation, wenn diese im Funktionsbaum markiert ist, bzw. ein Tooltip, wenn die Funktion in der Befehlseingabe markiert ist.

Die Schaltfläche <Ausdruck zurücksetzen> löscht alle Eintragungen aus der Befehlszeile.

Das Auswahlfeld "Signalreferenz aus Signalnamen" können Sie aktivieren, wenn Sie in den Ausdrücken die Signalnamen anstelle der üblichen Signalbezeichnungen, bestehend aus [Modulnummer:Signalnummer], verwenden wollen.

Hinweis

Bei Verwendung der Signalnamen als Signalreferenz muss sichergestellt sein, dass die Signalnamen eindeutig sind.

2.2 Syntax von Ausdrücken und Platzhalter

Signale werden in *ibaAnalyzer*-Ausdrücken durch eckige Klammern '[' und ']' um die Signalkennung gekennzeichnet. Jedes in *ibaAnalyzer* verfügbare Signal hat eine Kennung, die in der Regel aus einer Modul- und einer Signalnummer besteht. Die Nummern werden durch einen Doppelpunkt ':' für analoge und Textsignale oder einen Punkt '.' für digitale Signale getrennt. In einigen Fällen ist auch eine Untersignalnummer vorhanden, die wiederum durch das gleiche Trennzeichen (je nach Signaltyp) getrennt ist. Bei Dateien, die von *ibaAnalyzer* extrahiert wurden, können die Ergebnisse der Berechnungen über ihren Namen referenziert werden.

Beispiele:

- [0:0] - Analog- oder Textsignal, Modul 0, Nummer 0
- [1.2] - digitales Signal, Modul 1, Nummer 2
- [3:2:1] - analoges Signal, Modul 3, Nummer 2, Untersignal Nummer 1
- [Ausdruck] - Ausdruck mit dem Namen 'Ausdruck'

Wenn mehrere Dateien parallel geöffnet werden, ist der Dateiindex ebenfalls Teil der Kennung. Die erste Datei hat den Index 0, dieser kann aber auch weggelassen werden.

Beispiele:

- [0_0:0] - Signal aus der ersten Datei, Modul 0, Nummer 0; gleich [0:0]
- [2_1:0] - Signal aus der Datei mit dem Index 2 (dritte Datei), Modul 1, Nummer 0

Mit Hilfe von Platzhaltern können mehrere Signale auf einmal adressiert werden. Das Ergebnis ist ein Vektor. Jeder Teil der Kennung (Dateiindex, Modulnummer, Signalnummer, Untersignalnummer) kann durch einen Bereich (z. B. 3-6) oder einen Stern '*' (für alle verfügbaren Nummern) ersetzt werden.

Beispiele:

- [*_0:0] - liefert das Signal [0:0] für alle parallel geöffneten Dateien
- [0:3-5] - liefert die Signale [0:3], [0:4], und [0:5]
- [0-2_3:4] - liefert das Signal [3:4] der ersten 3 parallel geöffneten Dateien
- [*.0] - liefert das erste digitale Signal (mit der Nummer 0) von allen verfügbaren Modulen
- [*_*:*] - liefert alle Analog- und Textsignale aller geöffneten Dateien

2.3 Funktionsweise des Ausdruckseditors

Der Ausdruckseditor ermöglicht es, sowohl die Operationen als auch die Operanden, also die Signale und Ausdrücke, per Doppelklick oder Drag & Drop in die Befehlszeile zu übernehmen. Diese Vorgehensweise wird empfohlen, um Schreibfehler zu vermeiden und schneller arbeiten zu können.

Grundsätzlich gilt: An die Stelle, an der der Cursor in der Eingabezeile steht, wird die Operation oder der Operand eingefügt, auf die im Funktionsbaum oder im Signalbaum ein Doppelklick ausgeführt wird.

Um die Übersicht über komplexe Ausdrücke nicht zu verlieren, kann mit der Tastenkombination <Strg>+<B> zwischen zusammengehörigen Klammerpaaren hin- und hergesprungen werden.

Hinweis

Die Funktion, Signale und Ausdrücke per Doppelklick in die Befehlszeile zu übernehmen, steht nur im Ausdruckseditor zur Verfügung und kann nicht im normalen Signalbaum im Signalbaumfenster verwendet werden.

Erfahrenen Anwendern steht sowohl in der Signaltabelle, als auch in der Befehlszeile des Ausdruckseditors die Eingabehilfe *Intellisense* zur Verfügung. Bei manuellen Eingaben öffnet sich automatisch ein Fenster mit möglichen Vervollständigungen Ihrer Eingabe. Dies beinhaltet sowohl Funktionen und ihre Parameter, als auch Signale oder virtuelle Ausdrücke, die in der Messdatei vorhanden sind.

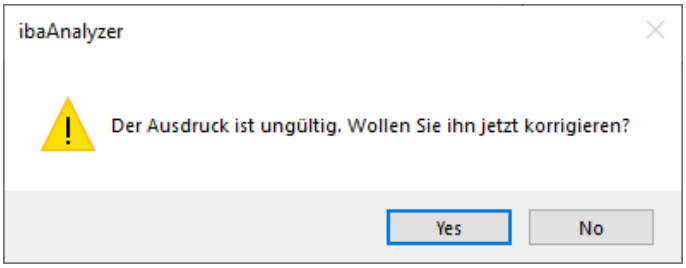
Mit der Cursor-Steuerung können Sie einen Vorschlag in dem Intellisense-Fenster auswählen und per <Return> übernehmen. Wenn Sie weiterschreiben, werden die dann passenden Vervollständigungen angeboten, bis der Befehl vollständig ist. Wird diese Funktion im Ausdruckseditor genutzt, werden auch automatisch alle notwendigen Klammern eingefügt.

2.4 Diagnose / Syntaxfehler-Erkennung

Wenn der Ausdruckseditor mit OK verlassen wird, erscheint in der entsprechenden Zeile der Signaldefinitionen der soeben erstellte Ausdruck.

Als Signalname wird automatisch der Ausdruck selbst eingetragen, er kann aber einfach von Hand mit einem Klartext überschrieben werden. Bei komplexeren Ausdrücken, die miteinander kaskadiert werden, ist es zu empfehlen, möglichst kurze und eindeutige Bezeichnungen zu wählen, um den Ausdruck übersichtlich zu halten.

Im Fall einer fehlerhaften Eingabe mit dem Ausdruckseditor erscheint eine Warnung, die es ermöglicht den Ausdruck zu korrigieren. Wird <Ja> geklickt, springt der Cursor automatisch zur Stelle an der der Fehler vermutet wird.

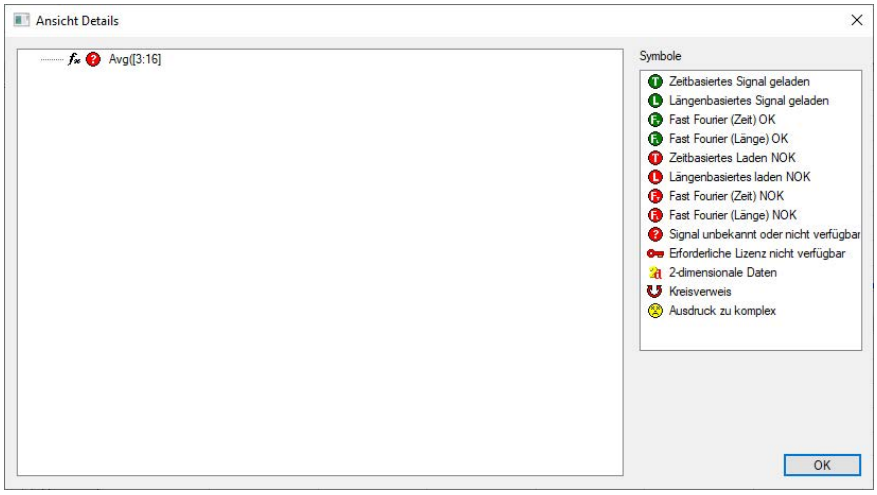


Tipp

Die Funktion, nach möglichen Fehlern zu suchen, kann auch manuell mit der Tastenkombination <Strg>+<E> gestartet werden.

Wird die Fehlermeldung ignoriert, oder der Fehler bei der Eingabe in der Signaldefinitionszeile gemacht, zeigt *ibaAnalyzer* dies mit einer roten Farbe an.

Auf diese Weise können formale oder syntaktische Fehler erkannt werden, mit denen eine Berechnung nicht möglich ist. Um genauere Informationen über die Fehlerursache zu erhalten, kann die Diagnose mit einem Mausklick auf das gelbe Fragezeichensymbol in der betreffenden Signaldefinitionszeile geöffnet werden.



3 Logische Funktionen

3.1 Vergleichsfunktionen

```
z. B. ('Expression1') < ('Expression2')
```

>	Größer
>=	Größer / gleich
<	Kleiner
<=	Kleiner / gleich
<>	Ungleich
=	Gleich

Beschreibung

Mit den Vergleichsoperationen >, >=, <, <=, <> und = können die Werte zweier Ausdrücke (Operanden) miteinander verglichen werden. Die Operationen liefern als Ergebnis jeweils den booleschen Wert TRUE oder FALSE. Als Operanden können Originalsignale, berechnete Ausdrücke oder einfach konstante Werte eingetragen werden. Das Ergebnis kann als neuer Ausdruck wie ein Signal dargestellt und ausgewertet werden. Somit lassen sich leicht binäre Signale bilden, die ihrerseits wieder als Bedingungen für andere Funktionen verwendet werden können.

Hinweis

Wenn die Kreuzung zweier Kurven zwischen zwei Messpunkten liegt, dann wird das Ergebnis der Vergleichsoperation der letzten beiden Messwerte bis zum nächsten Messpunkt gehalten. D.h. ein Wechsel von TRUE nach FALSE (oder umgekehrt) wird stets im Raster der Messpunkte eingetragen. Die Verbindungslinie zwischen zwei Messpunkten bei der Darstellung von Analogwerten ist nur eine grafische Näherung.

3.2 Boolesche Funktionen

```
z. B. ('Expression1') AND ('Expression2')
```

AND	Logisches UND
OR	Logisches ODER
XOR	Logisches Exklusiv-ODER
NOT	Logisches NOT, Negation

Beschreibung

Mit den booleschen Funktionen AND, OR, NOT und XOR können binäre Ausdrücke, wie z. B. Digitalsignale, miteinander verknüpft werden. Entsprechend den Regeln der booleschen Logik liefern die Funktionen als Ergebnis jeweils den Wert TRUE oder FALSE. Als Parameter können Digitalsignale, berechnete (binäre) Ausdrücke oder die Zahlenwerte 0 bzw. 1 eingetragen werden.

Das Ergebnis kann als neuer Ausdruck wie ein Signal dargestellt und ausgewertet werden. Somit lassen sich leicht binäre Signale bilden, die ihrerseits wieder als Bedingungen für andere Funktionen verwendet werden können.

Logische Funktionen, Wahrheitstabelle:

A	B	A AND B	A OR B	A XOR B	NOT A
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	
1	1	1	1	0	

3.3 Boolesche Funktionen (bitweise)

```
z. B. ('Expression1') bw_NOT  ('Expression2')
```

bw_AND	Bitweise UND
bw_OR	Bitweise ODER
bw_XOR	Bitweise Exklusiv-ODER
bw_NOT	Bitweise NOT

Beschreibung

Diese Funktionen dienen dem bitweisen Verknüpfen von zwei Analogwerten auf Basis der booleschen Algebra. Die Funktionen liefern als Ergebnissignal ein 32Bit-Integer. Als Argumente werden 32Bit-Integers erwartet.

Sind die Argumente keine Integer, dann wird zunächst der Dezimalteil verworfen bevor die Operation durchgeführt wird. Sind die Argumente zu groß, so dass ihr Absolutwert nicht in ein 32Bit-Integer passt, wird die Operation nur auf den 32 niederwertigeren Bits ausgeführt.

Bei der Verknüpfung zweier Analogwerte mit einer bw-Funktion werden die einzelnen Bits beider Werte logisch verknüpft. Das Ergebnis ist wieder ein gleichartiger Analogwert mit einem Bitmuster entsprechend der logischen Verknüpfung.

Beispiel

Für 2 Analogwerte W1 = 15 und W2 = 2 ergeben sich folgende Ergebnisse:

	Dez.-Wert	Bits	Hex	Ergebniswert
Ausgangswert W1	15	...1111	0x0000000F	
Ausgangswert W2	2	...0010	0x00000002	
W1 bw_AND W2		...0010	0x00000002	2
W1 bw_OR W2		...1111	0x0000000F	15
W1 bw_XOR W2		...1101	0x0000000D	13
bw_NOT (W1)		...0000	0xFFFFFFF0	-16

3.4 Verzweigungen

3.4.1 If

```
If('Condition','IF-True','IF-False')
```

Argumente

'Condition'	Bedingung als Operation mit den booleschen Ergebnissen TRUE oder FALSE
'IF-True'	Operation wird ausgeführt, wenn 'Condition' gleich TRUE
'IF-False'	Operation wird ausgeführt, wenn 'Condition' gleich FALSE

Beschreibung

Die If-Funktion dient zur bedingten Ausführung weiterer Berechnungen. Abhängig vom booleschen Ergebnis einer Bedingung ('Condition'), die selbst eine Operation sein kann, wird beim Ergebnis TRUE die Operation 'IF-True' ausgeführt, beim Ergebnis FALSE entsprechend die Operation 'IF-False'.

Damit lassen sich unterschiedliche Berechnungen prozessgesteuert durchführen. Die Funktion kann natürlich auch verschachtelt genutzt werden, um weitere Verzweigungen zu realisieren.

Tipp

Wenn für 'Condition' nur eine Größe eingetragen wird, wird als Bedingung abgefragt, ob die Größe größer als (TRUE) oder kleiner als (FALSE) 0,5 ist.

3.4.2 Switch

`Switch('Selector_Expression',`	`'Case_1_Expression','Value_1_Expression',`
	`'Case_2_Expression','Value_2_Expression',`
	`...`
	`'Case_n_Expression','Value_n_Expression',`
	`'Default_Value_Expr')`

Argumente

'Selector_Expression'	Ausdruck, welcher auf verschiedene Bedingungen überprüft wird
'Case_n_Expression'	Ausdruck, welcher mit 'Selector_Expression' verglichen wird
'Value_n_Expression'	Ergebnis, falls 'Selector_Expression' und 'Case_n_Expression' übereinstimmen
'Default_Value _Expr'	Ergebnis, falls keiner der 'Case_n_Expression' mit 'Selector_Expression' übereinstimmt

Beschreibung

Diese Anweisung vergleicht eine eingehende 'Selector_Expression' mit beliebig vielen 'Case_n_Expression', angelehnt an das SQL Statement CASE. Es werden mindestens 3 Argumente benötigt. Bei einer geraden Anzahl von Argumenten wird automatisch das letzte als 'Default_Value _Expr' interpretiert, welches herangezogen wird, wenn keine der 'Case_n_Expression' zur 'Selector_Expression' passt.

Falls 'Selector_Expression' und 'Case_n_Expression' zusammenpassen, wird die dazugehörige 'Value_n_Expression' zurückgegeben. Passen mehrere 'Case_n_Expression' zum Eingangssignal, wird automatisch die erste ausgewählt.

Als 'Selector_Expression' sind folgende Signale zulässig:

- Eine numerische Konstante
- Eine Text-Konstante
- Ein äquidistant oder nicht-äquidistant gesampelter Kanal
- Ein Text-Kanal

Grundsätzlich müssen die Typen der Vergleichswerte zusammenpassen, ansonsten wird der entsprechende Fall nicht ausgewählt.

3.5 Flankenerkennung

3.5.1 OneShot

```
OneShot('Expression')
```

Beschreibung

Diese Funktion liefert das Ergebnis TRUE, wenn der aktuelle Messwert von 'Expression' ungleich dem vorigen ist. Sie liefert das Ergebnis FALSE, wenn der aktuelle Messwert gleich dem vorigen ist.

Tipp

Die Funktion arbeitet auch mit nicht-äquidistanten Messwerten.

3.5.2 SetReset

```
SetReset('Set','Reset','SetDominant=1')
```

Argumente

'Set'	Positive Flanke setzt Funktion auf TRUE	
'Reset'	Positive Flanke setzt Funktion auf FALSE	
'SetDominant'	Optionaler Parameter (Voreinstellung = 1), der steuert, welches Eingangsargument dominant ist, wenn beide Argumente gleichzeitig eine positive Flanke erhalten.	
	'SetDominant' = 1	Set hat Vorrang gegenüber Reset
	'SetDominant' = 0	Reset hat Vorrang gegenüber Set

Beschreibung

Diese Funktion wird verwendet, um ein digitales Ergebnis (TRUE/FALSE) mithilfe von positiven Flanken (Übergang von 0 zu 1) der Argumente 'Set' und 'Reset' zu steuern.

Eine positive Flanke von Operand 'Set' gibt als Ergebnis ein statisches TRUE wieder. Eine positive Flanke von Operand 'Reset' setzt das Ergebnis auf FALSE zurück. Das Argument 'SetDominant' ist optional und bestimmt die Dominanz von 'Set' bzw. 'Reset'.

Tipp

Bei einem analogen Signal entspricht das Überschreiten des Wertes 0,5 einer positiven Flanke.

3.6 Timer-Funktionen (IEC 61131-3)

TOF

```
TOF('in','pt')
```

Beschreibung
Ausschaltverzögerung. Der Ausgang wird 'pt' Sekunden nach dem Ausschalten des Eingangs 'in' ausgeschaltet.

TON

```
TON('in','pt')
```

Beschreibung
Einschaltverzögerung. Der Ausgang wird 'pt' Sekunden nach dem Einschalten des Eingangs 'in' eingeschaltet.

TP

```
TP('in','pt')
```

Beschreibung
Impuls-Funktion. Der Ausgang wird für 'pt' Sekunden nach steigender Flanke an Eingang 'in' eingeschaltet.

Tipp

Eine weitere steigende Flanke während des Ausgangs-Impulses verlängert die Schaltung nicht und startet den Impuls nicht neu.

3.7 IsData / Coalesce

IsData

```
IsData('Expression','End')
```

Argumente

'Expression'	Eingangssignal
'End'	Länge des Ausgangssignals

Beschreibung

Das Ergebnis dieser Operation ist WAHR, wenn für 'Expression' Messwerte vorhanden sind. Das Ergebnis ist FALSCH, wenn Messwerte fehlen bzw. Signale leer sind. Diese Funktion kann beispielsweise als Bedingung für andere Berechnungen verwendet werden.

Optional kann der Parameter 'End‘ angegeben werden. Mit diesem Parameter kann das Ergebnissignal der Funktion verkürzt oder verlängert werden, so dass es mit anderen Signalen übereinstimmt und für weitere Verknüpfungen genutzt werden kann. Geben Sie den Parameter 'End‘ nicht an, dann entspricht die Länge des Ergebnissignals der des Eingangssignals (inkl. ungültiger Samples).

Coalesce

```
Coalesce('Candidate1','Candidate2',...)
```

Beschreibung

Inspiriert von der entsprechenden SQL-Abfrage gibt die Funktion *Coalesce* das erste ihrer Argumente zurück, welches Daten enthält.

Dies kann beispielsweise genutzt werden, um eine Absicherung gegen fehlende Signale in einer DAT-Datei zu schaffen.

4 Mathematische Funktionen

4.1 Grundrechenarten

4.1.1 Grundrechenarten +, -, *, /

```
z. B. ('Expression1') + ('Expression2')
```

Beschreibung
Alle Signale und Ausdrücke können mit den Grundrechenarten (Addition, Subtraktion, Multiplikation und Division) verarbeitet werden. Wenn als Operanden digitale Signale oder Ausdrücke mit den Grundrechenarten verwendet werden, dann übersetzt *ibaAnalyzer* die Werte TRUE mit 1.0 und FALSE mit 0.0. Das Ergebnis einer Grundrechenoperation ist stets ein analoger Ausdruck.

4.1.2 Abs

```
Abs('Expression')
```

Beschreibung
Die Absolutfunktion gibt den Absolutwert (= |Betrag|) von 'Expression' zurück.

Tipp

Interpolierte Werte bei einem Vorzeichenwechsel zwischen zwei Messpunkten können sich im Betrag unterscheiden.

4.1.3 Mod

```
Mod('Dividend','Divider')
```

Beschreibung
Diese Funktion liefert als Ergebnis den Modulo von 'Dividend' und 'Divider'. Die Funktion verwendet intern die C-Funktion fmod, die es gestattet, Gleitkommawerte für 'Dividend' und 'Divider' zu verwenden.

Der Modulo r ist der Rest der Division Dividend / Divider, so dass umgekehrt gilt:

Dividend = Divider * x + r, wobei x eine ganze Zahl (Integer) ist.

Modulo r hat stets das gleiche Vorzeichen wie 'Dividend' und der Absolutwert von r ist stets kleiner als der Absolutwert von 'Divider'.

Wenn 'Dividend' < 'Divider' ist, dann liefert die Funktion den Wert 'Dividend' als Ergebnis. Der Rest kann in der mathematischen Sprache auch als "Dividend modulo Divider" bezeichnet werden.

4.1.4 Ceiling / Floor / Round

Ceiling

```
Ceiling('Expression')
```

Beschreibung

Diese Funktion gibt den kleinsten Integer-Wert wieder, der größer oder gleich 'Expression' ist.

Floor

```
Floor('Expression')
```

Beschreibung

Diese Funktion gibt den größten Integer-Wert wieder, der kleiner oder gleich 'Expression' ist.

Round

```
Round('Expression')
```

Beschreibung

Diese Funktion rundet 'Expression' auf die nächste Ganzzahl (Integerwert) auf oder ab.

4.2 Integral- und Differenzialrechnung

4.2.1 Int

```
Int('Expression','Reset')
```

Argumente

'Expression'	Messwert	
'Reset'	Optionaler digitaler Parameter, der zum Rücksetzen des Integrals bzw. Unterdrücken des Integrationsvorgangs verwendet werden kann. 'Reset' kann selbst auch ein Ausdruck sein.	
	'Reset' > 0	Integral wird zurückgesetzt.
	'Reset'= 0	Integration freigegeben (Voreinstellung)

Beschreibung
Diese Funktion liefert als Ergebnis das Integral von 'Expression' zurück. Der Parameter 'Reset' kann zum Rücksetzen des Integrals bzw. Unterdrücken des Integrationsvorgangs verwendet werden, z. B. um bei periodischen oder reversierenden Vorgängen dasselbe Signal mehrfach zu integrieren. 'Reset' kann selbst auch ein Ausdruck sein.

4.2.2 Diff / Dif

```
Diff('Expression','dy'=0)
```

Beschreibung
Diese Funktion liefert als Ergebnis die Ableitung (oder das Differential) von 'Expression' zurück. Setzt man den optionalen Parameter 'dy' auf True(), wird statt dem Differential lediglich die Differenz zwischen den Messwerten berechnet.

Beispiel
Ist 'Expression' ein Längenmesssignal, so kann mit der *Diff*-Funktion daraus der Geschwindigkeitsverlauf ermittelt werden.

4.3 Potenzen und Wurzeln

4.3.1 Pow

```
Pow('Expression1','Expression2')
```

Argumente

'Expression1'	Basis
'Expression2'	Exponent

Beschreibung
Diese Funktion potenziert 'Expression1' (Basis) mit 'Expression2' (Exponent).

Beispiel
Berechnung einiger wichtiger Potenzen

(2)^0 = Pow(2, 0) = 1

(2)^-2 = Pow(2, -2) = 0,25

(-2)^2 = Pow(-2, 2) = 4

(10)^(lg 2) = Pow(10, lg 2) = 2

(0)-1 = Pow(0, -1) = +∞ (unendlich)

Tipp

Die Potenzierung von 0 mit -1 gibt zwar keine Fehlermeldung, allerdings auch kein Ergebnis.

4.3.2 Sqrt

```
Sqrt('Expression')
```

Beschreibung
Diese Funktion liefert als Ergebnis die Quadratwurzel von 'Expression'.

Hinweis

Negative Werte für 'Expression' geben zwar keine Fehlermeldung, allerdings auch kein Ergebnis.

4.4 e-Funktion und Logarithmen

4.4.1 Exp

```
Exp('Expression')
```

Beschreibung
Diese Funktion berechnet den Ausdruck (e)$^{'Expression'}$

4.4.2 Log

```
Log('Expression')
```

Beschreibung
Diese Funktion liefert als Ergebnis den natürlichen Logarithmus von 'Expression'.

Hinweis

Negative Werte für 'Expression' geben zwar keine Fehlermeldung, allerdings auch kein Ergebnis.

4.4.3 Log10

```
Log10('Expression')
```

Beschreibung
Diese Funktion liefert als Ergebnis den dekadischen Logarithmus von 'Expression'.

Hinweis

Negative Werte für 'Expression' geben zwar keine Fehlermeldung, allerdings auch kein Ergebnis.

4.5 PI

```
Pi()
```

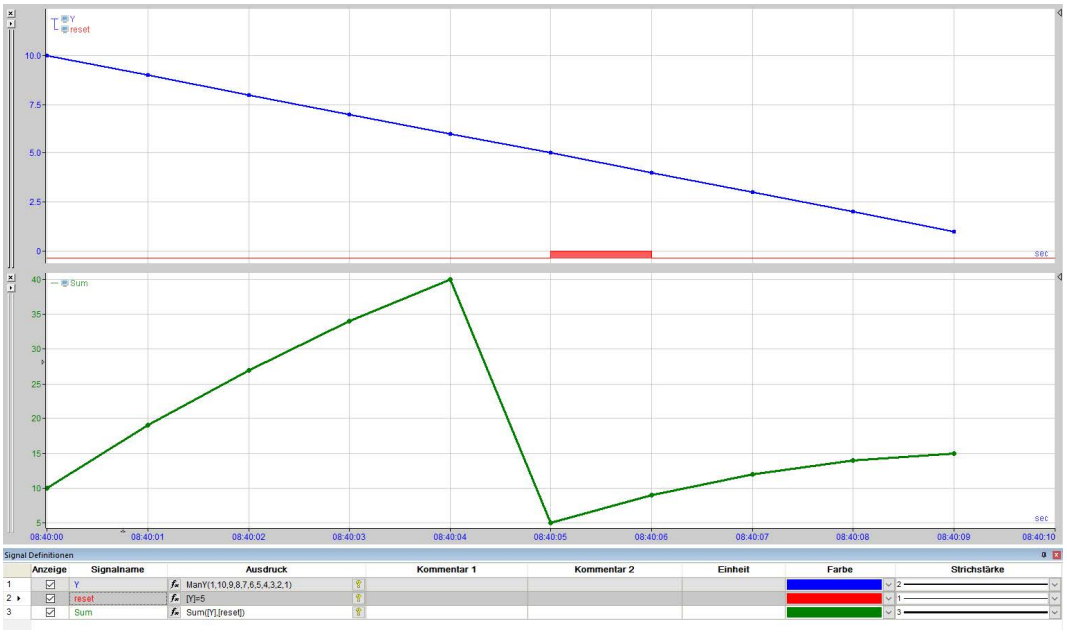
Beschreibung
Die Zahl Pi ist als Konstante (π = 3.1415927...) für diverse Berechnungen im System hinterlegt. Mit dieser Funktion fügen Sie die Zahl π in Ihre Berechnung ein.

4.6 Sum

```
Sum('Expression','Reset'=0)
```

Beschreibung
Diese Operation summiert alle Signalpunkte einer Funktion Punkt für Punkt. Wird die Summenbildung durch einen Reset-Wert unterbrochen, beginnt anschließend die Summenbildung erneut.

Beispiel
Die Summenbildung beginnt mit dem Signalpunkt 10 + 9 + 8 + …6. Hier erfolgt durch 'Reset' = TRUE eine Unterbrechung und die Funktion wird auf Null gesetzt. Anschließend beginnt die Summenbildung erneut.



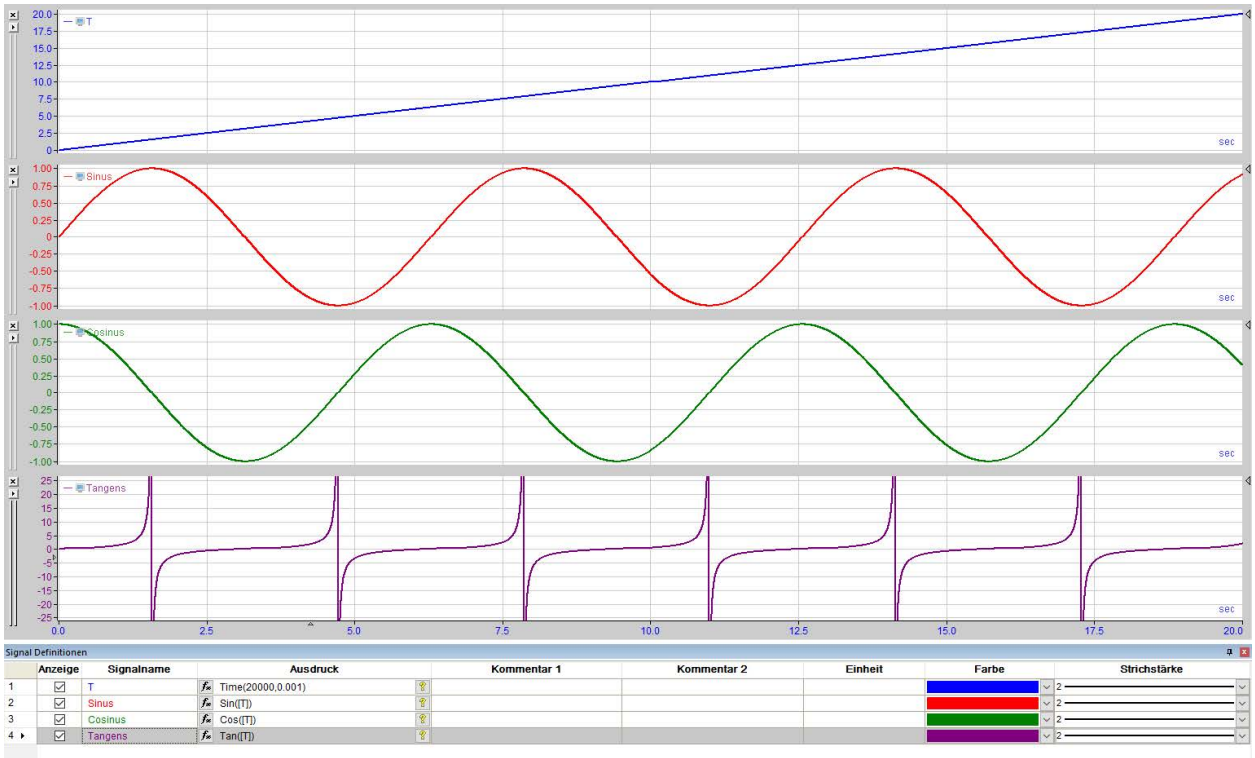
4.7 Trigonometrische Funktionen

```
z. B. Sin('Expression')
```

Sin('Expression')	Diese Funktion liefert als Ergebnis den Sinus von 'Expression' in rad.
Cos('Expression')	Diese Funktion liefert als Ergebnis den Cosinus von 'Expression' in rad.
Tan('Expression')	Diese Funktion liefert als Ergebnis den Tangens von 'Expression' in rad.
Asin('Expression')	Diese Funktion liefert als Ergebnis den Arkussinus von 'Expression' in rad.
Acos('Expression')	Diese Funktion liefert als Ergebnis den Arkuscosinus von 'Expression' in rad.
Atan('Expression')	Diese Funktion liefert als Ergebnis den Arkustangens von 'Expression' in rad.
Atan2('X', 'Y')	Diese Funktion liefert als Ergebnis den Arcustangens von 'Y'/'X'.

Beschreibung
Für die verschiedensten Berechnungen, in denen trigonometrische Funktionen benötigt werden, z. B. Leistungsermittlung in Wechselstromsystemen, stehen die Standardfunktionen und die entsprechenden Umkehrfunktionen zur Verfügung.

Beispiel
Darstellung der trigonometrischen Funktionen:

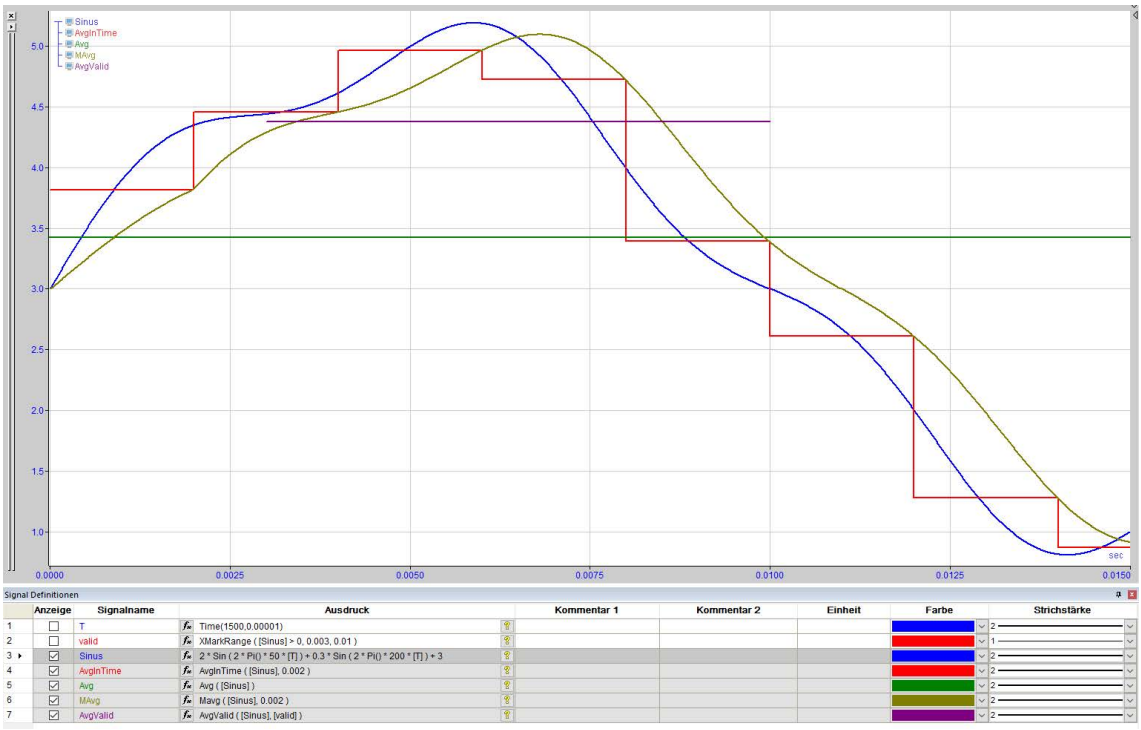


5 Statistische Funktionen

ibaAnalyzer unterstützt die Berechnung verschiedener statistischer Funktionen. Für alle Funktionen sind im Normalfall vier verschiedene Varianten verfügbar:

- Die Standardfunktion berechnet die entsprechende Größe immer über das gesamte Signal
- Das Suffix *InTime* kennzeichnet, dass die entsprechende Größe über Intervalle einer angegebenen Länge gebildet wird
- Das Suffix *Valid* wird benutzt, um die entsprechende Größe über Intervalle zu berechnen, die mit einem Binärsignal gekennzeichnet werden
- Das Präfix *M* kennzeichnet, dass die entsprechende Größe über gleitende Intervalle einer angegebenen Breite gebildet wird

5.1 Mittelwert (Avg)



Avg

```
Avg('Expression')
```

Beschreibung

Diese Funktion liefert als Ergebnis den Mittelwert von 'Expression'. Er wird als konstanter Wert (horizontale Linie) im Signalstreifen angezeigt.

AvgInTime

```
AvgInTime('Expression','Interval')
```

Argumente

'Expression'	Messwert, für den der Mittelwert gebildet wird
'Interval'	Angabe der Intervalllänge

Beschreibung

Diese Funktion liefert als Ergebnis den Mittelwert je Zeitabschnitt der Länge 'Interval' von 'Expression'.

MAvg

```
MAvg('Expression','Interval')
```

Argumente

'Expression'	Messwert, für den der Mittelwert gebildet wird
'Interval'	Angabe der Länge des Intervalls, über das der Mittelwert gebildet wird

Beschreibung

Diese Funktion liefert als Ergebnis den gleitenden arithmetischen Mittelwert von 'Expression' berechnet über ein 'Interval' in Sekunden.

Tipp

Es können auch Signale oder Ausdrücke mit diesen Funktionen verarbeitet werden, die nicht zeitbasiert sind, d. h. die die Basis Länge, Frequenz oder 1/Länge haben. Anstelle von Sekunden ist dann der X-Achsen-Abschnitt entsprechend der Basis in m, Hz oder 1/m anzugeben.

AvgValid

```
AvgValid('Expression','Valid')
```

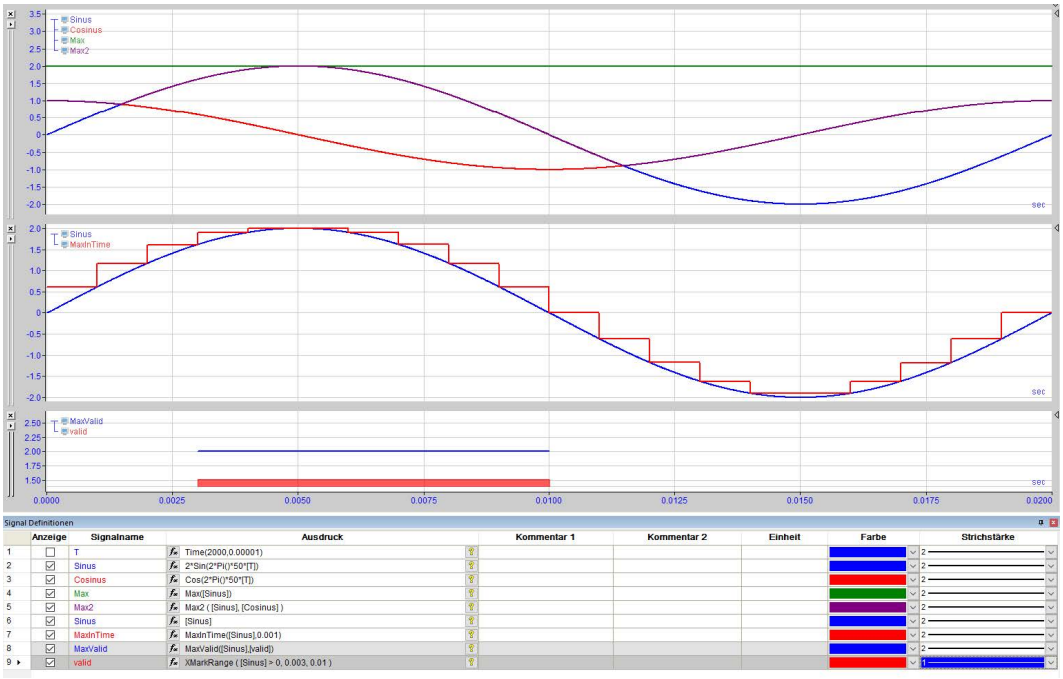
Argumente

'Expression'	Messwert, für den der Mittelwert gebildet wird
'Valid'	Kontrollsignal

Beschreibung

Diese Operation liefert als Ergebnis den Mittelwert von 'Expression' für das Intervall (Zeit oder Länge) zurück, in dem ein zugehöriges Kontrollsignal WAHR ist.

5.2 Maxima (Max)



Max

```
Max('Expression')
```

Beschreibung

Diese Funktion liefert als Ergebnis den Maximalwert von 'Expression'. Es wird als konstanter Wert (horizontale Linie) im Signalstreifen angezeigt.

Max2

```
Max2('Expression1','Expression2')
```

Beschreibung

Diese Funktion liefert als Ergebnis das Maximum von zwei Signalen 'Expression1' und 'Expression2'. Die beiden Signale werden Messwert für Messwert verglichen und der jeweils größere Wert wird als Ergebnis übergeben.

MaxInTime

```
MaxInTime('Expression','Interval')
```

Argumente

'Expression'	Messwert, für den das Maximum gebildet wird
'Interval'	Länge des Intervalls, über das das Maximum berechnet werden soll.

Beschreibung

Diese Funktion liefert als Ergebnis das Maximum von 'Expression' innerhalb jedes Intervalls der Länge 'Interval'. Es können Signale und Ausdrücke verarbeitet werden, die zeitbasiert ('Interval' in Sekunden) oder längenbasiert ('Interval' in Metern) sind.

MaxValid

```
MaxValid('Expression','Valid')
```

Argumente

'Expression'	Messwert, für den das Maximum gebildet wird
'Valid'	Kontrollsignal

Beschreibung

Diese Operation liefert als Ergebnis das Maximum von 'Expression' für das Intervall (Zeit oder Länge) zurück, in dem ein zugehöriges Kontrollsignal WAHR ist.

MMax

```
MMax('Expression','Interval')
```

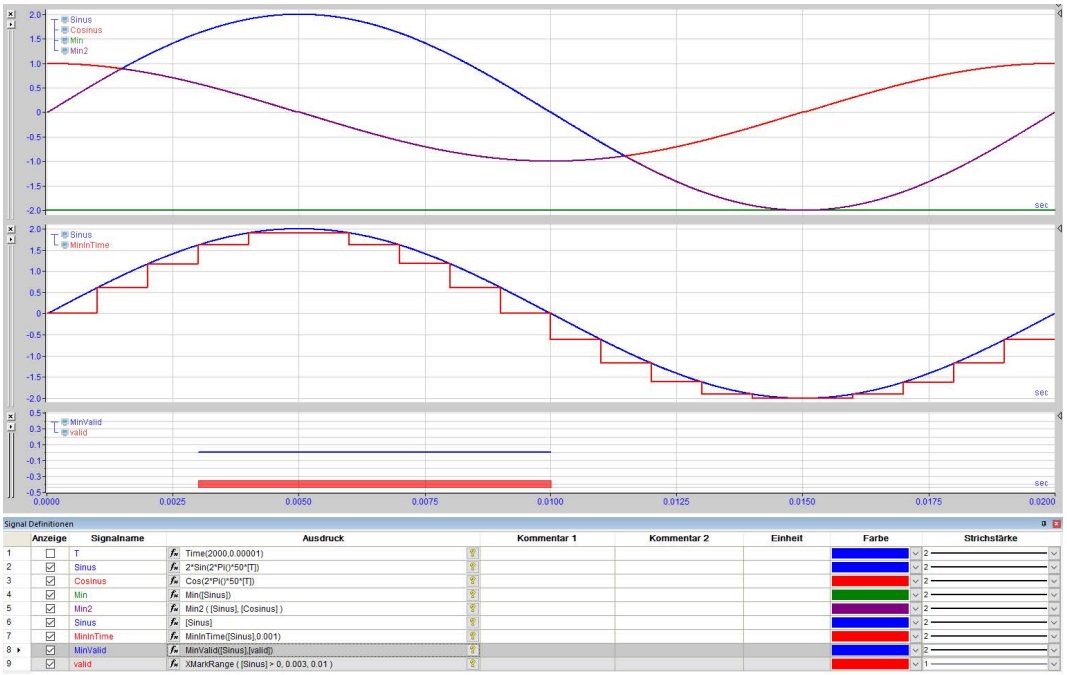
Argumente

'Expression'	Messwert, für den das Maximum gebildet wird
'Interval'	Länge des Intervalls, über das das Maximum berechnet werden soll

Beschreibung

Diese Funktion liefert das Maximum von 'Expression' innerhalb eines gleitenden X-Achsenintervalls der Länge 'Interval', fortschreitend um jeweils einen Messpunkt.

5.3 Minima (Min)



Min

```
Min('Expression')
```

Beschreibung

Diese Funktion liefert als Ergebnis den Minimalwert des Signals 'Expression'. Es wird als konstanter Wert (horizontale Linie) im Signalstreifen angezeigt.

Min2

```
Min2('Expression1','Expression2')
```

Beschreibung

Diese Funktion liefert als Ergebnis das Minimum von zwei Signalen 'Expression1' und 'Expression2'. Die beiden Signale werden Messwert für Messwert verglichen und der jeweils kleinere Wert wird als Ergebnis übergeben.

MinInTime

```
MinInTime('Expression','Interval')
```

Argumente

'Expression'	Messwert, für den das Minimum gebildet wird
'Interval'	Länge des Intervalls, über das das Minimum berechnet werden soll.

Beschreibung

Diese Funktion liefert als Ergebnis das Minimum von 'Expression' innerhalb jedes Intervalls der Länge 'Interval'. Es können Signale und Ausdrücke verarbeitet werden, die zeitbasiert ('Interval' in Sekunden) oder längenbasiert ('Interval' in Metern) sind.

MinValid

```
MinValid('Expression','Valid')
```

Argumente

'Expression'	Messwert, für den das Minimum gebildet wird
'Valid'	Kontrollsignal

Beschreibung

Diese Operation liefert als Ergebnis das Minimum von 'Expression' für das Intervall (Zeit oder Länge) zurück, in dem ein zugehöriges Kontrollsignal WAHR ist.

MMin

```
MMin('Expression','Interval')
```

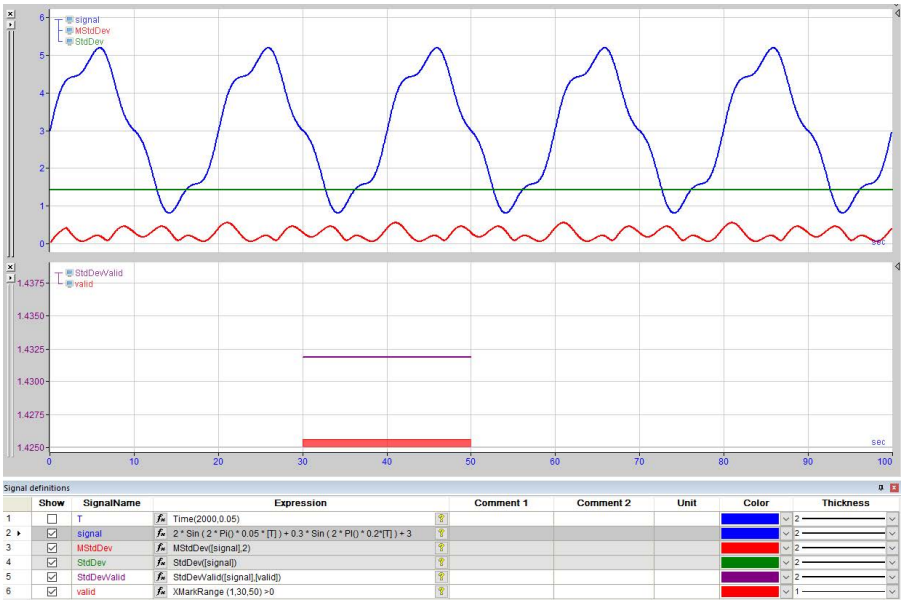
Argumente

'Expression'	Messwert, für den das Minimum gebildet wird
'Interval'	Länge des Intervalls, über das das Minimum berechnet werden soll

Beschreibung

Diese Funktion liefert das Minimum von 'Expression' innerhalb eines gleitenden X-Achsenintervalls der Länge 'Interval', fortschreitend um jeweils einen Messpunkt.

5.4 Standardabweichung (StdDev)



StdDev

```
StdDev('Expression')
```

Beschreibung

Diese Funktion liefert als Ergebnis die Standardabweichung von 'Expression'.

Die Berechnung der Standardabweichung erfolgt entsprechend der Formel:

$$s_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

s_x = Standardabweichung
$\bar{x}$ = Mittelwert
n = Anzahl Messungen

MStdDev

```
MStdDev('Expression','Interval')
```

Argumente

'Expression'	Messwert, für den die Standardabweichung gebildet wird
'Interval'	Länge des Intervalls, über das die Standardabweichung berechnet werden soll.

Beschreibung
Diese Funktion liefert als Ergebnis die gleitende Standardabweichung von 'Expression' über jedes Zeitintervall der Länge 'Interval'. Es können Signale und Ausdrücke verarbeitet werden, die zeitbasiert ('Interval' in Sekunden) oder längenbasiert ('Interval' in Metern) sind.

StdDevInTime

```
StdDevInTime('Expression','Interval')
```

Argumente

'Expression'	Messwert, für den die Standardabweichung gebildet wird
'Interval'	Länge des Intervalls, über das die Standardabweichung berechnet werden soll.

Beschreibung
Diese Funktion liefert als Ergebnis die Standardabweichung von 'Expression' über jedes Zeitintervall der Länge 'Interval'.

Hinweis

Die Standardabweichung wird immer für das vorhergehende Intervall angegeben.

StdDevValid

```
StdDevValid('Expression','Valid')
```

Argumente

'Expression'	Messwert, für den die Standardabweichung gebildet wird
'Valid'	Kontrollsignal

Beschreibung
Diese Funktion liefert als Ergebnis die Standardabweichung von 'Expression' für das Intervall (Zeit oder Länge) zurück, in dem ein zugehöriges Kontrollsignal WAHR ist.

5.5 Perzentile (Percentile)

Percentile

```
Percentile('Expression','p'=0.5)
```

Argumente

'Expression'	Messwert, für den die Perzentile berechnet wird
'p'	Die Perzentile

Beschreibung

Diese Funktion liefert als Ergebnis die 'p'-te Perzentile von 'Expression'.

Die Angabe der 'p'-ten Perzentile bezeichnet den kleinsten Wert einer Menge von Messwerten, der größer ist als p% der Anzahl der Messwerte. Eine typische Perzentile ist die 50%-Perzentile, der so genannte Median. Der Median teilt die Menge der Messwerte in zwei gleiche Hälften: 50% aller Messwerte sind kleiner als der Median-Wert, die anderen 50% sind größer oder gleichgroß. Weitere gebräuchliche Perzentilen sind 25% und 75%, die zusammen mit dem Median die Unterteilung einer Menge von Messwerten in vier Gruppen, die so genannten Quartile, erlauben. (< 25%, <50%, <75%, ≥75%).

Die Funktion "Percentile" ermittelt den Perzentilen-Wert aus der Gesamtmenge der Messpunkte eines Signals. Die Angabe der Perzentile 'p' muss als Dezimalwert erfolgen:

- 50 % --> p = 0.5 (Standard Wert)
- 75 % --> p = 0.75
- 95,9 % --> p = 0.959

Diese Funktion ist insbesondere dafür geeignet, z. B. die Qualität eines Produktes zu beurteilen, indem eine bestimmte Produkteigenschaft einer festgelegten Klassifizierung genügen muss.

PercentileValid

```
PercentileValid('Expression','Valid','p'=0.5)
```

Argumente

'Expression'	Messwert, für den die Perzentile berechnet wird
'Valid'	Angabe des Intervalls, über das die Perzentile gebildet wird
'p'	Die Perzentile

Beschreibung

Diese Funktion liefert als Ergebnis die Perzentile von 'Expression' für jedes Intervall (Zeit oder Länge), für das ein zugehöriges Kontrollsignal 'Valid' WAHR ist.

PercentileInTime

```
PercentileInTime('Expression','Interval','p'=0.5)
```

Argumente

'Expression'	Messwert, für den die Perzentile berechnet wird
'Interval'	Angabe der Intervallgröße, über das die Perzentile gebildet wird
'p'	Die Perzentile

Beschreibung

Diese Funktion liefert als Ergebnis die Perzentile von 'Expression' über jedes Zeitintervall der Länge 'Interval'.

MPercentile

```
MPercentile('Expression','Interval','p'=0.5)
```

Argumente

'Expression'	Messwert, für den die Perzentile berechnet wird
'Interval'	Angabe des gleitenden Intervalls, über das die Perzentile gebildet wird
'p'	Die Perzentile

Beschreibung

Diese Funktion liefert als Ergebnis die gleitende Perzentile von 'Expression' über jedes Intervall der Länge 'Interval'. Es können Signale und Ausdrücke verarbeitet werden, die zeitbasiert ('Interval' in Sekunden) oder längenbasiert ('Interval' in Metern) sind.

5.6 Korrelation und Kovarianz (Correl, CoVar)

Correl

```
Correl('Expression1','Expression2')
```

Argumente

'Expression1/2'	Messwerte, für die der Korrelationskoeffizient berechnet wird

Beschreibung

Diese Funktion berechnet den Korrelationskoeffizienten zwischen 'Expression1' und 'Expression2'. Betrachtet wird die gesamte Aufzeichnungslänge. Die Funktion liefert einen konstanten Wert zurück.

Mcorrel

```
Mcorrel('Expression1','Expression2','Interval')
```

Argumente

'Expression1/2'	Messwerte, für die der Korrelationskoeffizient berechnet wird
'Interval'	Angabe des Intervalls, über das der Korrelationskoeffizient gebildet wird

Beschreibung

Diese Funktion berechnet den Korrelationskoeffizienten zwischen 'Expression1' und 'Expression2' über gleitende Intervalle der Größe 'Interval' gemessen in s, m, Hz oder 1/m.

CoVar

```
CoVar('Expression1','Expression2')
```

Argumente

'Expression1/2'	Messwerte, für die die Kovarianz berechnet wird

Beschreibung

Diese Funktion berechnet die Kovarianz zwischen 'Expression1' und 'Expression2'. Betrachtet wird die gesamte Aufzeichnungslänge. Die Funktion liefert einen konstanten Wert zurück.

MCoVar

```
MCoVar('Expression1','Expression2','Interval')
```

Argumente

'Expression1/2'	Messwerte, für die die Kovarianz berechnet wird
'Interval'	Angabe des Intervalls, über das die Kovarianz gebildet wird

Beschreibung

Diese Funktion berechnet die Kovarianz zwischen 'Expression1' und 'Expression2' über gleitende Intervalle der Länge 'Interval', gemessen in s, m, Hz oder 1/m.

5.7 Kurtosis (Kurtosis)

Die Berechnung der *Kurtosis* wird z. B. für die Bewertung und Analyse von Schwingungen verwendet. Sie dient dazu, innerhalb eines Schwingungssignals die Anzahl von Ausreißern zu bestimmen.

Mathematisch gesehen ist die Kurtosis ein Maß für die relative "Flachheit" einer Verteilung (im Vergleich zur Normalverteilung, die eine Kurtosis von null aufweist). Eine postive Kurtosis zeigt eine spitz zulaufende Verteilung (eine so genannte leptokurtische Verteilung), wohingegen eine negative Kurtosis eine flache Verteilung (platykurtische Verteilung) anzeigt.

Diese statistische Methode eignet sich speziell zur Analyse zufälliger oder stochastischer Signale, z. B. in der zustandsorientierten Instandhaltung (Condition Monitoring) bei der Analyse von Schwingungen.

Zur Charakterisierung des Signalverlaufes werden Methoden der Wahrscheinlichkeitsdichte oder Häufigkeit verwendet. Dabei wird von der Annahme ausgegangen, dass nach dem Ausfiltern z. B. von drehfrequenten Schwingungsanteilen bei intakten Maschinen ein Rauschsignal mit einer Gauß'schen Amplitudenverteilung messbar ist. Diesem Signal überlagern sich bei auftretender Schädigung einzelne Impulssignale, welche die Verteilungsfunktion verändern. Durch die Bildung geeigneter Kennwerte, wie dem *Crest-Faktor* oder dem *Kurtosis-Faktor* kann eine Bewertung des Anlagenzustandes stattfinden.

Diese Verfahren bieten bei regelmäßigen Messungen einen Überblick über den Zustand der Maschine. Der Nachteil liegt jedoch darin, dass die Kennwerte, nachdem sie angestiegen sind, wieder sinken. Grund dafür ist, dass bei fortschreitender Schädigung die Anzahl der Impulssignale ansteigt. Das wiederum beeinflusst den Effektivwert, jedoch kaum den Spitzenwert.

Durch Stoßimpulse hervorgerufene Veränderungen des Zeitsignals bewirken eine Veränderung der sich ergebenen Verteilungsfunktion. Schädigungen mit ausgeprägtem diskretem Charakter lassen den *Kurtosis-Faktor* somit stark ansteigen. Sein Absolutwert lässt somit bereits Aussagen über eine Schädigung zu.

Die Berechnung der Kurtosis erfolgt nach ähnlichem Muster wie die Berechnung der Standardabweichung ‚StdDev'.

Kurtosis

```
Kurtosis('Expression')
```

Argumente

'Expression'	Messwert, für den die Kurtosis gebildet wird

Beschreibung

Diese Operation liefert als Ergebnis die Kurtosis des gewählten Zeitsignals (Ausdruck).

KurtosisInTime

```
KurtosisInTime('Expression','Interval')
```

Argumente

'Expression'	Messwert, für den die Kurtosis gebildet wird
'Interval'	Länge des Intervalls, über das die Kurtosis berechnet werden soll.

Beschreibung

Bei dieser Operation wird der ausgewählte Ausdruck in gleichlange Intervalle der Größe 'Interval' unterteilt. Für diese Intervalle erfolgt anschließend die Berechnung der Kurtosis.

MKurtosis

```
MKurtosis('Expression','Interval')
```

Argumente

'Expression'	Messwert, für den die Kurtosis gebildet wird
'Interval'	Angabe der Länge des Intervalls in Sekunden, über das die Kurtosis gebildet wird

Beschreibung

Bei dieser Operation wird die Kurtosis von 'Ausdruck' über ein festes aber gleitendes X-Achsenintervall berechnet.

KurtosisValid

```
KurtosisValid('Expression','Valid')
```

Argumente

'Expression'	Messwert, für den die Kurtosis gebildet wird
'Valid'	Kontrollsignal

Beschreibung

Diese Operation gibt die Kurtosis für die Bereiche wieder, in denen ein zugehöriges Kontrollsignal WAHR ist.

5.8 Skewness (Skewness)

Ähnlich wie der Kurtosis-Faktor so eignet sich der Skewness-Faktor zur Bewertung und Analyse von Schwingungen. Der Skewness-Faktor kann dann angewendet werden, wenn z. B. die Symmetrieeigenschaften eines Schwingungssignals überprüft werden sollen.

Mathematisch gesehen handelt es sich hierbei um die Beurteilung der Schiefe (Skewness) einer Verteilungsfunktion. Eine Verteilung wird rechtsschief (bzw. linkssteil) genannt, wenn der Hauptanteil der Verteilung auf der linken (bzw. rechten) Seite konzentriert ist. Der Grad der Schiefe wird durch das dritte Moment der Verteilung bestimmt.

Für die Berechnung der Skewness wird ähnlich wie bei den Funktionen Kurtosis und Standardabweichung verfahren:

Skewness

```
Skewness('Expression')
```

Beschreibung

Diese Operation liefert als Ergebnis die Skewness des gewählten Zeitsignals 'Expression'.

SkewnessInTime

```
SkewnessInTime('Expression','Interval')
```

Argumente

'Expression'	Messwert, für den die Skewness gebildet wird
'Interval'	Länge des Intervalls, über das die Skewness berechnet werden soll.

Beschreibung

Bei dieser Operation wird der ausgewählte Ausdruck in gleichlange Intervalle der Größe 'Interval' unterteilt. Für diese Intervalle erfolgt anschließend die Berechnung der Skewness.

MSkewness

```
MSkewness('Expression','Interval')
```

Argumente

'Expression'	Messwert, für den die Skewness gebildet wird
'Interval'	Länge des gleitenden Intervalls, über das die Skewness berechnet werden soll.

Beschreibung

Bei dieser Operation wird die Skewness von 'Expression' über ein festes aber gleitendes X-Achsenintervall berechnet.

SkewnessValid

```
SkewnessValid('Expression','Valid')
```

Argumente

'Expression'	Messwert, für den die Skewness gebildet wird
'Valid'	Kontrollsignal

Beschreibung

Diese Operation gibt die Skewness für die Bereiche wieder, in denen ein zugehöriges Kontrollsignal WAHR ist.

6 Zählen und Sortieren

6.1 Count

```
Count('Expression','Level'=0.5,'Hysteresis'=0,'EdgeType'=1,'Reset=0')
```

Argumente

'Expression'	Messwert	
'Level'	Angabe des Niveauwertes	
'Hysteresis'	Angabe eines Hysteresebandes	
'EdgeType'	Angabe, ob steigende, fallende oder steigende und fallende Flanken gezählt werden sollen	
	'EdgeType' <0	nur fallende Flanken (verlassen des Hysteresebands in negativer Richtung)
	'EdgeType' >0	nur steigende Flanken (verlassen des Hysteresebands in positiver Richtung)
	'EdgeType' = 0	fallende und steigende Flanken
'Reset'	Optionaler digitaler Parameter, der zum Rücksetzen des Zählers verwendet werden kann. 'Reset' kann selbst auch ein Ausdruck sein.	
	'Reset' > 0	Zähler wird zurückgesetzt
	'Reset' = 0	Zählerwert bleibt erhalten/zählt weiter (Voreinstellung)

Hinweis

Die 'Reset'-Bedingung darf nicht auf die *Count*-Funktion selbst bezogen sein.

Beschreibung

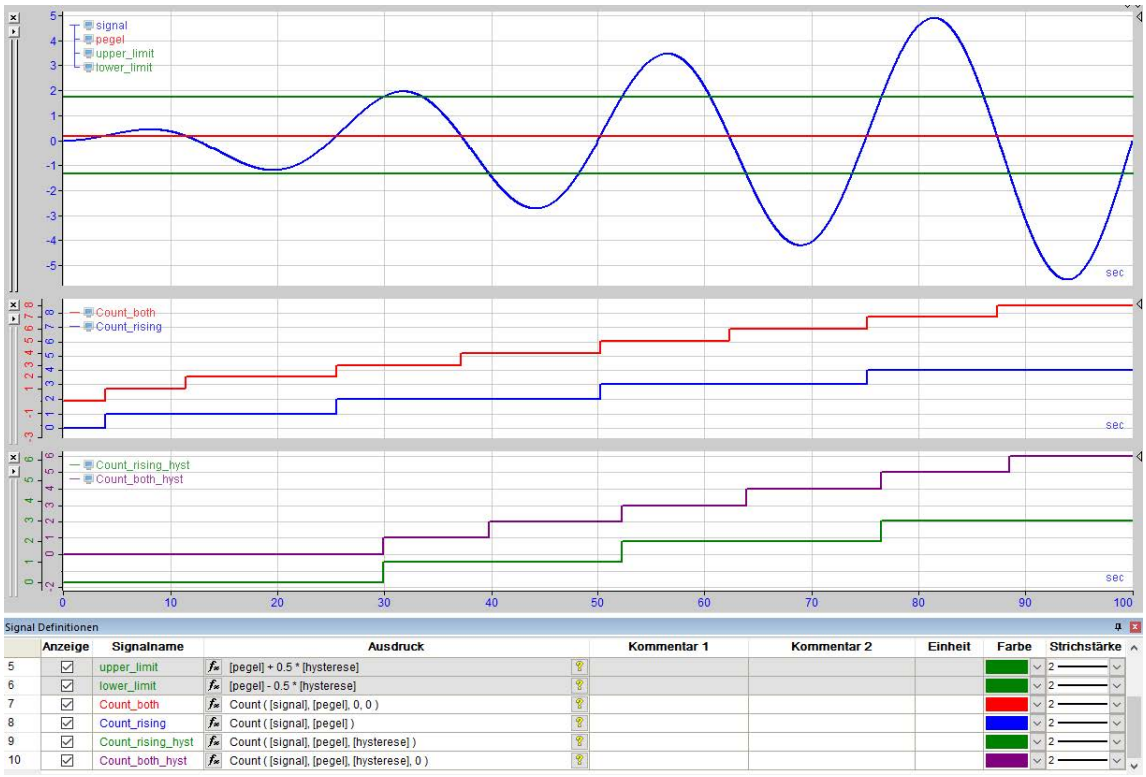
Die Funktion zählt die Durchgänge von 'Expression' durch das Niveau 'Level'.

Mit dem Parameter 'Hysteresis' kann ein Toleranzband angegeben werden, was zu gleichen Teilen ober- und unterhalb von 'Level' liegt. Es werden nur komplette Durchgänge durch das Toleranzband gezählt.

Der Parameter 'EdgeType' bestimmt, welche Flanken gezählt werden. Der Parameter 'Reset' dient zum Rücksetzen des Zählerwertes auf 0. 'Reset' kann auch als Ausdruck formuliert werden.

Beispiel

Wenn für 'Level' 2.5 eingegeben wurde und für 'Hysteresis' 2.0, dann werden Pegeldurchgänge in steigender Richtung erst bei 'Expression' > 3.5 und in fallender Richtung erst bei 'Expression' < 1.5 gezählt.



Signal Definitionen

	Anzeige	Signalname	Ausdruck	Kommentar 1	Kommentar 2	Einheit	Farbe	Strichstärke
5	☑	upper_limit	[pegel] + 0.5 * [hysterese]					2
6	☑	lower_limit	[pegel] - 0.5 * [hysterese]					2
7	☑	Count_both	Count ([signal], [pegel], 0, 0)					2
8	☑	Count_rising	Count ([signal], [pegel])					2
9	☑	Count_rising_hyst	Count ([signal], [pegel], [hysterese])					2
10	☑	Count_both_hyst	Count ([signal], [pegel], [hysterese], 0)					2

Tipp

Die *Count*-Funktion kann auch für binäre Signale verwendet werden. Dazu gibt man als Pegel 0.5 und als Hysterese z. B. 0.1 ein. Damit werden alle Wechsel von FALSE nach TRUE und umgekehrt erfasst und gezählt.

6.2 CountSamples

```
CountSamples('Expression','Reset'=0)
```

Argumente

'Expression'	Messwert, für den die Anzahl der Signalpunkte ermittelt wird
'Reset'	Optionaler digitaler Parameter, der zum Rücksetzen bzw. Unterdrücken des Zählvorgangs verwendet werden kann. 'Reset' kann selbst auch ein Ausdruck sein 'Reset' > 0 Zählvorgang wird zurückgesetzt. 'Reset' = 0 Zählvorgang freigegeben (Voreinstellung)

Beschreibung

Mit dieser Funktion kann die Anzahl der einzelnen Signalpunkte ermittelt werden, ganz egal ob die Signalpunkte äquidistant sind oder nicht. Ungültige Signale werden nicht gezählt. Ist das Eingangssignal ungültig, wird als Ergebnis der konstante Wert 0 ausgegeben.

Tipp

Diese Funktion kann auch in Kombination mit beispielsweise XMarkValid (siehe XMark-Funktionen ↗ *XMarkRange / XMarkValid*, Seite 54) benutzt werden.

6.3 Sort

```
Sort('Expression','Descending'=0)
```

Argumente

'Expression'	Messwert, für den die Samples sortiert werden
'Descending'	Optionaler digitaler Parameter zum Umkehren der Sortierreihenfolge

Beschreibung
Diese Funktion sortiert alle Samples einer Kurve ('Expression') entsprechend ihrer Werte in aufsteigender Reihenfolge von links nach rechts.

Voreinstellung: Sortierung in aufsteigender Reihenfolge (‘Descending‘=FALSE). Sollen die Samples in absteigender Reihenfolge von links nach rechts sortiert werden, dann muss als zweiter Operand TRUE gesetzt werden.

7 Zeit - Länge - Funktionen

7.1 Konvertieren und Neuabtasten

7.1.1 ConvertBase

```
ConvertBase('Expression','From','To')
```

Argumente

'Expression'	Messwert, für den die Basis geändert werden soll
'From'/‘To'	Einstellung der Basis von bzw. zu der gewechselt werden soll. 0 = Zeit 1 = Länge 2 = Frequenz 3 = inverse Länge

Beschreibung
Diese Operation wandelt einen Ausdruck von einer Basis zu einer anderen Basis. Es wird keine physikalische Umrechnung oder Skalierung vorgenommen.

Diese Funktion dient dazu, die Bezugsgröße eines Signals zu ändern. Dies kann dann von Vorteil sein, wenn für weitere Berechnungen beispielsweise längenbasierte Bezugsgrößen verwendet werden, das vorhandene Signal jedoch nur zeitbasiert vorliegt.

7.1.2 Resample (Neuabtasten)

```
Resample('Expression','Basis','interpolate'=1)
```

Argumente

'Expression'	Messwert, der neu abgetastet werden soll
'Basis‘	Neue Abtastrate des Ergebnisses
'interpolate'	Optionaler Parameter, um die automatische Interpolation für die neuen Messwerte zu verhindern

Beschreibung
Diese Operation liefert als Ergebnis den Signalverlauf von 'Expression' auf einer neuen Zeitbasis. Dabei werden aus dem Originalverlauf die Momentanwerte entsprechend der neuen Zeitbasis zeitrichtig übertragen, so dass die Länge der neuen Kurve praktisch gleich ist. Die Funktion kann auch für längenbasierte Signale verwendet werden. Anstelle eines Zeitraums ist dann der Betrag eines Weges in m einzugeben.

Tipp

Eine Kurve lässt sich grafisch glätten, wenn in der Resample Funktion eine größere Zeitbasis verwendet wird, da weniger Punkte miteinander verbunden werden. Die Werte werden nicht gemittelt.

7.1.3 SampleAndHold

```
SampleAndHold('Expression','Sample','Initial'=0)
```

Argumente

'Expression'	Messwert
'Sample'	Parameter, der bestimmt, ob die Funktion dem Messwert folgt (1) oder den letzten Messwert hält (0). 'Sample' kann selbst eine Bedingung sein oder durch eine andere Funktion bestimmt werden.
'Initial'	Optionaler Parameter (Voreinstellung = 0), der den Initialwert der Funktion festlegt, wenn bei Beginn der Messung 'Sample' inaktiv ist.

Beschreibung
Diese Funktion ist eine Abtast-Halte-Funktion. Der Ausgang folgt 'Expression', wenn 'Sample' = TRUE. Er bleibt unverändert, wenn 'Sample' = FALSE. Mit dem optionalen Parameter 'Initial' kann der Initialwert des Ausgangs angegeben werden, wenn die Funktion beim Aufruf auf "Halten" steht.

7.1.4 SampleOnce

```
SampleOnce('Expression','Sample')
```

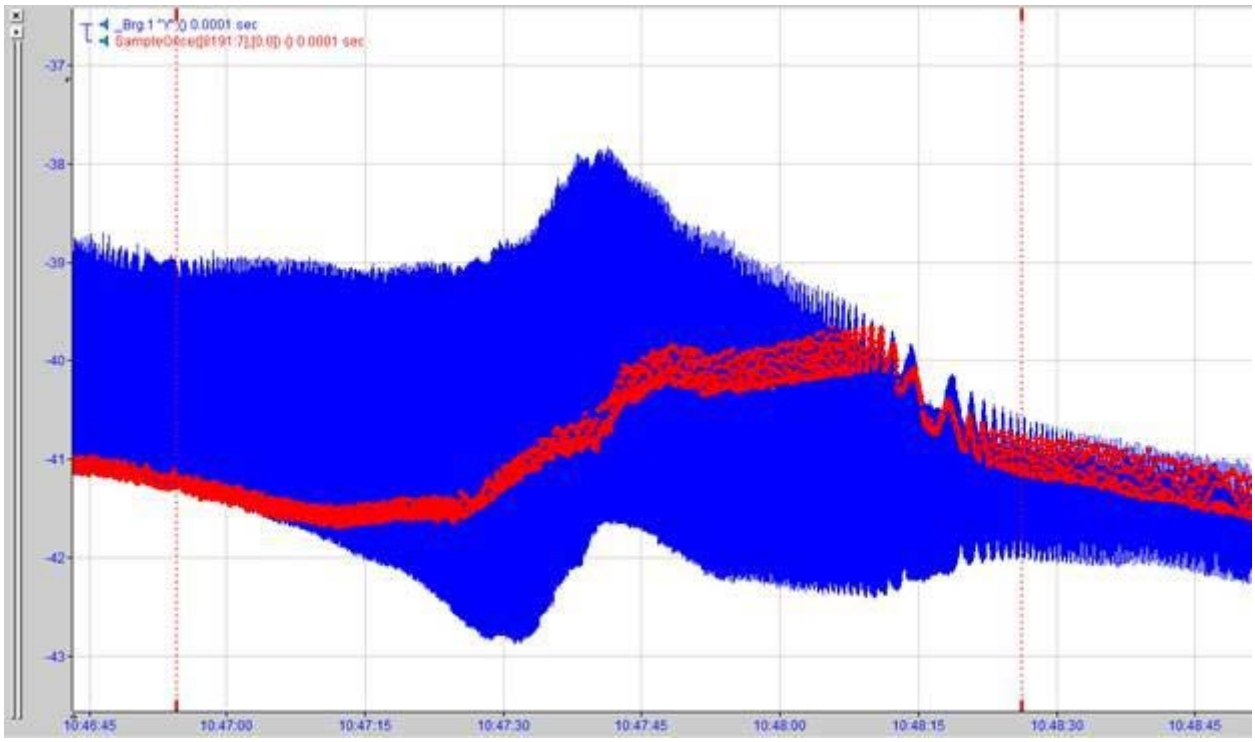
Argumente

'Expression'	Messwert
'Sample'	Digitales Signal, dessen steigende Flanken die Abtastpunkte festlegen.

Beschreibung
Diese Funktion tastet ein Eingangssignal ‘Expression‘ an einzelnen Punkten, die durch die steigenden Flanken des digitalen Signals ‘Sample‘ bestimmt werden, neu ab. Das Ergebnis enthält einen Messpunkt pro steigender Flanke und ist ungültig in den Bereichen dazwischen.

Beispiel
Diese Funktion kann genutzt werden um ein Phasengeber (keyphasor)-Signal im Zeitbereich darzustellen. Immer wenn der Phasengeber auf ‘TRUE‘ springt, wird das ursprüngliche Signal abgetastet. Durch Überlagerung beider Darstellungen werden die Zeitpunkte des Phasengebers passend dargestellt. Im Beispiel unten kann eine 180° Phasenverschiebung beim Durchlaufen einer Resonanz erkannt werden.

Die nachfolgende Abbildung zeigt die Funktion *SampleOnce* angewendet auf ein Schwingungssignal.


-37
-38
-39
-40
-41
-42
-43
10:46:45
10:47:00
10:47:15
10:47:30
10:47:45
10:48:00
10:48:15
10:48:30
10:48:45

7.2 Time

7.2.1 Time

```
Time('Count','Basis')
```

Argumente

'Count'	Anzahl der zu erzeugenden Messpunkte
'Basis‘	Abtastrate des Ergebnisses

Diese Funktion liefert als Ergebnis ein lineares, zeitproportionales Signal mit einer Anzahl ‘Count‘ Werte im Abstand von 'Basis'. Die Angabe der Zeitbasis versteht sich in Sekunden. Die Zeitwerte werden dabei sowohl auf der X-Achse als auch auf der Y-Achse abgetragen.

Hinweis

Für die Verwendung der Time-Funktion muss keine Messdatei geladen werden.

7.2.2 AbsoluteTime

```
AbsoluteTime('Time','DoSync'=0)
```

Argumente

'Time'	Relative Zeit, die konvertiert werden soll
'DoSync'	Optionaler Parameter, der festlegt, ob die absolute Zeit am Startzeitpunkt der aktuell angezeigten Zeitachse ausgerichtet werden soll

Beschreibung
Diese Funktion transformiert die relative Zeitinformation 'Time' (z. B. generiert mit XFirst, XLast oder XValues) in absolute Zeit. Dabei wird abhängig davon, ob die Eingangszeit konstant ist oder nicht, ein Vektor mit konstanten oder variierenden Einträgen zurückgegeben. Der optionale, binäre Parameter 'DoSync' legt fest, ob das Ergebnis am Startzeitpunkt der aktuell angezeigten Zeitachse ausgerichtet werden soll.

Das Ergebnis ist ein Vektor mit folgenden Einträgen, die mit GetRows ausgelesen werden können:

- Index 0: Millisekunden
- Index 1: Sekunden
- Index 2: Minuten
- Index 3: Stunden
- Index 4: Tag des Monats
- Index 5: Monat

- Index 6: Jahr
- Index 7: Tag des Jahres
- Index 8: Wochentag (1=Montag, 2=Dienstag, …, 7=Sonntag)

7.3 Umrechnung von Zeit- auf Längenbezug

TimeToLength

```
TimeToLength('Expression','Speed','Precision')
```

Argumente

'Expression'	Ausdruck, der auf Länge umgerechnet werden soll
'Speed'	Geschwindigkeitssignal
'Precision'	Optionaler Parameter, der die Abtastrate des Ergebnisses in m festlegt.

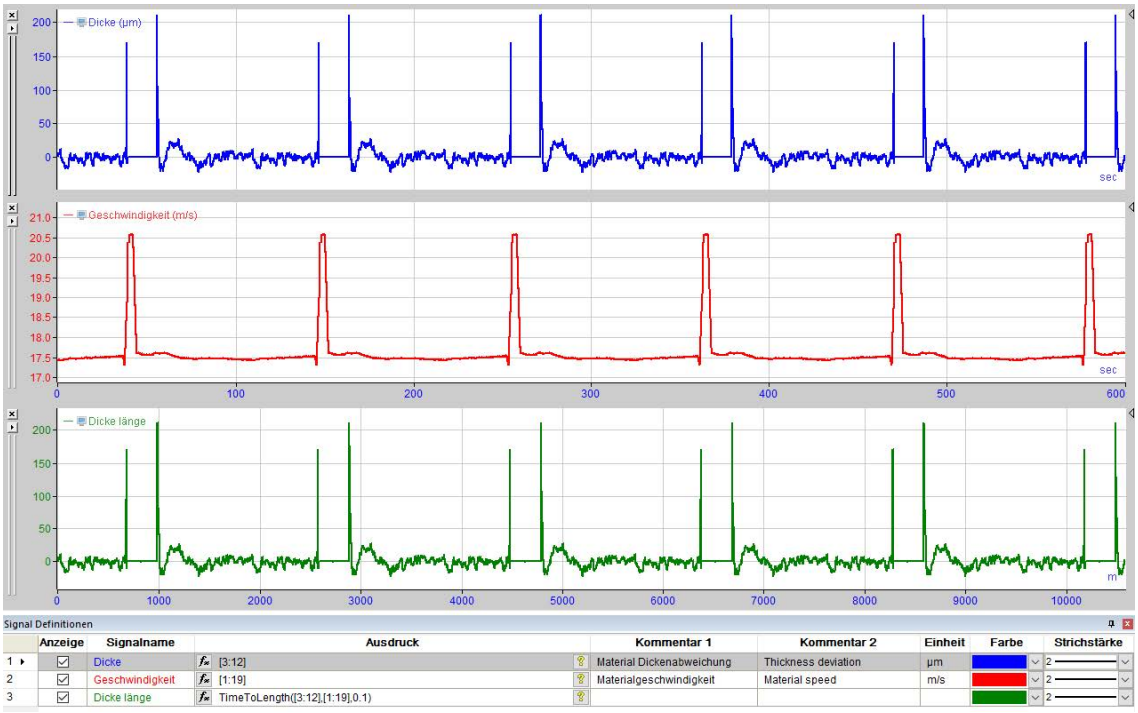
Beschreibung

Diese Funktion wandelt den zeitbezogenen Messwert 'Expression' in einen längenbezogenen um, mit der Geschwindigkeit des Messobjektes 'Speed' als Vortriebsvektor [m/s].

Jeder Messwert, zu dem ein passender Geschwindigkeitsmesswert vorliegt, kann mit dieser Funktion in eine längenbezogene Darstellung umgerechnet werden. D.h., dass nicht nur der Zusammenhang zwischen Messwert und Zeit sondern auch zwischen Messwert und zurückgelegtem Weg dargestellt werden kann. Am Beispiel eines Stahlbands im Walzwerk kann mit dieser Funktion die Verteilung der Messwerte entlang der Bandlänge ermittelt werden. Wenn prozesstechnisch dafür gesorgt wurde, dass Beginn und Ende der Messung exakt mit Anfang und Ende des Bandes übereinstimmen, dann lässt sich mit dieser Funktion auch die Gesamtlänge des Bandes berechnen. Der größte ermittelte Längenwert wird als Skalenendwert an der X-Achse eingetragen (Autoskalierung).

'Precision' ist eine optionale Angabe in [m]. Wenn nicht angegeben, dann werden die Punkte für die längenbezogene Kurve entsprechend der Anzahl der Messpunkte des Originalsignals berechnet und im Signalstreifen eingetragen. Wenn eine Genauigkeit angegeben wird, z. B. 0.1, dann wird alle 0,1 m ein neuer längenbezogener Wert berechnet und als Punkt der Kurve eingetragen.

Die nachfolgende Abbildung zeigt Zeit-Länge-Funktionen *TimeToLength*.



TimeToLengthL

```
TimeToLengthL('Expression','Length','Precision')
```

Argumente

'Expression'	Ausdruck, der auf Länge umgerechnet werden soll
'Length'	Längensignal
'Precision'	Optionaler Parameter, der die Abtastrate des Ergebnisses in m festlegt.

Beschreibung

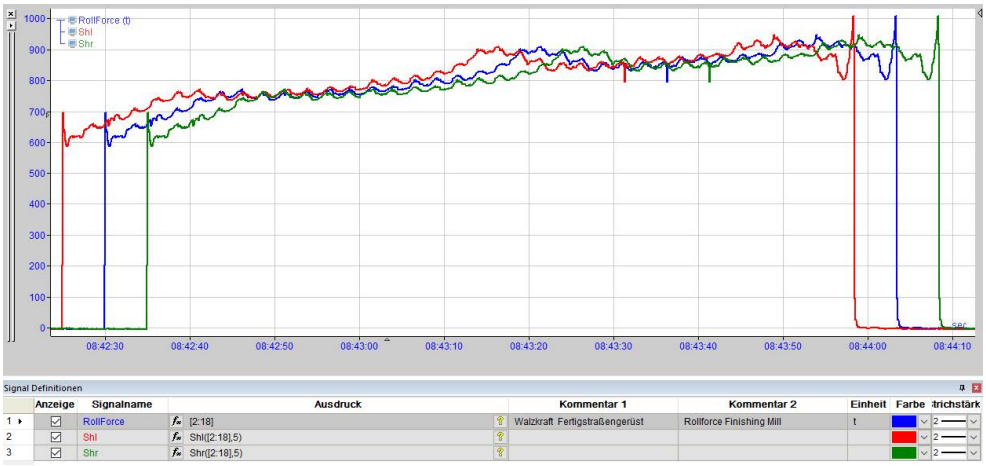
Diese Funktion wandelt den zeitbezogenen Messwert 'Expression' in einen längenbezogenen um, mit einem Längenmesswert 'Length' als Position [m].

Die Erläuterungen unter *TimeToLength* gelten entsprechend, nur dass anstelle der Geschwindigkeit ein passender Längen- oder Positionsmesswert verwendet wird.

8 X-Achsen-Operationen

8.1 Verschiebung entlang der X-Achse

Die folgende Abbildung zeigt Zeit-Länge-Funktionen Shift links (rot) / rechts (grün).



Shl

```
Shl('Expression','Distance')
```

Argumente

'Expression'	Ausdruck, der verschoben werden soll
'Distance'	Verschiebung in Sekunden, bzw. in Metern bei längenbezogenen Signalen

Diese Operation liefert als Ergebnis einen Signalverlauf, der um einen Betrag der Länge 'Distance' auf der X-Achse nach links gegenüber dem Originalsignal verschoben ist. Ansonsten bleiben die Messwerte unverändert. Die Funktion kann sowohl für zeitbasierte Signale ('Verschiebung' in Sekunden) als auch für längenbasierte Signale ('Verschiebung' in Metern) verwendet werden.

Shr

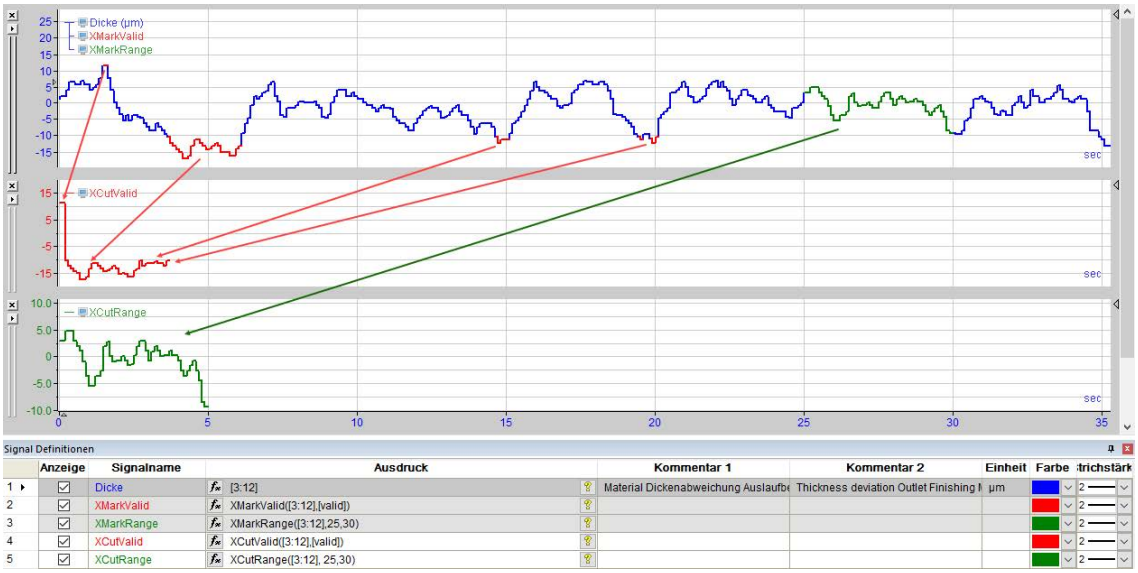
```
Shr('Expression','Distance')
```

Argumente

'Expression'	Ausdruck, der verschoben werden soll
'Distance'	Verschiebung in Sekunden, bzw. in Metern bei längenbezogenen Signalen

Diese Operation liefert als Ergebnis einen Signalverlauf, der um einen Betrag der Länge 'Distance' auf der X-Achse nach rechts gegenüber dem Originalsignal verschoben ist. Ansonsten bleiben die Messwerte unverändert. Die Funktion kann sowohl für zeitbasierte Signale ('Verschiebung‘ in Sekunden) als auch für längenbasierte Signale ('Verschiebung‘ in Metern) verwendet werden.

8.2 XCutRange / XCutValid



XCutRange

```
XCutRange('Expression','Start','End')
```

Argumente

'Expression'	Ausdruck, aus dem ein Teil ausgeschnitten werden soll
'Start'	Anfang des ausgewählten Bereichs in Sekunden bzw. Metern
'End'	Ende des ausgewählten Bereichs in Sekunden bzw. Metern

Beschreibung

Mit dieser Funktion kann ein Bereich eines Kurvenzuges herausgeschnitten werden. Die Funktion kann sowohl auf zeit- als auch längenbezogene Signalstreifen angewendet werden. Die Parameter 'Start' und 'End', angegeben in [s] oder [m], definieren Anfang und Ende des auszuschneidenden Bereiches.

Der herausgeschnittene Teil wird an den Anfang eines eigenen Signalstreifens geschoben. Da die X-Achse (Zeit oder Länge) jedoch unverändert bleibt, ist der korrekte Zeit- bzw. Längenbezug der Messwerte nicht mehr gegeben.

XCutValid

```
XCutValid('Expression','Valid')
```

Argumente

'Expression'	Ausdruck, aus dem ein Teil ausgeschnitten werden soll
'Valid'	Binärsignal, welches den ausgewählten Bereich beschreibt

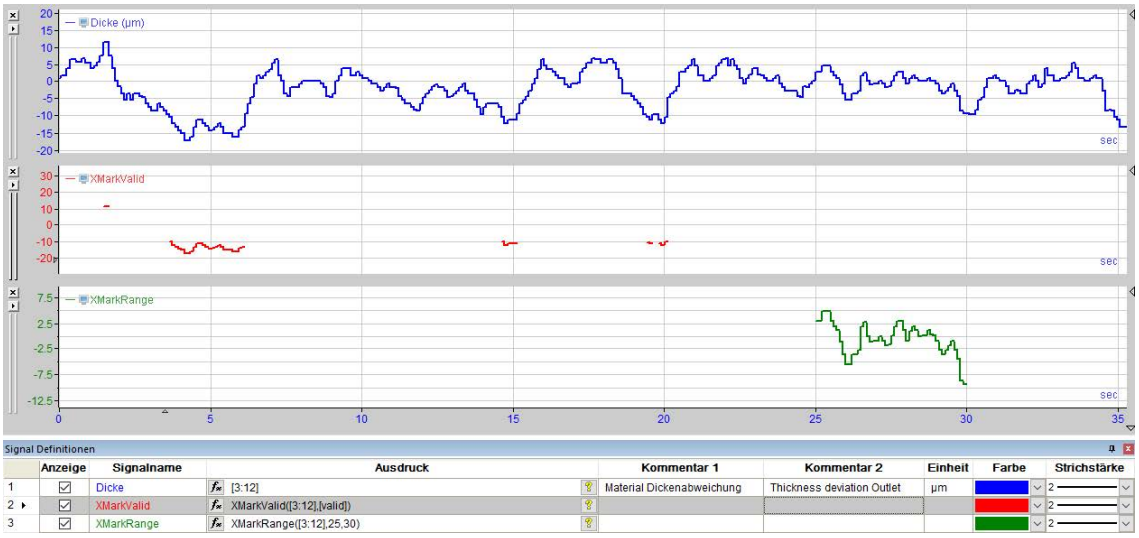
Beschreibung

Mit dieser Funktion werden alle Messpunkte eines Signalverlaufes 'Expression' abhängig von einer Bedingung 'Valid' herausgeschnitten, wenn diese Bedingung den Wert TRUE liefert. Die Funktion kann sowohl auf zeit- als auch längenbezogene Signale angewendet werden. Der Parameter 'Valid' ist ein boolescher Ausdruck. Das kann ein digitales Eingangssignal sein, das

Ergebnis einer Vergleichsoperation oder jeder andere binäre Ausdruck. Messpunkte, für die die Bedingung FALSE ist, werden nicht übernommen.

Die herausgeschnittenen Teile werden hintereinander an den Anfang eines neuen Signalstreifens gestellt.

8.3 XMarkRange / XMarkValid



XMarkRange

```
XMarkRange('Expression','Start','End')
```

Argumente

'Expression'	Ausdruck, aus dem ein Teil ausgewählt werden soll
'Start'	Anfang des ausgewählten Bereichs in Sekunden bzw. Metern
'End'	Ende des ausgewählten Bereichs in Sekunden bzw. Metern

Beschreibung

Mit dieser Funktion kann ein Bereich eines Kurvenzuges ähnlich wie mit *XCutRange* herausgeschnitten werden. Die Funktion kann sowohl auf zeit- als auch längenbezogene Signalstreifen angewendet werden. Die Parameter 'Start' und 'End', angegeben in [s] oder [m], definieren Anfang und Ende des auszuschneidenden Bereiches. Der herausgeschnittene Teil wird in einem eigenen Signalstreifen dargestellt, bleibt jedoch an der Originalstelle auf der Zeit- oder Wegachse stehen, und die Messpunkte, die sich nicht in dem angegebenen Bereich befinden, werden verworfen.

XMarkValid

```
XMarkValid('Expression','Valid')
```

Argumente

'Expression'	Ausdruck, aus dem ein Teil ausgewählt werden soll
'Valid'	Binärsignal, welches den ausgewählten Bereich beschreibt

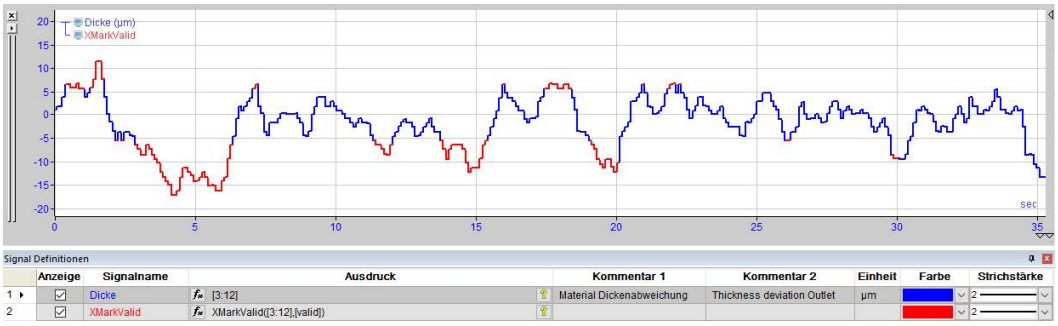
Beschreibung

Mit dieser Funktion werden - ähnlich wie mit *XCutValid* - alle Messpunkte eines Signalverlaufes 'Expression' abhängig von einer Bedingung 'Valid' herausgeschnitten, wenn diese Bedingung den Wert TRUE liefert. Die Funktion kann sowohl auf zeit- als auch längenbezogene Signalstreifen angewendet werden. Der Parameter 'Valid' ist ein boolescher Ausdruck. Das kann ein digitales Eingangssignal sein, das Ergebnis einer Vergleichsoperation oder jeder andere binäre Ausdruck. Messpunkte, für die die Bedingung FALSE ist, werden verworfen. Die herausgeschnittenen Teile werden in einem neuen Signalstreifen dargestellt, wobei sie ihre X-Positionen beibehalten.

Tipp

Die Funktion *XMarkValid* eignet sich besonders gut dazu, z. B. Grenzwertverletzungen farbig in einem Signalverlauf hervorzuheben, indem das Ergebnissignal im selben Streifen und auf derselben Y-Achse wie das Originalsignal dargestellt wird. Durch eine andere Farbgebung lassen sich die Bereiche der Grenzverletzung leicht erkennen.

Beispiel: Werte in Toleranz = blau, Werte außer Toleranz = rot



8.4 XMirror / XStretch / XStretchScale

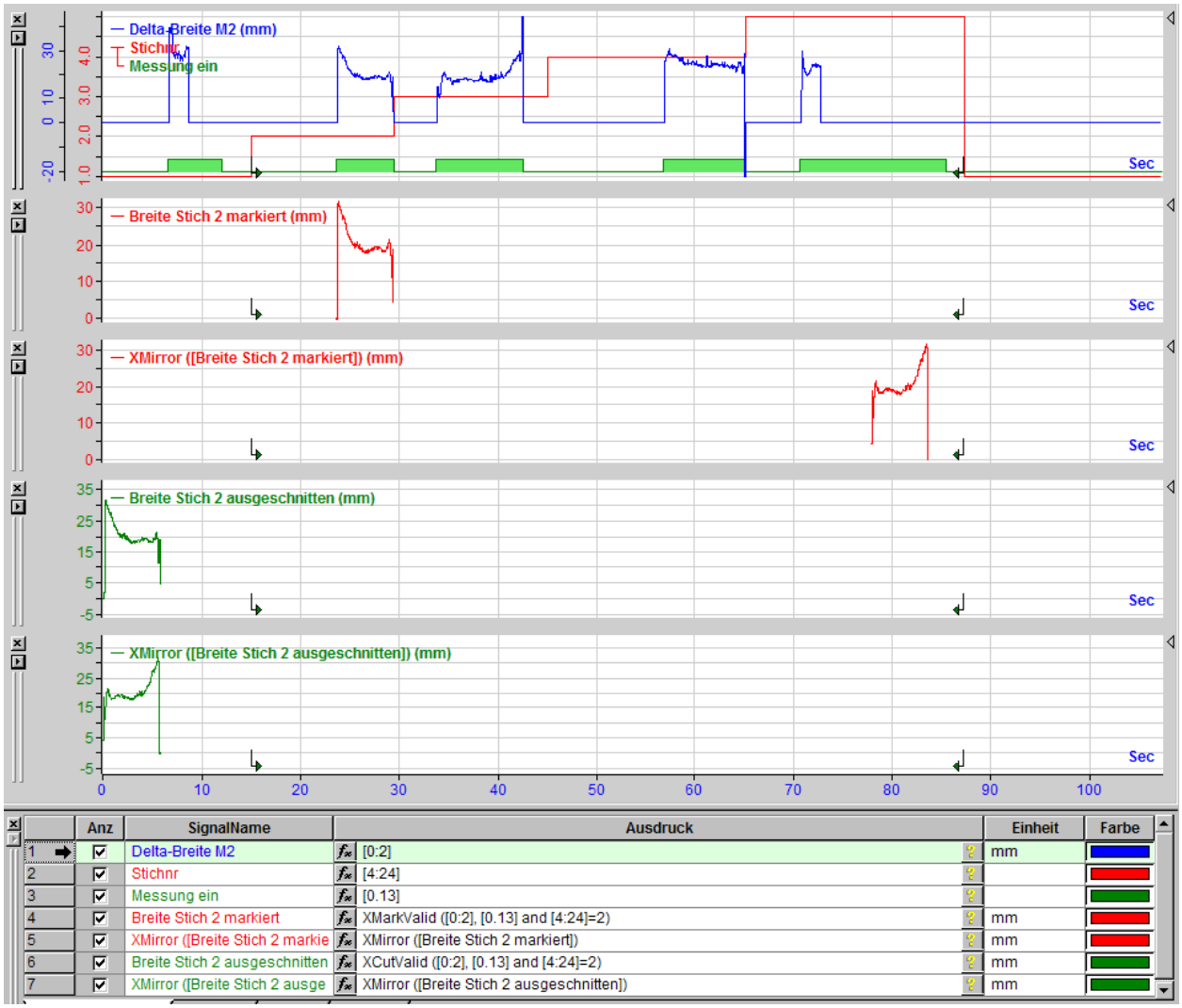
XMirror

```
XMirror('Expression')
```

Beschreibung

Mit dieser Funktion kann ein kompletter Kurvenverlauf gespiegelt werden (Anfang und Ende vertauschen). Die Spiegelung erfolgt dabei um die vertikale Mittelachse des gesamten Signalverlaufs. Die Funktion kann sowohl auf zeit- als auch längenbezogene Signalstreifen angewendet werden.

Damit lassen sich Messkurven von reversierenden Prozessen (Richtungsumkehr) besser miteinander vergleichen. So können beispielsweise in der Walztechnik Bandkopf und Bandende bei (geraden) Reversierstichen vertauscht werden, um die Richtungsumkehr grafisch zu neutralisieren. Um mehrere Stiche miteinander vergleichen zu können, müssen die entsprechenden Messwerte jedoch zuvor mit der *XCutValid*-Funktion aus dem Originalsignal herausgeschnitten werden, um sie individuell spiegeln und später übereinanderlegen zu können.



	Anz	SignalName	Ausdruck	Einheit	Farbe
1	☑	Delta-Breite M2	[0:2]	mm	
2	☑	Stichnr	[4:24]		
3	☑	Messung ein	[0.13]		
4	☑	Breite Stich 2 markiert	XMarkValid ([0:2], [0.13] and [4:24]=2)	mm	
5	☑	XMirror ([Breite Stich 2 markie	XMirror ([Breite Stich 2 markiert])	mm	
6	☑	Breite Stich 2 ausgeschnitten	XCutValid ([0:2], [0.13] and [4:24]=2)	mm	
7	☑	XMirror ([Breite Stich 2 ausge	XMirror ([Breite Stich 2 ausgeschnitten])	mm	

Im Bild oben ist das unterschiedliche Ergebnis der Spiegelung zu sehen, je nachdem, ob der zu spiegelnde Ausschnitt zuvor mit *XMarkValid* (rot) oder mit *XCutValid* (grün) herausgeschnitten wurde.

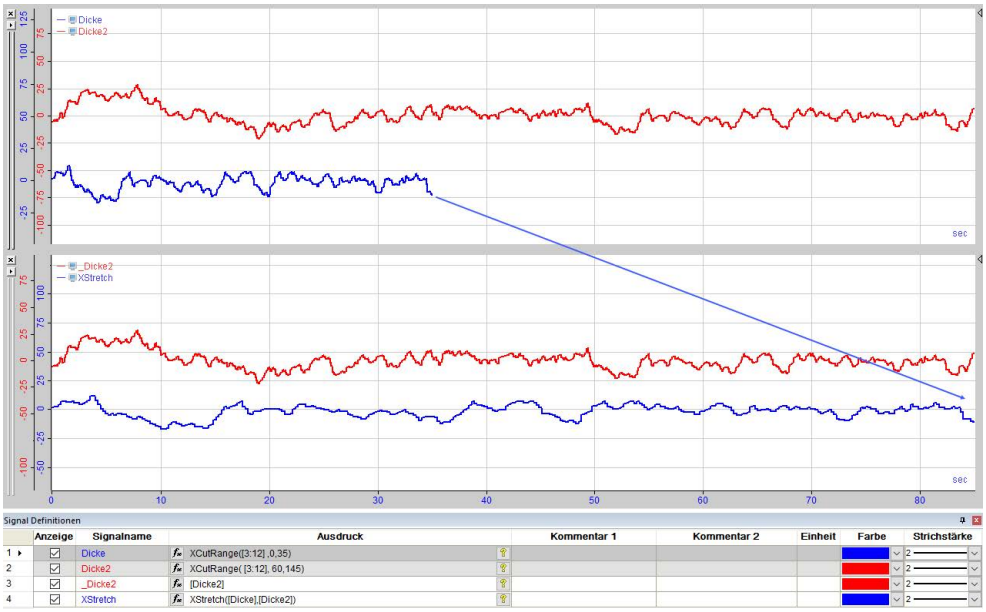
XStretch

```
XStretch('Expression','ReferenceExpression')
```

Beschreibung

Mit dieser Funktion kann der Kurvenzug eines Signals auf die gleiche (End-)Länge eines anderen Signals grafisch gestreckt werden. Die Funktion kann sowohl auf zeit- als auch längenbezogene Signalstreifen angewendet werden.

So lassen sich z. B. Messwerte eines gewalzten Bandes von der Vorstraße mit denen von der Fertigstraße in Relation setzen oder die einzelnen Stiche eines Reversierwalzwerks miteinander vergleichen.



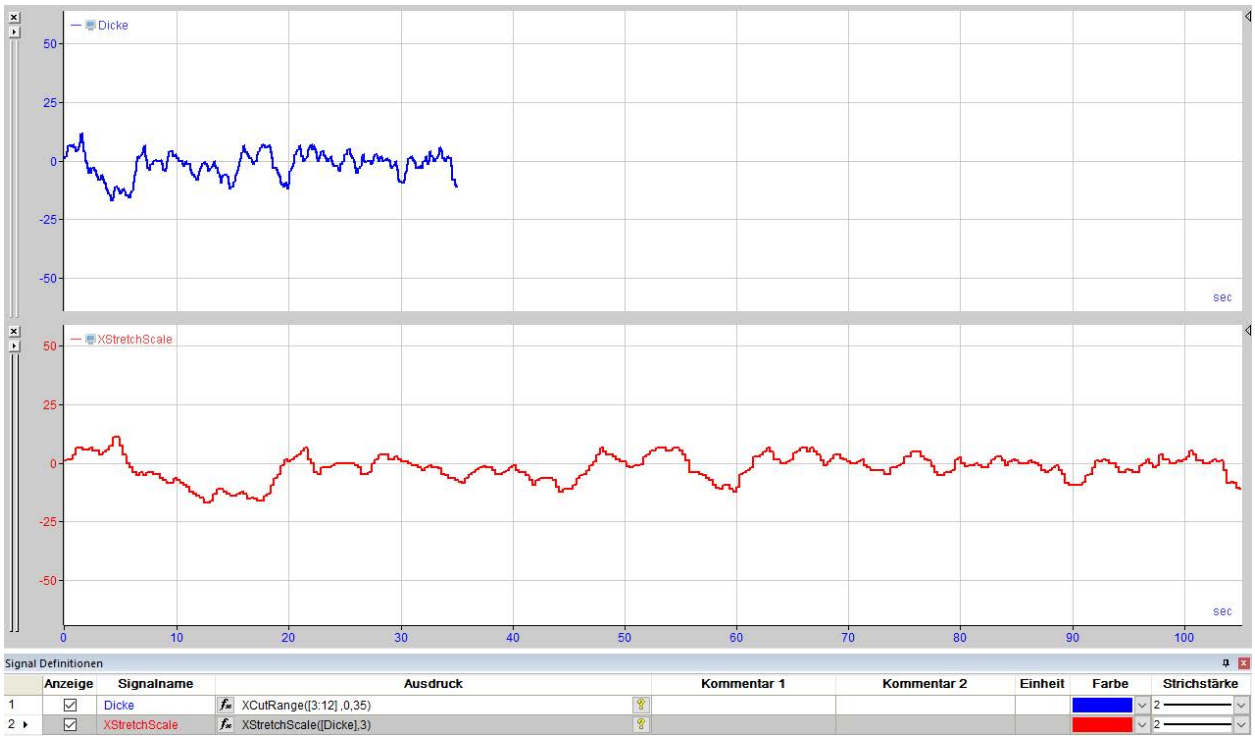
Im Bild oben wird die Walzkraftkurve vom ersten Stich (blaue Kurve) auf die Endlänge gemäß dem neunten Stich gedehnt.

XStretchScale

```
XStretchScale('Expression','Scale')
```

Beschreibung

Mit dieser Funktion kann der Kurvenzug eines Signals um einen festgelegten Faktor gestreckt werden. Der Skalierungsfaktor wird auch dann angewendet, wenn der Kurvenzug bereits mit einem Offset versehen ist.



Signal Definitionen

	Anzeige	Signalname	Ausdruck	Kommentar 1	Kommentar 2	Einheit	Farbe	Strichstärke
1	☑	Dicke	XCutRange([3:12] ,0,35)					2
2	☑	XStretchScale	XStretchScale([Dicke],3)					2

8.5 XFirst / XLast / XNow

XFirst

```
XFirst('Expression','Skip=0','SkipInitialEdge=FALSE')
```

Argumente

'Expression'	(boolesches) Eingangssignal
'Skip'	Optional zum Überspringen der ersten steigenden Flanken
'SkipInitialEdge'	Entscheidet, ob das erste Sample als steigende Flanke gezählt wird, falls es WAHR ist

Beschreibung

Diese Funktion liefert als Ergebnis einen Wert auf der X-Achse (Zeit [s] oder Weg [m]), für den der Ausdruck 'Expression' zum ersten Mal TRUE ist. Dementsprechend muss 'Expression' eine boolesche Größe sein. Das kann ein digitales Eingangssignal sein, das Ergebnis einer Vergleichsoperation oder jeder andere binäre Ausdruck.

XLast

```
XLast('Expression','Skip=0','SkipFinalEdge=FALSE')
```

Argumente

'Expression'	(boolesches) Eingangssignal
'Skip'	Optional zum Überspringen der letzten fallenden Flanken
'SkipFinalEdge'	Entscheidet, ob das letzte Sample als fallende Flanke gezählt wird, falls es WAHR ist

Beschreibung

Diese Funktion liefert als Ergebnis einen Wert auf der X-Achse (Zeit [s] oder Weg [m]), für den der Ausdruck 'Expression' zum letzten Mal TRUE ist. Dementsprechend muss 'Expression' eine boolesche Größe sein. Das kann ein digitales Eingangssignal sein, das Ergebnis einer Vergleichsoperation oder jeder andere binäre Ausdruck.

XNow

```
XNow()
```

Beschreibung

Diese Funktion gibt die relative Zeit seit dem letzten Start von *ibaAnalyzer* zurück.

8.6 XSize / XSumValid

XSize

```
XSize ('Expression')
```

Beschreibung

Diese Funktion liefert als Ergebnis die Gesamtlänge von 'Expression' in Einheiten der X-Achse (Zeit in [s] oder Weg in [m]). Das Ergebnis ist konstant 0, wenn das Eingangssignal ungültig ist.

XSumValid

```
XSumValid ('Expression')
```

Beschreibung

Mit dieser Funktion wird die Dauer oder die Länge ermittelt, für die die Bedingung 'Expression' TRUE ist. Alle Messpunkte, bei denen die Bedingung nicht erfüllt ist (FALSE) bleiben bei der Berechnung unberücksichtigt. Dementsprechend muss 'Expression' eine boolesche Größe sein. Das kann ein digitales Eingangssignal sein, das Ergebnis einer Vergleichsoperation oder jeder andere binäre Ausdruck.

Das Ergebnis ist konstant 0, wenn das Eingangssignal ungültig ist.

8.7 XValues / YValues

XValues

```
XValues('Expression')
```

Beschreibung

Diese Funktion liefert als Ergebnis die X-Werte aller Messpunkte eines Ausdrucks zurück. Das Besondere an dieser Funktion ist, dass sie auch mit Signalen bzw. Ausdrücken arbeitet, die nicht zeitbasiert sind, also als Basis Länge (m), Frequenz (Hz) oder inverse Länge (1/m) haben.

Bei einem normalen zeit- oder wegkontinuierlichen Signalverlauf ist das Ergebnis eine steigende Gerade, mit den Basiseinheiten (Zeit- oder Wegwerte) auf der Y-Achse in s oder m. Die Funktion arbeitet auch mit nicht-äquidistanten Messwerten.

YValues

```
YValues('Expression','TimeBase=1')
```

Argumente

'Expression'	Eingangssignal
'TimeBase'	Zeitbasis des Ausgangssignals

Beschreibung

Diese Funktion liefert als Ergebnis die Y-Werte aller Messwerte eines Ausdrucks zurück. Unabhängig davon, ob das Eingangssignal äquidistant gesampelt ist oder nicht, ist das Ergebnis ein äquidistantes Signal mit Zeitbasis 'TimeBase'.

Die Angabe der Zeitbasis ist optional, und als Standardwert wird 'TimeBase'=1 verwendet.

8.8 VarDelay

```
VarDelay('Expression','Delay')
```

Argumente

'Expression'	Eingangssignal
'Delay'	Verzögerungszeit in Sekunden

Beschreibung
Diese Operation liefert als Ergebnis den Ausdruck ‘Expression’ um eine Zeitkonstante 'Delay' verzögert zurück.

8.9 XY

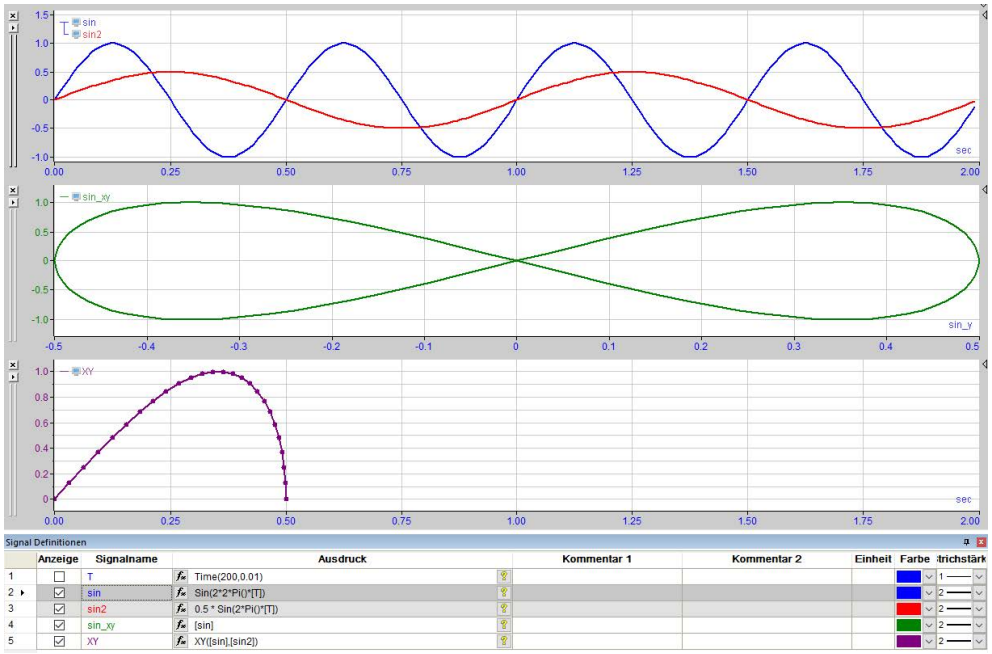
```
XY('Expression1','Expression2','Precision')
```

Argumente

'Expression1'	Eingangssignal, welches die X-Werte des neuen Signals enthält
'Expression2'	Eingangssignal, welches die Y-Werte des neuen Signals enthält
'Precision'	Optionaler Parameter zur Angabe der Abtastrate des Ausgangssignals

Beschreibung
Diese Funktion wird verwendet, wenn das Resultat aus der X-Y-Darstellung für weitere Operationen genutzt werden soll. Nach erfolgter Auswahl werden der Funktion die Signale der X- und Y-Achse zugewiesen.

Zu beachten ist, dass in der resultierenden Funktion die Abstände zwischen den Signalpunkten nicht gleich den Abständen der Ursprungssignale sind. Auch der Abstand zwischen den Signalpunkten selbst ist unterschiedlich. Mit dem Parameter 'Precision' kann ein fester Abstand zwischen den Signalpunkten eingestellt werden. Wird kein Parameter eingegeben, wird für alle Folgeoperationen der kürzeste Abstand der Signalpunkte als fester Wert verwendet.



8.10 XMarker1 / XMarker2

```
XMarker1 () bzw. XMarker2 ()
```

Beschreibung

Diese Funktion liefert als Ergebnis die Position des Markers X1 bzw. X2 auf der X-Achse.

8.11 XBase / XOffset

XBase

```
XBase('Expression')
```

Beschreibung

Mit dieser Funktion können Sie die Aufzeichnungszeitbasis bzw. längen- oder frequenzbasierte Abstände zwischen den Samples ermitteln. Die Funktion liefert bei einem äquidistant abgetasteten Signal den Abstand zwischen zwei Messpunkten in X-Achseneinheiten.

Bei einem Signal, dessen Samples nicht den gleichen Abstand haben, wird der Abstand in X-Achseneinheiten ausgegeben, der bei einem Resampling auf äquidistante Samples ermittelt werden würde. Standardmäßig ist das der kleinste Abstand zwischen zwei Samples des Signals.

XOffset

```
XOffset('Expression')
```

Beschreibung

Diese Funktion liefert den zeitlichen Abstand des ersten Messpunktes eines Signals vom Beginn der Messdatei in Sekunden. Das Ergebnis ist negativ, wenn das Signal früher beginnt und positiv, wenn es später beginnt.

Wenn mehrere Messdateien gleichzeitig geöffnet sind und die Option "*Messdateien synchronisieren mit Aufnahmezeit*" aktiviert ist, dann wird der Offset nicht notwendigerweise mit Bezug auf den Beginn der Messdatei des gewählten Signals ermittelt, sondern auf den Beginn der Messdatei, die den frühesten Startzeitpunkt hat.

8.12 FillGaps

```
FillGaps('Expression')
```

Beschreibung

Mit Hilfe der Funktion *FillGaps* können Lücken in einem Signal 'Expression' durch linear interpolierte Einträge gefüllt werden.

Diese Funktion ist insbesondere nützlich, wenn bei Trendabfragen aus einer Datenbank durch NULL-Einträge Lücken entstehen.

8.13 XAlignFft

```
XAlignFft('Expression_fixed','Expression_shift','Start','End','MinScale','MaxSca-
le','scale','type')
```

Argumente

'Expression_fixed'	Ausdruck, der als Referenz dient
'Expression_shift'	Ausdruck, welcher der Referenz angeglichen wird
'Start'	
'End'	
'MinScale'	Der kleinste x-Skalierungsfaktor, der überprüft werden soll
'MaxScale'	Der größte x-Skalierungsfaktor, der überprüft werden soll
'scale'	
'type'	

Beschreibung
Mit dieser Funktion können längenbasierte Signale mit derselben physikalischen Bedeutung, die an verschiedenen Stellen im Prozess gemessen werden, zueinander ausgerichtet werden.

Einige Parameter werden nachfolgend genauer beschrieben.

- 'Expression_fixed'

Eine Dickenmessung, die im Laufe des Algorithmus als “fest”, also nicht skalierbar oder verschiebbar betrachtet wird. Dies sollte die Dickenmessung sein, die das Profil der anderen Messung enthält. (Im Warmband-Kaltbandvergleich also das Warmband)

- 'Expression_shift'

Auf diese Dickenmessung bezieht sich später das Ergebnis des Alignments. Diese Messung muss also mit den Ergebniswerten skaliert und verschoben werden.

- 'Start'

Das Intervall von Start bis Ende gibt den X-Achsenabschnitt an, in dem die Messung 'Expression_fixed' verschoben werden darf. Der Nullpunkt ist hierbei der Nullpunkt des Ausdrucks. Es sind auch negative Werte erlaubt. Wenn die Messung 'Expression_shift' auf der linken Seite in *ibaAnalyzer* 10 Achseneinheiten im Vergleich zu 'Expression_fixed' herausragen darf, so muss Start = -10 sein.

- 'End'

Gibt das Ende des eben beschriebenen Intervalls an. Es empfiehlt sich, dieses Ende in Abhängigkeit von der Länge von 'Expression_fixed' zu wählen. Also z. B. End = XSize([Expression_fixed]) oder End = 1.2 * Xsize([Expression_fixed]), falls ein Überhang von 20 Prozent erlaubt sein soll.

- 'scale'

Mit diesem Parameter lässt sich das Verhältnis zwischen Genauigkeit und Geschwindigkeit steuern. Je kleiner der Wert, desto langsamer und verlässlicher arbeitet der Algorithmus. Je höher der Wert, desto mehr wird der Algorithmus durch eine Heuristik beschleunigt. Bei zu hohen Werten für 'scale' kann dies jedoch zu einem falschen Ergebnis führen. Für ein optimales Ergebnis empfiehlt es sich, die Auflösung der Messdaten zu übergeben. Haben die Messpunkte z. B.

einen Abstand von 10 cm, sollte scale = 0.1 sein. Falls die Ergebnisberechnung zu langwierig ist, kann der Wert dann nach oben korrigiert werden.

9 Vektor-Operationen

Vektor-Operationen erweitern die Analysemöglichkeiten für zweidimensionale Signale.

Vektoren, in früheren Beschreibungen auch oft als Arrays bezeichnet, können auf unterschiedliche Arten gebildet werden:

- Durch Gruppierung mehrerer Signale in *ibaPDA* und Kennzeichnung der Gruppe als "Vektor"
- Durch Zusammenstellung mehrerer Signale in den Logischen Signaldefinitionen in *ibaAnalyzer*
- Als Ergebnis verschiedener Berechnungsfunktionen, z. B. FFT-Funktionen

Vektoren können in *ibaAnalyzer* in der 2D-Draufsicht und in der 3D-Ansicht dargestellt werden. Die Vektor-Operationen im Ausdruckseditor dienen der Nutzung der Vektordaten für weiterführende Berechnungen.

Andere Dokumentation

Ausführliche Hinweise zu diesen Darstellungen und ihren Einstellungen finden Sie im *ibaAnalyzer*-Handbuch, Teil 2, Darstellungsarten.

9.1 GetFirstIndex / GetLastIndex

```
GetFirstIndex('Condition') bzw. GetLastIndex('Condition')
```

Beschreibung
Diese Funktionen liefern den Index des ersten bzw. letzten Kanals im Vektor zurück, für den die Bedingung ‘Condition‘ wahr ist. Der Vektor selbst muss dabei ein Operand von ‘Condition‘ sein. Wenn ‘Condition‘ FALSE für alle Kanäle des Vektors ergibt, dann liefert die Funktion -1 als Ergebnis.

9.2 GetRows

```
GetRows('Vector','StartIndex','Counter','Step')
```

Argumente

'Vector'	Vektor, aus dem einzelne Einträge ausgelesen werden sollen
'StartIndex'	Erster Index, aus dem Einträge ausgelesen werden sollen
'Counter'	Anzahl der Einträge
'Step'	Schrittgröße

Beschreibung
Diese Funktion liest Wertereihen aus einem Vektor aus. Dabei wird ausgehend von einem ‘StartIndex‘ (der kleinste mögliche Index ist 0) in Schritten der Größe ‘Step‘ eine Gesamtzahl ‘Counter‘ an Einträgen ausgelesen.

9.3 GetZoneCenters

```
GetZoneCenters('Vector')
```

Beschreibung
Diese Funktion ermittelt die Position der Zonenmitte auf der Y-Achse für jede Zone des Vektors. Einziges Argument der Funktion ist der Vektor. Das Ergebnis ist wieder ein Vektor mit einer Anzahl Werte entsprechend der Zonenanzahl.

Beispiel
Die Funktion *GetZoneCenters* ist besonders hilfreich, wenn sie auf das Ergebnis einer *FftInTime*-Funktion angewendet wird. Die *FftInTime*-Funktion liefert als Ergebnis einen Vektor mit *n* "Zonen", die den Frequenzbändern (Bins) entsprechen. Mit der Funktion *GetZoneCenters* können die Mittenfrequenzen der einzelnen Bänder des Spektrums und somit der Frequenzvektor bestimmt werden. Damit ist es dann möglich in der Frequenzdomäne zu differenzieren oder zu integrieren, indem die Ergebnisse der *FftInTime*- und der *GetZoneCenters*-Funktion entsprechend multipliziert oder dividiert werden.

9.4 GetZoneOffset

```
GetZoneOffset(‘Vector‘)
```

Beschreibung
Diese Funktion ermittelt den Offset der ersten Zone, d. h. die Position der Zonenmitte der ersten Zone, bezogen auf die Nulllinie der Y-Achse. Einziges Argument der Funktion ist der Vektor. Das Ergebnis ist ein konstanter Wert.

9.5 GetZoneWidths

```
GetZoneWidths('Vector')
```

Beschreibung
Diese Funktion ermittelt die Breite jeder Zone des Vektors in Einheiten der Y-Achse. Einziges Argument der Funktion ist der Vektor. Das Ergebnis ist wieder ein Vektor mit einer Anzahl Werte entsprechend der Zonenanzahl.

9.6 MakeVector

```
MakeVector(r_0,r_1,...,r_n)
```

Beschreibung
Diese Funktion erzeugt einen Vektor mit den Wertereihen *r_0* bis r_n. Die Argumente können konstante Werte oder Signale bzw. Ausdrücke sein. Das ist vergleichbar mit der Erzeugung eines Vektors im Dialog der *Logischen Signaldefinitionen*.

Beispiel
Die Funktion *MakeVector* dient hauptsächlich dazu, um es Makros zu ermöglichen mehrdimensionale Signale als Ergebnis auszugeben. Im Makro-Editor können die Teilergebnisse verschiedener Berechnungen innerhalb des Makros als Zwischenwerte deklariert werden. Als finales Ergebnis des Makros kann dann ein Vektor definiert werden, dessen Argumente die Zwischenwerte sind. Der Vektor wird quasi als Behälter für Makroergebnisse genutzt, um die Makroschnittstelle zu vereinfachen.

9.7 SetZoneWidths

```
SetZoneWidths('Vector','Widths','Offset')
```

Argumente

'Vector'	Vektor mit den (Mess-)Werten des Ergebnisvektors
'Widths'	Vektor, der als Werte die Zonenbreiten enthält
'Offset'	Abstand der Zonenmitte der ersten Zone von der Nulllinie

Beschreibung
Diese Funktion erzeugt einen Vektor mit vorgegebenen Zonenbreiten. Dabei werden die Werte des Ergebnisvektors aus einem Vektor 'Vector' und die Zonenbreiten aus einem Vektor 'Widths' entnommen. Da der Vektor mit den Zonenbreiten seinerseits Ausdrücke als Argumente verwenden kann, lassen sich mit dieser Funktion Vektoren mit verschiedenen Zonenbreiten in Abhängigkeit der geladenen Daten erzeugen. Die Ausdrücke zur Definition der Zonenbreiten sollten dabei konstant über die Zeit sein und sich für einmal geladene Daten nicht mehr ändern. Sollte dies nicht der Fall sein, werden die Breitenwerte über den Gesamtzeitraum gemittelt.

Beispiel
Die Funktion *SetZoneWidths (MakeVector(1,2,3,2,1),MakeVector(2,4,10,4,2), -10)* erzeugt den gleichen Vektor wie den, der mit den logischen Signaldefinitionen erzeugt wurde, siehe *ibaAnalyzer*-Handbuch Teil 2, Kap. *Logische Signaldefinitionen*).

9.8 VectorAvg

```
VectorAvg('Vector')
```

Beschreibung
Diese Funktion berechnet für jedes Sample den Mittelwert des Querprofils, d. h. den Mittelwert aller Vektorspuren pro Zeitpunkt bzw. pro X-Achsenposition. Als Ergebnis liefert die Funktion ein eindimensionales Signal, das den Verlauf des Querprofil-Mittelwerts über die Zeit/Länge des Vektorsignals mit gleicher Anzahl Samples zeigt.

9.9 VectorKurtosis

```
VectorKurtosis('Vector')
```

Beschreibung
Diese Funktion berechnet für jedes Sample die Kurtosis (Wölbung) des Querprofils. Als Ergebnis liefert die Funktion ein eindimensionales Signal, das den Verlauf der Querprofil-Wölbung über die Zeit/Länge des Vektorsignals mit gleicher Anzahl Samples zeigt.

Hinweis

Es müssen mindestens 4 Signalpunkte vorhanden sein, d.h. der Vektor muss mindestens 4 Eingänge haben.

9.10 VectorMarkRange

```
VectorMarkRange('Vector','Start','End')
```

Argumente

'Vector'	Vektor mit den Eingangssignalen
'Start'	Anfang des auszuwählenden Bereichs
'End'	Ende des auszuwählenden Bereichs

Beschreibung
Diese Funktion liefert als Ergebnis einen Teilvektor von 'Vector' mit einer Zonenbreite von 'Start' (untere Kante) bis 'End' (obere Kante).

Die Angabe der Positionen muss in Einheiten der Y-Achse erfolgen. Die Positionen können sowohl feste Werte als auch Signale oder Ausdrücke und somit abhängig von den geladenen Daten sein.

Die Ausdrücke zur Definition der Positionen sollten dabei konstant über die Zeit sein und sich für einmal geladene Daten nicht mehr ändern. Sollte dies nicht der Fall sein, werden die Positionswerte über den Gesamtzeitraum gemittelt.

9.11 VectorMin / VectorMax

VectorMax

```
VectorMax('Vector')
```

Beschreibung
Diese Funktion berechnet für jedes Sample das Maximum des Querprofils, d. h. den Maximalwert aller Vektorspuren pro Zeitpunkt bzw. pro X-Achsenposition. Als Ergebnis liefert die Funktion ein eindimensionales Signal, das den Verlauf des Querprofil-Maximums über die Zeit/Länge des Vektorsignals mit gleicher Anzahl Samples zeigt.

VectorMin

```
VectorMin('Vector')
```

Beschreibung

Diese Funktion berechnet für jedes Sample das Minimum des Querprofils, d. h. den Minimalwert aller Vektorspuren pro Zeitpunkt bzw. pro X-Achsenposition. Als Ergebnis liefert die Funktion ein eindimensionales Signal, das den Verlauf des Querprofil-Minimums über die Zeit/Länge des Vektorsignals mit gleicher Anzahl Samples zeigt.

9.12 VectorPercentile

```
VectorPercentile('Vector','Percentile'=0.5)
```

Argumente

'Vector'	Vektor mit den Eingangssignalen
'Percentile'	Die Perzentile, zwischen 0 und 1

Beschreibung

Diese Funktion berechnet für jedes Sample die Perzentile des Querprofils.

Das zweite Argument neben dem 'Vector' ist die Angabe der 'Percentile', die berechnet werden soll. Default-Wert ist 0.5 (Median). Als Ergebnis liefert die Funktion ein eindimensionales Signal, das den Verlauf der Querprofil-Perzentilen über die Zeit/Länge des Vektorsignals mit gleicher Anzahl Samples zeigt.

9.13 VectorSkewness

```
VectorSkewness('Vector')
```

Beschreibung

Diese Funktion berechnet für jedes Sample die *Skewness* (Schiefe) des Querprofils. Als Ergebnis liefert die Funktion ein eindimensionales Signal, das den Verlauf der Querprofil-Schiefe über die Zeit/Länge des Vektorsignals mit gleicher Anzahl Samples zeigt. Zu beachten ist, dass für diese Berechnung mindestens 4 Signalpunkte vorhanden sein müssen.

9.14 VectorStdDev

```
VectorStdDev('Vector')
```

Beschreibung

Diese Funktion berechnet für jedes Sample die Standardabweichung des Querprofils. Als Ergebnis liefert die Funktion ein eindimensionales Signal, das den Verlauf der Standardabweichung über die Zeit/Länge des Vektorsignals mit gleicher Anzahl Samples zeigt.

9.15 VectorSum

```
VectorSum('Vector')
```

Beschreibung
Diese Funktion berechnet für jedes Sample die Summe aller Werte des Querprofils. Als Ergebnis liefert die Funktion ein eindimensionales Signal, das den Verlauf der Wertesumme im Querprofil über die Zeit/Länge des Vektorsignals mit gleicher Anzahl Samples zeigt.

Beispiel
Dividiert man den *VectorSum*-Ausdruck durch die Anzahl der Vektorspuren, dann erhält man das gleiche Ergebnis wie mit der Funktion *VectorAvg*.

9.16 VectorToSignal / SignalToVector

VectorToSignal

```
VectorToSignal('Vector','XBase')
```

Argumente

'Vector'	Vektor mit den (möglichst konstanten) Eingangssignalen
'XBase'	Abtastrate des Ausgangssignals

Beschreibung
Diese Funktion erzeugt entlang des Querprofils aus den Elementen eines Vektors ein eindimensionales Signal. Jedes Sample des resultierenden Signals entspricht einem Element des Vektors. Das Ergebnis entspricht dem Querprofil.

Der Parameter 'XBase' ist optional. Wird 'XBase' nicht angegeben, dann werden die Zonenbreiten und der Offset des Vektors verwendet. Das resultierende Signal kann somit auch nicht-äquidistante Samples erhalten.

Beispiel
In Verbindung mit den Funktionen *YatX* und der Markerposition kann die Funktion *VectorToSignal* dazu genutzt werden, an einer beliebigen Stelle im Vektor das Querprofil auszugeben:

VectorToSignal (YatX([Vektor],XMarker1()))

SignalToVector

```
SignalToVector('Signal')
```

Beschreibung
Im Gegensatz zur *VectorToSignal* Funktion erzeugt die Funktion *SignalToVector* einen Vektor mit konstanten Einträgen aus dem Signal 'Signal‘. Die Zonenbreite und Offset wird dabei durch die Abtastrate des Eingangssignals bestimmt. Beachten Sie, dass diese Funktion im Gegensatz zu *VectorToSignal* kein optionales Argument besitzt, um die Zonenbreiten zu bestimmen. Dazu kann die Funktion *SetZoneWidth* verwendet werden.

Hinweis

Das Eingangssignal sollte weniger als 1000 Samples haben, ansonsten wird das Signal nicht ausgewertet und als zu komplex markiert.

9.17 Traverse / TraverseW

Traverse

```
Traverse('Signal','Position','N'=40,'Min','Max','Avg'=1)
```

Argumente

'Signal'	Das von einem traversierenden Messgerät gemessene Signal
'Position'	Die Position des traversierenden Messgeräts entlang des Querprofils
'N'	Anzahl der Zonen des resultierenden Vektors
'Min'	Optionaler Threshold für das Minimum des abzubildenden Bereichs
'Max'	Optionaler Threshold für das Maximum des abzubildenden Bereichs
'Avg'	Optionaler, binärer Parameter, um mehrere Samples innerhalb eines Zonendurchgangs zu mitteln; per Default wird der Mittelwert gebildet

Beschreibung

Diese Funktion wandelt Signale, die von einem traversierenden Messgerät stammen, in einen Vektor zur zweidimensionalen Darstellung um.

TraverseW

```
TraverseW('Signal','Position','Widths','Offset'=0,'Avg'=1)
```

Argumente

'Signal'	Das von einem traversierenden Messgerät gemessene Signal
'Position'	Die Position des traversierenden Messgeräts entlang des Querprofils
'Widths'	Die Breite der resultierenden Zonen
'Offset'	Optionaler Versatz der ersten Zone
'Avg'	Optionaler, binärer Parameter, um mehrere Samples innerhalb eines Zonendurchgangs zu mitteln; per Default wird der Mittelwert gebildet

Beschreibung

Diese Funktion funktioniert analog zu *Traverse*, mit dem Unterschied, dass die Dimensionen des resultierenden Vektors direkt über die Zonenbreiten 'Widths' und einen optionalen Offset-Parameter gesetzt werden.

9.18 VectorPolynomial / VectorLSQPolyCoef

VectorPolynomial

```
VectorPolynomial('Coefs','Vector','PolynomialType' = 0)
```

Argumente

'Coefs'	Koeffizienten des Interpolationspolynoms; berechnet mit *VectorLSQPolyCoef*
'Vector'	Stützstellen zur Auswertung des Interpolationspolynoms
'PolynomialType'	Definiert die Basispolynome, die verwendet werden (0 = Lagrange (default), 1 = Chebyshev I, 2 = Chebyshev II, 3 = Legendre)

Beschreibung

Diese Funktion kann genutzt werden, um das Interpolationspolynom, welches durch die Koeffizienten 'Coefs' als Ergebnis der Funktion *VectorLSQPolyCoef* beschrieben wird, darzustellen.

Die Stützstellen zur Auswertung des Polynoms werden durch die Abtastpunkte des Vektors 'Vector' festgelegt. Falls Zonen Offset und/oder Zonenbreite spezifiziert sind, werden diese auch benutzt, ansonsten werden die Indizes als Y-Werte herangezogen.

Der optionale Parameter 'PolynomialType' kann verwendet werden, um verschiedene Basispolynome zu verwenden.

Hinweis

Beachten Sie, dass die Einträge von 'Vector' hier keine Rolle spielen.

VectorLSQPolyCoef

```
VectorLSQPolyCoef('Vector','Degree','PolynomialType' = 0)
```

Argumente

'Vector'	Vektor, für dessen Einträge die Least Squares Approximationspolynome berechnet werden
'Degree'	Grad des Interpolationspolynoms
'PolynomialType'	Definiert die Basispolynome, die verwendet werden (0 = Lagrange (default), 1 = Chebyshev I, 2 = Chebyshev II, 3 = Legendre)

Beschreibung

Diese Funktion ist die Erweiterung der Funktion *LSQPolyCoef* auf Vektoren. Dabei werden die Koeffizienten eines Interpolationspolynoms vom Grad 'Degree' mit Hilfe der Methode der kleinsten Quadrate für jedes Querprofil berechnet. Als Basis werden hierzu die Indizes des Vektors herangezogen, es sei denn ein Zonen Offset und/oder die Zonenbreite wurden bei der Erstellung des Vektors gesetzt. In diesem Fall werden die entsprechenden Werte als Basis verwendet.

Der optionale Parameter 'PolynomialType' kann verwendet werden, um verschiedene Basispolynome zu verwenden.

10 Text-Funktionen

10.1 InfofieldText / ChannelInfoFieldText / ModuleInfoFieldText

Diese Funktionen erlauben es, Informationen aus einem Info-Feld einer Messdatei, eines Kanals oder eines Moduls, als Textkanal zur Verfügung zu stellen.

Argumente

'Index'	Index der Datei bzw. des Kanals oder Moduls
'InfoField'	Das Infofeld, welches ausgelesen werden soll; Muss in Anführungszeichen gesetzt werden!
'Start'	Erstes Zeichen des Feldinhalts, das ausgelesen werden soll (optional) Wird kein Wert angegeben, dann wird der komplette Inhalt ausgelesen
'End'	Letztes Zeichen des Feldinhalts, das ausgelesen werden soll (optional) Wird kein Wert angegeben, wird von 'Start' bis zum letzten Zeichen ausgelesen

Hinweis

Bei der Funktion *ModuleInfoFieldText* müssen zwei Indizes angegeben werden. Der Index der Messdatei als erstes Argument und der Index des Moduls als zweites. Alle anderen Argumente bleiben gleich.

InfoFieldText

```
InfofieldText('FileIndex','InfoField','Start','End')
```

Beschreibung

Diese Funktion gibt den Inhalt eines Infofeldes als Textkanal aus.

Tipp

Wenn Sie im Signalbaum einen Doppelklick auf das gewünschte Infofeld machen, dann fügt *ibaAnalyzer* die entsprechende Funktion automatisch als neues Signal in die Signaltabelle ein. Anschließend brauchen Sie bei Bedarf nur noch den Signalnamen und Anfang/Ende anpassen.

Wenn Sie den Inhalt eines Infofeldes als Zahlenwert auslesen wollen, verwenden Sie die Funktion *Infofield*.

ChannelInfoFieldText

```
ChannelInfofieldText('ChannelIndex','InfoField','Start','End')
```

Beschreibung

Diese Funktion gibt den Inhalt des Infofeldes eines Kanals als Text aus.

ModuleInfoFieldText

```
ModuleInfoFieldText('FileIndex','ModuleIndex','InfoField','Start','End')
```

Argumente

'FileIndex'	Index der Datei, zu der das Modul gehört
'ModuleIndex'	Der Index des Moduls
'InfoField'	Das Infofeld, welches aus dem Modul gelesen werden soll
'Start'	Start Index des Info-Strings
'End'	Ende Index des Info-Strings

Beschreibung

Diese Funktion arbeitet wie die Funktionen *InfoFieldText* und *ChannelInfoFieldText*, jedoch bezieht sie sich auf die Infofelder eines Moduls und nicht der Messdatei oder Signals. Die Funktion liefert als Ergebnis einen Textkanal mit dem Inhalt des spezifizierten Infofelds.

10.2 TextCompare / CompareText

```
TextCompare('Text1','Text2','CaseSensitive=True')
```

Argumente

'Text1/2'	Die beiden Strings, die verglichen werden sollen
'CaseSensitive'	Optionaler Parameter, mit dem Sie spezifizieren können, ob beim Vergleich Groß- und Kleinschreibung berücksichtigt werden soll.

Beschreibung

Diese Funktion erlaubt es Ihnen, Textinformationen lexikographisch miteinander zu vergleichen. Die Funktion arbeitet sowohl mit Inhalten von Textkanälen als auch mit Klartext, der – versehen mit Anführungszeichen – direkt in die Signaldefinition eingetragen wird.

Vergleich und Ergebnisse:

- Das Ergebnis ist -1, wenn die Information des ersten Textes lexikographisch vor der des zweiten Textes anzuordnen ist.
- Das Ergebnis ist 0, wenn beide Texte die gleiche Information haben.
- Das Ergebnis ist +1, wenn die Information des ersten Textes lexikographisch nach der des zweiten Textes anzuordnen ist.

Beispiel
Den Einfluss des Parameters ‘CaseSensitive‘ zeigt die folgende Tabelle (Beispiele):

Text1	Text2	Ergebnis		Anmerkung
		TextCompare ("Text1","Text2",0)	TextCompare ("Text1","Text2",1)	
1234 abcd	1234 abcd	0	0	1 = 2
1234 abcd	1234 bcde	-1	-1	1 < 2 "a" steht vor "b"
1234 Abcd	1234 abcd	0	1	1 = 2 (Groß-/Kleinschreibung nicht berücksichtigt) 1 > 2 (Groß-/Kleinschreibung berücksichtigt) "A" steht nach "a"
12340 abcd	1234 abcd	1	1	1 > 2 "0" steht nach " "
1234 0abcd	1234 abcd	-1	-1	1 < 2 "0" steht vor "a"
12034 abcd	1234 abcd	-1	-1	1 < 2 "0" steht vor "3"
1234 abcd	1y34 abcd	-1	-1	1 < 2 "2" steht vor "y"
1z34 abcd	1Y34 abcd	1	1	1 > 1 "z" steht nach "Y"

10.3 ToText / FromText

ToText

```
ToText('Expression','Format'="%g",'datatype'=0)
```

Argumente

'Expression'	Ausdruck, dessen Inhalt in einen Textkanal konvertiert werden soll
'Format'	Optionaler Parameter für einen Formatstring
'datatype'	Optionaler Parameter, der das Gleitkommaformat festlegt - 'datatype'=0 Default Wert - 'datatype'=1 Ausgabe wird als signed 16 bit integer formatiert - 'datatype'=2 Ausgabe wird als unsigned 16 bit integer formatiert - 'datatype'=3 Ausgabe wird als signed 32 bit integer formatiert - 'datatype'=4 Ausgabe wird als unsigned 16 bit integer formatiert

Beschreibung

Diese Funktion wandelt einen numerischen Signalwert in einen Textkanal um. Dabei wird bei äquidistanten Samples des Eingangssignals 'Expression' und gleichbleibendem Y-Wert nur der Wert des ersten Signalpunkts als Sample im Textkanal eingetragen und angezeigt. Ändert sich der Y-Wert, wird für jeden neuen Y-Wert ein Sample im Textkanal eingetragen und angezeigt.

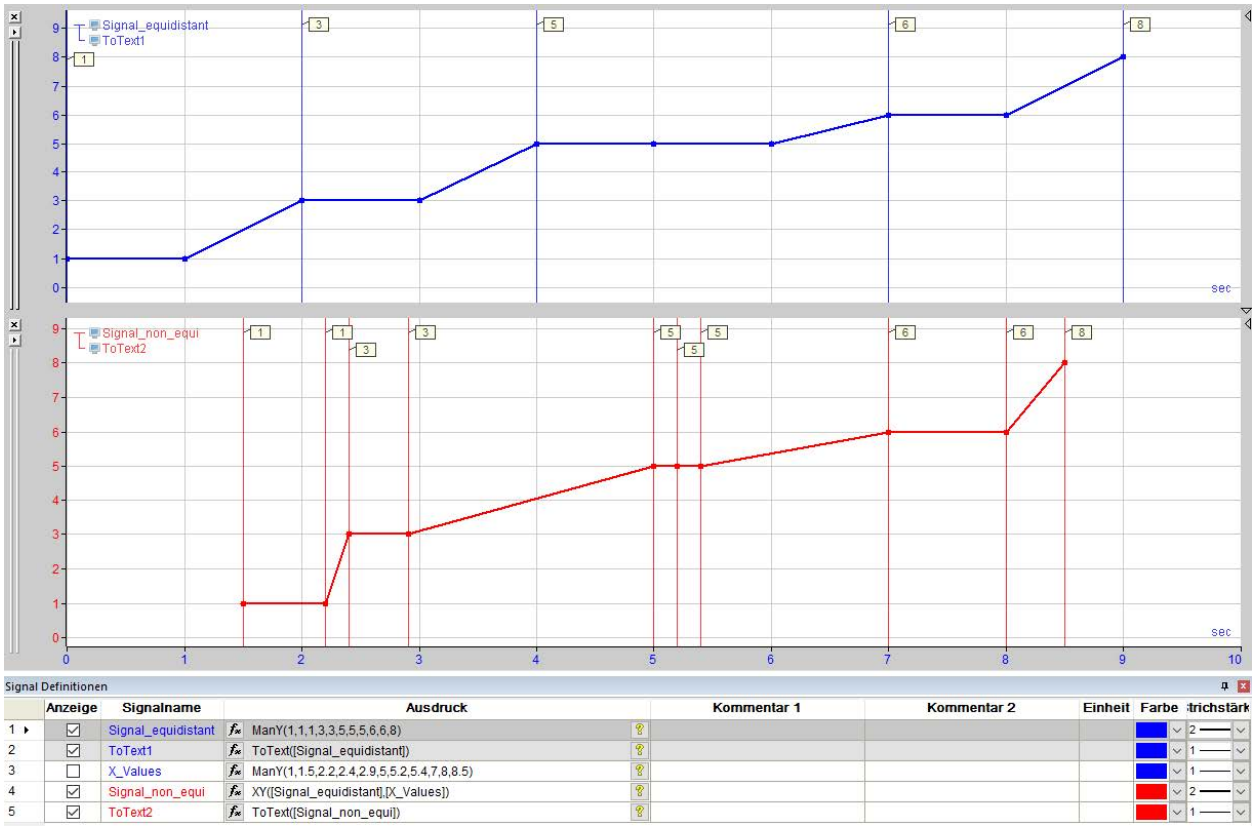
Enthält das Eingangssignal 'Expression' keine äquidistanten Samples, dann wird für jedes Eingangssample auch ein Sample im Textkanal eingetragen und angezeigt.

Der optionale Parameter 'Format' ist gemäß C printf-Syntax einzugeben. Sie können nur einen Parameter (%) angeben, der einem IEEE 32-Bit Gleitkommawert entsprechen muss. Default-Wert ist %g. Dieser Wert wird auch verwendet, wenn Sie den optionalen Parameter nicht angeben.

Beispiele:
%g = Wandel des Gleitkommawertes in einen Text
%.4f = Text/Zahl mit 4 Stellen nach dem Komma, etc.

Beispiel

Die *ToText*-Funktion kann z. B. hilfreich sein, wenn Trends visualisiert werden, die große Datenmengen beinhalten. Ohne ständig zwischen der Marker- und Signalansicht zu wechseln, können auf einfache Weise die numerischen Werte angezeigt werden. Die nachfolgende Abbildung zeigt eine Anwendung der Funktion *ToText*.



FromText

```
FromText('TextChannel','Start','End')
```

Argumente

'TextChannel'	Textkanal, der konvertiert werden soll
'Start'	Erstes Zeichen des Feldinhalts, das ausgelesen werden soll (optional) Wird kein Wert angegeben, dann wird der komplette Inhalt ausgelesen
'End'	Letztes Zeichen des Feldinhalts, das ausgelesen werden soll (optional) Wird kein Wert angegeben, wird von 'Start' bis zum letzten Zeichen ausgelesen

Beschreibung

Diese Funktion konvertiert den Inhalt des Textkanals ‚TextChannel' in einen numerischen Wert. Die Parameter ‚Start' und ‚End' können optional als Indizes genutzt werden um nicht den kompletten String umzuwandeln. Per Default wird ‚Start'=0 und ‚End'=Länge des Strings verwendet.

10.4 TrimText

```
TrimText('Text','RemoveOption=0')
```

Argumente

'Text'	Textkanal oder Ausdruck, aus dem Leerzeichen entfernt werden sollen.
'RemoveOption'	Parameter zur Einstellung der Betriebsart: 0 (default): Leerzeichen vor und nach dem Text entfernen 1: Nur Leerzeichen vor dem Text entfernen 2: Nur Leerzeichen nach dem Text entfernen 3: Alle Leerzeichen entfernen, auch im Text

Beschreibung
Mithilfe dieser Funktion können Sie aus Texten die Leerzeichen entfernen. Die Funktion kann sowohl auf Textkanäle, die bereits in der Messdatei enthalten sind, als auch auf Ergebnisse der Funktionen *InfofieldText* und *ToText* angewendet werden.

10.5 ConcatText

```
ConcatText('Text1','Text2',...)
```

Beschreibung
Mit dieser Funktion können mehrere Textkanäle zusammengefügt werden. Dabei sind auch numerische Signale zulässig. Diese werden automatisch in Text umgewandelt.

Falls die X-Positionen der einzelnen Kanäle nicht zueinander passen, wird für jede vorhandene X-Postion ein neuer Eintrag erstellt und der fehlende Eintrag durch den linken Nachbarn ersetzt, falls vorhanden.

Beispiel
Für aneinander gereihte Signale ist die File ID als Technostring-Kanal vorhanden. Für die Endprodukte gibt es einen Produkt Counter. Als Ergebnis sollen beide Informationen zusammengebracht werden.

10.6 CharValue

```
CharValue('Text','CharNumber'=0)
```

Argumente

'Text'	Eingabetext
'CharNumber'	Nullbasierte Position des zu bewertenden Zeichens.

Beschreibung
Diese Funktion liefert als Ergebnis den ASCII-Wert des Zeichens an der Position 'CharNumber' in 'Text'. Standardmäßig wird das erste Zeichen verwendet, das die 'CharNumber' Null hat.

10.7 CountText / TextLength

```
CountText('Text','CountOnlyDifferent'=0,'Reset'=0)
```

Argumente

'Text'	Eingabetext
'CountOnlyDifferent'	Optionaler Parameter zum Zählen von Texten, die sich vom vorherigen Textausdruck unterscheiden
'Reset'	Optionaler Parameter zum Zurücksetzen des Zählers

Beschreibung
Diese Funktion liefert als Ergebnis die Anzahl der Texte innerhalb eines Textkanals 'Text' zurück. Wenn der Parameter 'CountOnlyDifferent' auf True gesetzt ist, werden nur Texte gezählt, die vom vorherigen Textausdruck abweichen. Mit dem Digitalsignal 'Reset' kann der Zähler auf Null zurückgesetzt werden.

```
TextLength('Text')
```

Beschreibung
Diese Funktion liefert als Ergebnis die Anzahl der Zeichen in 'Text'.

10.8 DeleteText / InsertText / ReplaceText

```
DeleteText('Text','StartPos','Length')
```

Argumente

'Text'	Eingabetext
'StartPos'	Nullbasierter Index des ersten gelöschten Zeichens
'Length'	Anzahl der Zeichen, die gelöscht werden

Beschreibung
Diese Funktion löscht eine Anzahl von 'Länge' Zeichen aus dem 'Text'. Das erste Zeichen, das gelöscht wird, befindet sich an der Position 'StartPos' im Text.

```
InsertText('Text1','Text2','Pos')
```

Argumente

'Text1'	Originaltext
'Text2'	Text, der in den Originaltext eingefügt wird
'Pos'	Nullbasierte Zeichenposition, an der 'Text2' eingefügt wird

Beschreibung
Diese Funktion fügt 'Text2' in 'Text1' an der nullbasierten Zeichenposition 'Pos' ein. Wenn 'Pos' kleiner oder gleich Null ist, wird 'Text2' vor 'Text1' gestellt. Wenn 'Pos' größer oder gleich der Länge von 'Text1' ist, wird 'Text2' an 'Text1' angehängt.

```
ReplaceText('Text','SearchText','ReplaceText')
```

Argumente

'Text'	Eingabetext
'SearchText'	Text, der innerhalb von 'Text' ersetzt werden soll
'ReplaceText'	Wenn 'SearchText' gefunden wird, wird dieser durch 'ReplaceText' ersetzt

Beschreibung
Diese Funktion ersetzt alle Vorkommen von 'SearchText' innerhalb von 'Text' durch 'Replace-Text'.

10.9 MidText / FindText

```
MidText('Text','StartPos','Length')
```

Argumente

'Text'	Eingabetext
'StartPos'	Nullbasierter Index des ersten zu extrahierenden Zeichens aus dem Text
'Length'	Anzahl der Zeichen, die aus dem Text extrahiert werden sollen

Beschreibung
Diese Funktion liefert als Ergebnis eine Anzahl von Zeichen ('Length') aus dem 'Text'. Die resultierende Zeichenfolge beginnt mit dem Zeichen an der nullbasierten Position 'StartPos'.

```
FindText('Text','SearchText','CaseSensitive'=0,'N'=1)
```

Argumente

'Text'	Eingabetext
'SearchText'	Der Text, der innerhalb des Eingabetextes identifiziert werden muss
'CaseSensitive'	Optionaler Parameter zur Aktivierung der groß- und kleinschreibungsabhängigen Textsuche
'N'	Parameter, um nicht das erste, sondern das n-te Auftreten von 'SearchText' zu finden.

Beschreibung
Diese Funktion liefert als Ergebnis einen nullbasierten Index der Position des 'N'ten Auftretens von 'SearchText' im 'Text'. Wenn der Parameter 'CaseSensitive' auf True gesetzt ist, wird eine groß- und kleinschreibungsabhängige Suche durchgeführt. Wird 'SearchText' wird nicht gefunden, ist das Ergebnis -1.

11 Verschiedene Funktionen

11.1 Debounce

```
Debounce('Expression','Interval')
```

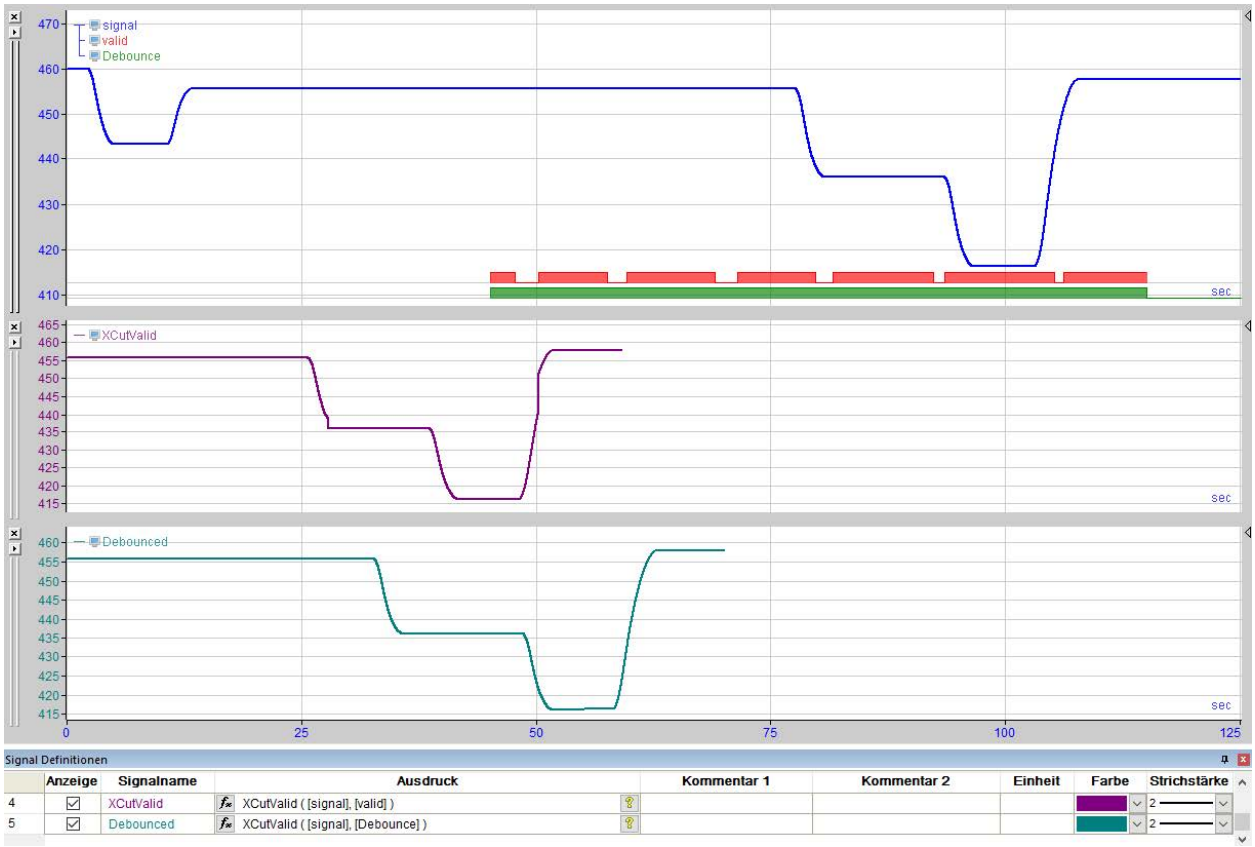
Argumente

'Expression'	Eingangssignal, welches entprellt werden soll
'Interval'	Totzeit

Beschreibung
Diese Funktion liefert als Ergebnis einen entprellten Signalverlauf von 'Expression' mit 'Interval' als Totzeit in [s]. Bei längenbezogenen Signalen wird 'Interval' als Weg in [m] interpretiert.

Die Funktion arbeitet ähnlich einem abfallverzögerten Zeitrelais, mit dem Unterschied, dass der Signalwechsel von TRUE nach FALSE (fallende Flanke) zeitrichtig, also unverzögert dargestellt wird, wenn innerhalb der eingestellten Zeit nicht wieder ein Wechsel von FALSE nach TRUE (steigende Flanke) folgt.

So können flatternde Signale, z. B. von Fotozellen oder Endschaltern beruhigt werden. Besonders wichtig ist dies, wenn diese Signale als Bedingungen in Operationen wie *XMarkValid* oder *XCutValid* verwendet werden, da bei jedem Aussetzer die Berechnung der Operation unterbrochen und damit Ergebniswerte verloren gehen würden. Der Unterschied ist im nachfolgenden Bild deutlich zu sehen.



11.2 Envelope

```
Envelope('Expression','Interval')
```

Argumente

'Expression'	Ausdruck, um den die Hüllkurve gebildet werden soll
'Interval'	X-Achsen Intervall

Beschreibung
Diese Funktion berechnet die obere Hüllkurve um ein Signal. Die Hüllkurve entsteht durch Verbindung der Maxima. Die Qualität der Hüllkurve kann mit dem Parameter 'Interval' beeinflusst werden. Ohne Angabe dieses Parameters werden nur die größten Maxima über die Aufzeichnungsdauer berücksichtigt. Mit dem Parameter 'Interval' geben Sie die Intervalllänge in Einheiten der X-Achse (s, m, Hz, 1/m) vor. Es werden dann auch die Maxima innerhalb dieser Intervalle berücksichtigt und die Hüllkurve schmiegt sich stärker an die Signalkurve an.

Tipp

Um auch eine Hüllkurve auf der Unterseite der Signalkurve zu erhalten, können Sie die gleiche Funktion in der Form

```
-Envelope (-'Expression','Interval')
```

verwenden. Damit werden dann die Minima miteinander verbunden.

11.3 False / True

```
False() bzw. True()
```

Beschreibung
Diese Operanden liefern den konstanten Wert 0 bzw. 1.

In booleschen Operationen (AND, OR usw.) wird der Wert als logisch 0 (false) bzw. logisch 1 (true) interpretiert. In arithmetischen Operationen und in Zusammenhang mit Analogwerten wird der Wert als 0,0 bzw. 1,0 interpretiert ("feste Null" bzw. "feste Eins").

11.4 GetBit / GetBitMask

Getbit

```
Getbit('Expression','Bitnumber')
```

Beschreibung

Diese Funktion liefert als Ergebnis den booleschen Wert des Bits 'Bitnumber' von 'Expression' nach Rundung auf den nächsten Integerwert. Die Rundungsgrenze liegt jeweils bei 0,5-Schritten. (2,48 --> 2; 2,50 -->3). Gültige Bitnummernreihenfolge: 0 (LSB) bis 15 (MSB).

Hinweis

Integerwerte mit 64 Bit können nicht mit dieser Funktion ausgewertet werden, da sie von *ibaPDA* nicht unterstützt werden und somit nicht in einer Messdatei enthalten sein können.

Beispiel

In der folgenden Tabelle ist als Beispiel das niederwertigste Byte eines Integerwertes dargestellt mit den Bits 0...7. Um die Werte 0...8 abzubilden werden die einzelnen Bits wie mit "X" markiert gesetzt. (X = TRUE)

Bitnr.	7	6	5	4	3	2	1	0
0								
1								X
2							X	
3							X	X
4						X		
5						X		X
6						X	X	
7						X	X	X
8					X			

Tipp

Wenn mehrere 8-, 16- oder 32-Bit Integerwerte in Einzelbits zu zerlegen sind, kann man sich die Arbeit erheblich erleichtern, indem man im Signalbaum einen rechten Mausklick auf das gewünschte Signal macht und im Kontextmenü „Bits anzeigen" wählt. Es werden sofort alle Bits als einzelne Digitalsignale angezeigt, ohne die *Getbit*-Funktion zu programmieren. Intern wird der gleiche Mechanismus wie bei *Getbit* angewendet.

GetBitMask

```
GetBitMask('Expression','Bitnumber')
```

Beschreibung

Diese Funktion interpretiert 'Expression' als Bitmaske eines Floatwertes und liefert als Ergebnis den Wert des Bits 'Bitnumber'. Gültiger Bereich: 0 (LSB) bis 31 (MSB)

Diese Funktion wurde speziell für die Arbeit mit Daten von SimadynD in einem speziellen Anwendungsfall entwickelt, wo bis zu 32 digitale Werte gepackt als Float-Variable aufzeichnet werden. Die Funktion *GetbitMask* wertet lediglich die Valenz des spezifizierten Bits 'Bitnumber' aus, ungeachtet dessen, ob es Teil der Mantisse oder des Exponenten ist. Im Gegensatz zur Funktion *GetBit* wird keine Rundung zum Integer vorgenommen.

Tipp

Wenn ein oder mehrere 32-Bit-Floating-Werte in Einzelbits zu zerlegen sind, kann man sich die Arbeit erheblich erleichtern, indem man im Signalbaum einen rechten Mausklick auf das gewünschte Signal macht und im Kontextmenü „Bits anzeigen" wählt. Es werden sofort alle Bits als einzelne Digitalsignale angezeigt. Intern wird der gleiche Mechanismus wie bei *GetBitMask* angewendet.

11.5 HighPrecision

```
HighPrecision('Expression')
```

Beschreibung

Mit dieser Funktion wird 'Expression' als Größe mit doppelter Genauigkeit gekennzeichnet. Berechnungen, die dann mit 'Expression' durchgeführt werden, erfolgen mit doppelter Genauigkeit, auch wenn der ursprüngliche Ausdruck nur einfache Genauigkeit hat.

Doppelte Genauigkeit hat zwar den Vorteil, dass Berechnungen genauer ausgeführt werden können, allerdings auch den Nachteil, dass doppelt so viel Speicherplatz belegt wird. *ibaAnalyzer* entscheidet daher automatisch auf Basis der Eingangsargumente für eine Berechnung, welche Genauigkeit verwendet wird.

11.6 InfoField / ChannelInfoField / ModuleInfoField

Diese Funktionen erlauben es, Informationen aus einem Info-Feld einer Messdatei, eines Kanals oder eines Moduls auszulesen.

Hinweis

Die Funktionen *InfoField, ChannelInfoField* und *ModuleInfoField* erwarten einen numerischen Wert. Falls Text ausgelesen werden soll, müssen die Funktionen *InfoFieldText*, *ChannelInfoFieldText* und *ModuleInfoFieldText* verwendet werden, siehe ↗ *InfofieldText / ChannelInfoFieldText / ModuleInfoFieldText*, Seite 74.

Argumente

'Index'	Index der Datei bzw. des Kanals oder Moduls
'InfoField'	Das Infofeld, welches ausgelesen werden soll; Muss in Anführungszeichen gesetzt werden!
'Start'	Erstes Zeichen des Feldinhalts, das ausgelesen werden soll (optional) Wird kein Wert angegeben, dann wird der komplette Inhalt ausgelesen
'End'	Letztes Zeichen des Feldinhalts, das ausgelesen werden soll (optional) Wird kein Wert angegeben, wird von Anfang bis zum letzten Zeichen ausgelesen

Hinweis

Bei der Funktion *ModuleInfoField* müssen zwei Indizes angegeben werden. Der Index der Messdatei als erstes Argument und der Index des Moduls als zweites. Alle anderen Argumente bleiben gleich.

InfoField

```
Infofield('FileIndex' ,"'InfoField'",'Start','End')
```

Tipp

Wenn Sie im Signalbaum einen Doppelklick auf das gewünschte Infofeld machen, dann fügt *ibaAnalyzer* die entsprechende Funktion automatisch als neues Signal in die Signaltabelle ein. Anschließend brauchen Sie bei Bedarf nur noch den Signalnamen und Anfang/Ende anpassen. Diese Methode funktioniert auch im Eingabefeld des Ausdruckseditors. Die Funktion wird dann an der Cursorposition eingefügt.

Wenn Sie den Inhalt eines Infofeldes als Textkanal auslesen wollen, verwenden Sie die Funktion *ChannelInfofieldText*.

ChannelInfoField

```
ChannelInfofield('ChannelIndex',"'InfoField'",'Start','End')
```

ModuleInfoField

```
ModuleInfoField('FileIndex','ModuleIndex',"'InfoField'",'Start','Ende')
```

Hinweis

Bei dieser Funktion müssen zwei Indizes angegeben werden. Der Index der Messdatei als erstes Argument und der Index des Moduls als zweites.

11.7 LimitAlarm

```
LimitAlarm('Expression','Limit','DeadBand','Time')
```

Argumente

'Expression'	Messwert
'Limit'	Grenzwert, ab dem die Funktion TRUE zurückgibt
'DeadBand'	Angabe einer Totzone unterhalb des Grenzwertes, innerhalb der die Funktion nicht auf FALSE zurückgesetzt wird
'Time'	Angabe der Zeit, die der Messwert oberhalb des Grenzwertes liegen muss, bis die Funktion auf TRUE gesetzt wird

Beschreibung

Diese Funktion überwacht den Messwert ('Expression') und setzt das Ergebnis auf TRUE, wenn der Messwert länger als die angegebene Zeit ('Time') oberhalb des Grenzwertes ('Limit') liegt. Das Ergebnis der Funktion wird wieder FALSE, wenn der Messwert den Grenzwert um den unter Totzone ('DeadBand') angegebenen Wert unterschreitet.

Tipp

Die Funktion *LimitAlarm* kann auch für einen unteren Grenzwert verwendet werden. Dazu müssen lediglich der Messwert und der Grenzwert gespiegelt werden, d.h. multipliziert mit (-1).

z. B.: LimitAlarm([0:1] *(-1), 9 *(-1), 0.5, 0.4)

11.8 ManY

```
ManY('Xbase','y0','y1',....)
```

Argumente

'XBase'	Abtastrate des Ausgangssignals
'y0', 'y1',...	Y-Werte des Ausgangssignals

Beschreibung

Mit dieser Funktion kann manuell ein Signalverlauf mit den "Messwerten" 'y0'....'y99' erzeugt werden, die jeweils einen Zeit- oder Wegabstand *'Xbase'* voneinander haben. Die Angabe von *'Xbase'* versteht sich in [s] für zeitbezogene Werte und in [m] für längenbezogene Werte. Die Menge der Punkte ist auf 100 begrenzt.

So können z. B. Referenzkurven eingegeben werden, mit denen dann die real gemessenen Signale verglichen werden. Oder es werden einer Analyse Daten hinzugefügt, die nicht als Messwert vorliegen. Außerdem können mit dieser Funktion auch Textkanäle manuell erzeugt und mit unterschiedlichen Werten belegt werden.

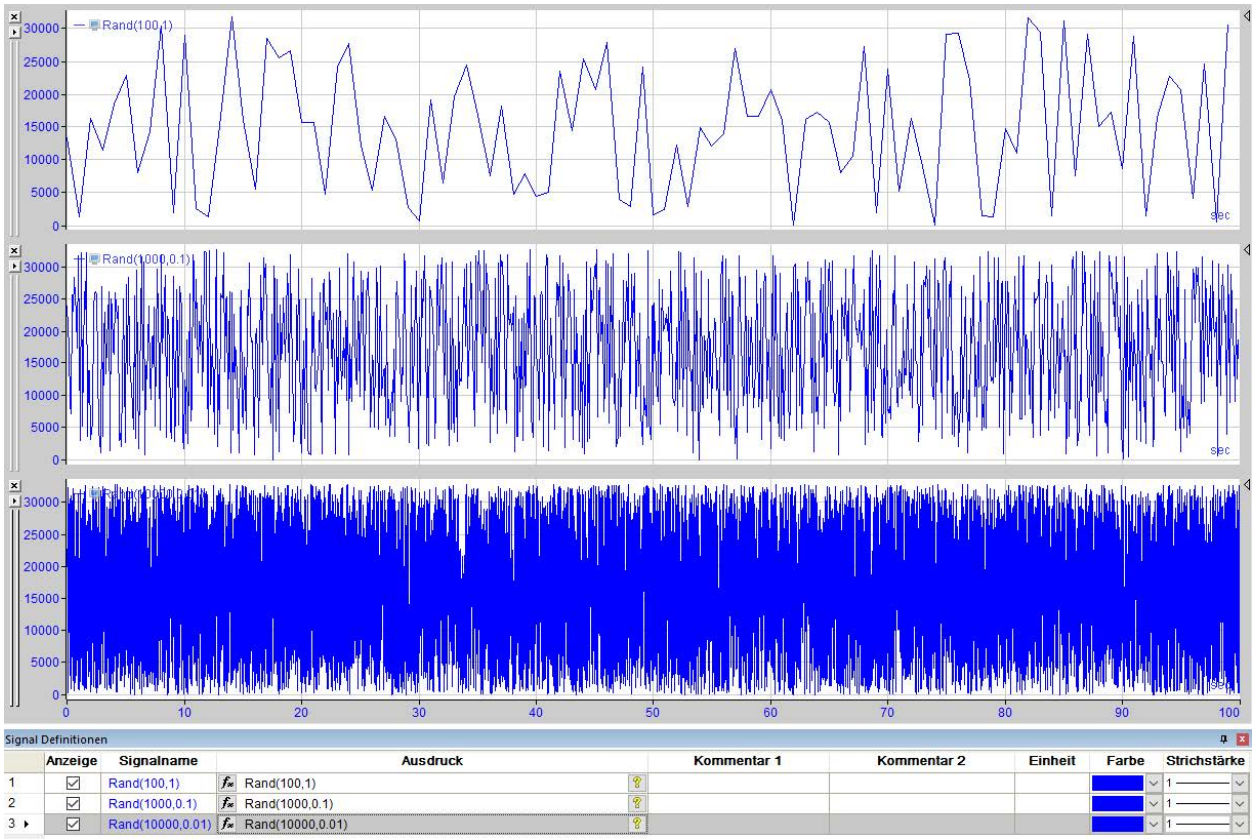
Tipp

Wenn Sie die Parameter y0 bis maximal y99 in Anführungszeichen setzen, werden eingegebene Zeichen nicht als Zahlenwerte sondern als ASCII-Zeichen übernommen.

11.9 Rand

```
Rand('Count','Xbase')
```

Beschreibung
Diese Funktion erzeugt ein Signal bestehend aus Zufallszahlen im Bereich von 0 bis 32767 für die 'Count' von Punkten in der 'Xbase' [s] (zeitbasiert) oder [m] (längenbasiert). Im nächsten Bild sind drei Signale dargestellt, die 100 s lang dauern, jedoch aus einer unterschiedlichen Anzahl Punkten bestehen. Die Zeitbasis 'Xbase' beträgt 1 s, 100 ms und 10 ms.



Signal Definitionen

	Anzeige	Signalname	Ausdruck	Kommentar 1	Kommentar 2	Einheit	Farbe	Strichstärke
1	☑	Rand(100,1)	Rand(100,1)					1
2	☑	Rand(1000,0.1)	Rand(1000,0.1)					1
3	☑	Rand(10000,0.01)	Rand(10000,0.01)					1

11.10 Sign

```
Sign('Expression')
```

Beschreibung
Diese Funktion liefert als Ergebnis das Vorzeichen von 'Expression':

'Expression' > 0 --> +1

'Expression' = 0 --> 0

'Expression' < 0 --> -1

11.11 Technostring

```
Technostring('Index','Begin','End')
```

Argumente

'Index'	Messdatei Index
'Begin'	Anfang des auszulesenden Bereichs
'End'	Ende des auszulesenden Bereichs

Beschreibung
Diese Funktion extrahiert den String aus dem Messdatei-Index 'Index' zwischen 'Begin' und 'End'. Standardstartindex ist 0. Somit können Informationen aus dem Technostring als Signale interpretiert werden (nur numerische Zeichen).

Ausgewertet wird die Technostring-Information, die im Info-Zweig im Signalbaumfenster zu sehen ist. Voraussetzung ist natürlich, dass die Technostring-Informationen von *ibaPDA* in der Messdatei mit abgespeichert wurden.

'Begin' und 'End' entsprechen der Position der Zeichen im Technostring, die den gewünschten Bereich begrenzen, der als Signal ausgewertet werden soll. Es können nur numerische Zeichen ausgewertet werden. Führende Nullen werden verworfen.

Die Angabe des 'Index' ist nur dann erforderlich, wenn mehrere Messdateien gleichzeitig geöffnet sind. Die Datei an oberster Stelle im Signalbaumfenster hat den Index 0. Alle weiteren Dateien dann von oben nach unten 1, 2, usw. Wenn nur eine Datei geöffnet ist, dann muss der Index stets = 0 sein.

11.12 WindowAlarm

```
WindowAlarm('Expression','Limit1','DeadBand1','Limit2','DeadBand2','Time')
```

Argumente

'Expression'	Messwert
'Limit1'	Oberer Grenzwert, ab dem die Funktion TRUE zurückgibt
'DeadBand1'	Angabe der Totzone unterhalb des oberen Grenzwertes ('Limit1'), innerhalb der die Funktion nicht auf FALSE zurückgesetzt wird
'Limit2'	Unterer Grenzwert, ab dem die Funktion TRUE zurückgibt
'DeadBand2'	Angabe der Totzone oberhalb des unteren Grenzwertes ('Limit2'), innerhalb der die Funktion nicht auf FALSE zurückgesetzt wird
'Time'	Angabe der Zeit, die der Messwert größer als der obere Grenzwert oder kleiner als der untere Grenzwert sein muss, bis die Funktion auf TRUE gesetzt wird

Beschreibung
Diese Funktion überwacht den Messwert ('Expression') und setzt das Ergebnis auf TRUE, wenn der Messwert länger als die angegebene Zeit ('Time') außerhalb des Bereichs zwischen oberem Grenzwert ('Limit1') und unterem Grenzwert ('Limit2') liegt. Das Ergebnis der Funktion wird wieder FALSE, wenn der Messwert den oberen Grenzwert um den unter Totzone1 ('DeadBand1') angegebenen Wert unter-, bzw. den unteren Grenzwert um den unter Totzone2 ('DeadBand2') angegebenen Wert überschreitet.

11.13 YatX / SetYatX

YatX

```
YatX('Expression','X','Continuous'=0)
```

Argumente

'Expression'	Eingangssignal
'X'	Position, an welcher der Wert ausgelesen werden soll
'Continuous'	Optionaler Parameter, um variable Werte von ‘X‘ zuzulassen

Beschreibung

Diese Funktion liefert als Ergebnis den Y-Wert von 'Expression' bei Position 'X' auf der X-Achse zurück. Die Funktion kann sowohl auf zeit- als auch längenbezogene Signale angewendet werden.

Im Standardmodus, d. h. wenn der Parameter 'Continuous' nicht angegeben wird oder FALSE bzw. 0 ist, erwartet die Funktion einen konstanten X-Wert und liefert einen konstanten Y-Wert als Ergebnis.

Der Parameter 'X' kann aber auch variabel sein, d. h. er kann selbst eine Funktion sein. In diesem Fall muss der kontinuierliche Modus aktiviert werden, indem der Parameter 'Continuous' auf TRUE bzw. 1 gesetzt wird. Die Funktion ermittelt dann zu jedem Wert von 'X' den passenden Y-Wert.

SetYatX

```
SetYatX('Expression','Value','XPos')
```

Argumente

'Expression'	Ausdruck, der verändert werden soll
'Value'	Der Wert, der an der Stelle 'XPos' eingefügt werden soll
'XPos'	Die X-Position, an der der Wert 'Value' eingefügt werden soll

Beschreibung

Die Funktion *SetYatX* erlaubt es, eine Kopie eines Signals zu erstellen, in der ein Wert geändert wurde. Sie liefert als Ergebnis eine Kopie des Signals 'Expression', bei dem an der Stelle 'XPos' der Wert 'Value' eingefügt wurde.

Je nachdem, ob ein äquidistant gesampeltes Signal vorliegt oder nicht, verhält sich die Funktion anders. Bei äquidistanten Signalen werden folgende Fälle unterschieden:

- Falls 'XPos' kleiner als der Offset des Signals ist, wird das Signal unverändert zurückgegeben.
- Falls 'XPos' der Größe des Signals (vgl. XSize) plus der Abtastgröße entspricht, wird das Signal um ein Sample mit dem Wert 'Value' verlängert.
- In allen anderen Fällen wird der neue Wert an der Stelle 'XPos' oder an der nächstkleineren Sample-Position eingefügt.

Für nicht äquidistant abgetastete Signale ersetzt die Funktion den Wert an der Stelle 'XPos', falls vorhanden, oder fügt ein neues Sample ein.

Hinweis

Die Funktion kann ebenfalls genutzt werden, um Text einzufügen.

11.14 PulseFreq

```
PulseFreq('Expression','Omega'=0,'EdgeType'=2,'MinFreq'=0.05)
```

Argumente

'Expression'	Pulszählersignal	
'Omega'	Filterfrequenz	
'EdgeType'	Flankenart, die gezählt werden soll	
	'EdgeType' = -1	nur fallende Flanken
	'EdgeType' = 0	steigende und fallende Flanken
	'EdgeType' = 1	nur steigende Flanken
	'EdgeType' = 2	'Expression' ist ein Impulszähler
'MinFreq'	Kleinste Frequenz, die dargestellt wird	

Beschreibung
Diese Funktion berechnet die Frequenz von Impulsen oder Impulszählern 'Expression'. Ergebniseinheit ist Pulse/Sek. bzw. Hz.

Ein Tiefpassfilter mit einer Grenzwinkelgeschwindigkeit 'Omega' wird auf das Ergebnis angewendet. Wenn 'Omega' 0 ist, dann ist der Tiefpassfilter deaktiviert. 'EdgeType' bestimmt, welche Flanken der Pulse gezählt werden sollen. Als berechnete Frequenz wird Null zurückgegeben, wenn während 1000 Abtastungen kein Puls auftritt.

Diese Funktion ist speziell für die Auswertung des WAGO-Inkrementalgebers 750-631 erstellt worden. Die Funktion kann zur Geschwindigkeitsberechnung aus dem Pulszählerstand benutzt werden. Der Pulszählerstand wird unter Beachtung eines möglichen Überlaufs differenziert.
Da das Ergebnis dieser Differentiation mit Störfrequenzen bzw. einem Rauschen versehen sein kann, wird anschließend ein Tiefpassfilter darauf angewendet. Die einzustellende Filterfrequenz sollte etwas oberhalb der maximalen Pulsfrequenz liegen.

12 Filter-Funktionen

12.1 LP

```
Lp('Expression','Omega')
```

Argumente

'Expression'	Messwert
'Omega'	Eckfrequenz für den Tiefpass-Filter

Beschreibung
Diese Funktion ist ein digitaler Tiefpassfilter 1. Ordnung mit der Eckfrequenz 'Omega'. Angewendet auf ein Signal 'Expression', liefert sie als Ergebnis ein Signal, das nur noch die Wechselanteile mit Frequenzen kleiner als 'Omega' enthält.

Hinweis

Digitale Filter, die mit dem Filtereditor erstellt wurden, können im System abgespeichert werden und stehen dann auch im Ausdrucksеditor als Filterfunktionen zur Verfügung.

12.2 PreWhiten

```
PreWhiten('Expression','Order')
```

Argumente

'Expression'	Messwert, der gefiltert werden soll
'Order'	Ordnung des FIR Filters

Beschreibung
Diese Funktion wendet einen FIR Filter mit Koeffizienten an, die mit Hilfe der Yule-Walker Gleichung bestimmt werden. Es handelt sich um einen Hochpass-Filter, der nur weißes Rauschen und die Impulskomponenten des Signals übriglässt.

13 Technologische Funktionen

13.1 ChebyCoef

```
ChebyCoef('Vector','begin_seg','end_seg','Order','CoverFactor'=1)
```

Argumente

'Vector'	Messwerte, die approximiert werden sollen
'begin_seg'	Erstes zu verwendendes Vektor-Segment
'end_seg'	Letztes zu verwendendes Vektor-Segment
'Order'	Ordnung des Chebyshev-Polynoms
'CoverFactor'	Optionales Argument um den Deckungsfaktor festzulegen

Beschreibung
Die Funktion *ChebyCoef* berechnet den Koeffizienten des Chebyshev-Polynoms der Ordnung ‘Order‘ über das Querprofil eines Vektors ‘Vector‘. Dabei werden nur die Einträge des Vektors zwischen den Segmenten ‘begin_seg‘ und ‘end_seg‘ berücksichtigt. Ein optionaler Deckungsfaktor ‘CoverFactor‘ bestimmt das Verhalten an den Rändern.

Beispiel
Das Chebyshev-Polynom, benannt nach dem gleichnamigen russischen Mathematiker, hat sich als geeignet erwiesen, das Profil eines Walzspalts mathematisch zu beschreiben. Relevant für die Walzspaltapproximation sind die Ordnungen 0 bis 6 des Polynoms, für die die Funktion die entsprechenden Koeffizienten liefert.

In der Praxis können die Koeffizienten aus den Messwerten einer Planheitsmessrolle abgeleitet werden. Dabei werden die Planheitsmesswerte der einzelnen Zonen in einem mehrdimensionalen Signal ‘Vector' zusammengefasst. Jedes Array-Feld entspricht einem Segment im Sinne eines Querprofils.

13.2 CubicSpline

```
CubicSpline('Expression','X','Y')
```

Argumente

'Expression'	Hilfssignal, um die Abtastrate des Ergebnisses festzulegen
'X'	X-Koordinaten (Stützstellen), die den Spline definieren
'Y'	Y-Koordinaten, die den Spline definieren

Beschreibung
Diese Funktion liefert als Ergebnis einen kubischen Spline, der an den Stützstellen ‘X‘ mit zugehörigen Werten ‘Y‘ ausgerichtet ist. Die Abtastrate und die Auswertepunkte des Ergebnisses werden durch ‘Expression‘ festgelegt.

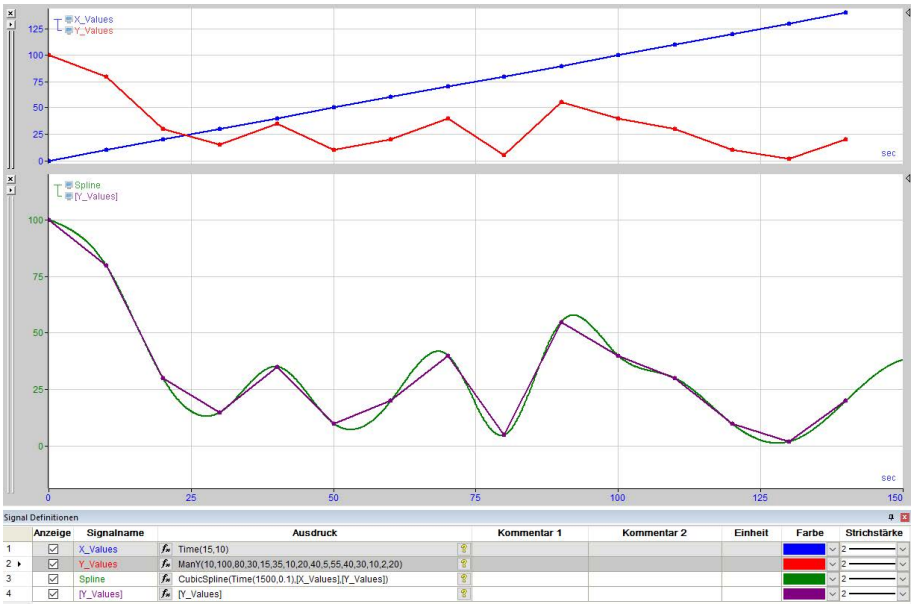
Die X-Koordinaten müssen nicht eindeutig und sortiert sein. Sollten mehrere Wertepaare mit der gleichen X-Koordinate vorhanden sein, wird für die Berechnung des Splines nur das letzte

Wertepaar verwendet. Die verbleibenden Wertepaare werden automatisch nach X-Koordinaten sortiert.

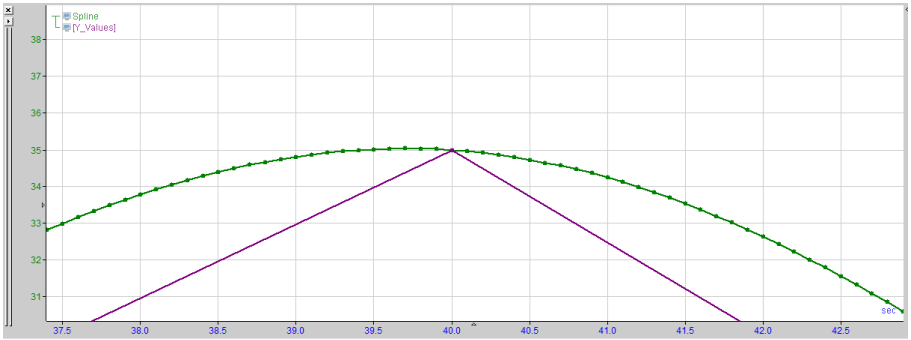
Beispiel

Für eine Reihe von Punkten liefert die Funktion als Ergebnis ein geglättetes Signal entlang des berechneten Splines. Die Funktion kann dazu verwendet werden, für ein Signal mit wenigen Samples eine Ausgleichskurve zu interpolieren:

Eine Kurve, weist nur 17 Samples über einen Zeitraum von 5000 s auf (Y-Werte, grüne Kurve). Die dazu passenden X-Koordinaten – ebenfalls nur 17 Werte – sind als blaue Kurve eingezeichnet. Die Ausgleichskurve soll als geglättetes Signal eine deutlich höhere Auflösung erhalten (mehr Samples). Daher wird der Funktion *CubicSpline* als Parameter 'Ausdruck' eine lineare Funktion mit 5000 Samples im Abstand von 1 s übergeben.



In der stark gezoomten Ansicht sind die berechneten Samples der Ausgleichskurve zu erkennen (grün). Die ursprünglichen X/Y-Koordinaten bilden die Knoten des Splines (violett).



13.3 LSQPolyCoef

```
LSQPolyCoef('X','Y','degree','PolynomialType' = 0)
```

Argumente

'X'	X-Koordinaten (Stützstellen), die das Interpolationspolynom definieren
'Y'	Y-Koordinaten, die das Interpolationspolynom definieren
'degree'	Polynomgrad (0 = Mittelwert, 1 = linear, 2 = quadratisch, 3 = kubisch, etc.)
'Polynomi-alType'	Definiert die Basispolynome, die verwendet werden (0 = Lagrange (default), 1 = Chebyshev I, 2 = Chebyshev II, 3 = Legendre)

Beschreibung

Diese Funktion berechnet die Koeffizienten eines Interpolationspolynoms vom Grad 'degree' für Wertepaare 'X' und 'Y' nach der Methode der kleinsten Quadrate.

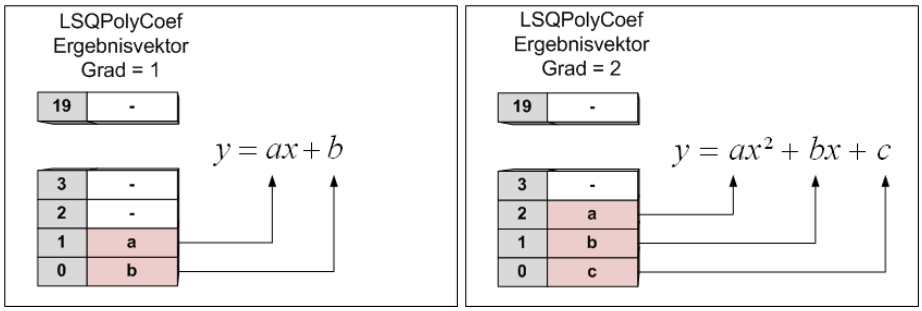
Das Ergebnis der Funktion ist ein Vektor (mehrdimensionales Signal, Array), in dem die Koeffizienten enthalten sind. Das Array-Feld mit dem Index 0 enthält den konstanten Anteil oder Offset des Polynoms. Die Koeffizienten werden mit aufsteigendem Grad entsprechend in Array-Felder mit aufsteigendem Index geschrieben.

Die Auswertung des Polynoms kann mit der Funktion '*Polynomial*' erfolgen.

Der optionale Parameter 'PolynomialType' kann verwendet werden, um verschiedene Basispolynome zu verwenden.

Beispiel

Eine quadratische Approximation liefert ein Polynom der Form $y = ax^2 + bx + c$, d. h. als Ergebnis erhält man einen Vektor mit insgesamt 3 Koeffizienten.



Die Funktion basiert prinzipiell auf einer X-Y-Beziehung, d. h. die Operanden X und Y können auch zwei unterschiedliche Messsignale sein. Soll nur eine Regressionskurve für ein Signal über die Zeit berechnet werden, dann müssen die Zeitwerte auch in die Form eines Signals gebracht werden, z. B. mithilfe der Funktion *XValues* ([Signal]). Dieses Zeitsignal, dessen Y-Werte identisch mit der Zeit entlang der X-Achse sind, kann dann als Operand ‚X' in der Funktion *LSQPoly-Coef* verwendet werden.

13.4 Polynomial

```
Polynomial('Coefs','X','PolynomialType' = 0)
```

Argumente

'Coefs'	Vektor mit Koeffizienten, z.B. als Ergebnis von *LSQPolyCoef*
'X'	X-Koordinaten (Stützstellen), an denen das Polynom ausgewertet werden soll
'Polynomi-alType'	Definiert die Basispolynome, die verwendet werden (0 = Lagrange (default), 1 = Chebyshev I, 2 = Chebyshev II, 3 = Legendre)

Beschreibung
Diese Funktion berechnet für jedes Sample von 'X' den Polynomwert auf Basis eines Koeffizientenvektors ‚*Coefs*'. Sie wird für die Darstellung von Regressionsgeraden oder Ausgleichskurven benötigt, deren Koeffizienten zuvor mit der Funktion *LSQPolyCoef* berechnet wurden.

Der optionale Parameter 'PolynomialType' kann verwendet werden, um verschiedene Basispolynome zu verwenden.

13.5 LSQExponentialCoef

```
LSQExponentialCoef('X','Y')
```

Argumente

'X'	X-Koordinaten (Abtastpunkte), die die interpolierende Exponentialfunktion definieren
'Y'	Y-Werte, die die interpolierende Exponentialfunktion definieren

Beschreibung
Diese Funktion berechnet die Koeffizienten eines interpolierenden Polynoms a*exp(b*x) für die durch die X-Koordinaten 'X' gegebene Funktion und die entsprechenden Funktionswerte 'Y' nach der Methode der kleinsten Quadrate. Das Ergebnis der Funktion ist ein Vektor, der die Koeffizienten enthält.

13.6 Exponential

```
Exponential('Coefs','X')
```

Argumente

'Coefs'	Vektor mit Koeffizienten, z.B. das Ergebnis von LSQExponentialCoef
'X'	X-Koordinaten (Abtastpunkte), an denen die Exponentialfunktion ausgewertet wird

Beschreibung
Diese Funktion berechnet den Wert einer Exponentialfunktion a*exp(b*X) für jedes Sample von 'X' mit den Koeffizienten a und b, die als Vektor 'Coefs' angegeben sind. Die Funktion kann zur Visualisierung der Ergebnisse von *LSQExponentialCoef* verwendet werden.

14 Spektralanalyse (FFT-Operationen)

ibaAnalyzer verfügt über die Möglichkeit der Spektralanalyse in Form der Fast Fourier Transformation (FFT). Mit den zur Verfügung stehenden FFT-Operationen kann ein zeit- oder längenbasiertes Signal nicht nur im FFT-Modus angezeigt werden, sondern als berechneter Ausdruck zur Verfügung gestellt und für weiterführende Analysen verwendet werden.

Für die meisten der hier beschriebenen Funktionen gibt es die Möglichkeit, entweder einen Amplituden- oder einen Leistungstrend anzuzeigen. Dies wird durch die Endungen ‘Ampl‘ bzw. ‘Power‘ in den Funktionsnamen gekennzeichnet.

14.1 FftInTimeAmpl / FftInTimePower

```
z.B. FftInTimeAmpl('Expression','Samples','#Freq','min_Freq'=0,
'max_Freq','Window'=0,'Overlap'=0,'DC-Suppression'=0)
```

Argumente

'Expression'	Ausdruck für den die FFT berechnet werden soll
'Time'	Festlegung der verwendeten Zeit- bzw. Längenintervalle. Hier wird auf ein Intervall gerundet, welches 2^N Samples enthält
'#Freq'	Anzahl der angezeigten Frequenzen
'min_Freq'	Minimale Frequenz
'max_Freq'	Maximale Frequenz
'Window'	Fenstertyp: 0 = Rechteck 1 = Bartlett 2 = Blackman 3 = Hamming 4 = Hanning 5 = Blackman-Harris 6 = Flat top
‘Overlap'	Überlappungsfaktor
'DC-Suppression'	Gleichanteilunterdrückung

Beschreibung
Diese Funktionen berechnen Amplitude bzw. Leistung der Fourier-Transformierten von 'Expression' für Abschnitte mit jeweils 2^N Messwerten. Dabei wird N bestimmt, indem das Produkt 'Time' x Abtastfrequenz auf eine Zweierpotenz gerundet wird.

Das Ergebnis ist ein Vektor, der '#Freq' gleichmäßig aufgeteilte Frequenzen zwischen 'min_Freq' und 'max_Freq' pro Abschnitt enthält. Der Fenstertyp, der für die Berechnung verwendet wird, kann über den Parameter ‘Window‘ gesteuert werden.

Der Überlappungsfaktor bestimmt die Überlappung der Zeitsegmente und kann zwischen 0 (keine Überlappung) und 1 (vollständige Überlappung) liegen. Optional ist es möglich die Gleichanteilunterdrückung mit dem Parameter ‘DC-Suppression‘ zu aktivieren.

Beispiel
Mithilfe der *FftInTime*-Funktion lassen sich Frequenzschwankungen über die Zeit darstellen. Dazu kann der resultierende Vektor in einer 2D-Ansicht angezeigt werden.

14.2 FftOrderAnalysisAmpl / FftOrderAnalysisPower

```
z.B. FftOrderAnalysisAmpl('Expression','Samples','Freq','min_Order'=0,'
max_Order','Order_division','Window'=0,'Overlap'=0,'DC-Suppression'=0)
```

Argumente

'Expression'	Ausdruck, für den die Ordnungsanalyse durchgeführt werden soll
'Time'	Festlegung der verwendeten Zeit- bzw. Längenintervalle
'Freq'	Grundfrequenz für die Ordnungsanalyse (Drehfrequenz)
'min_Order'	Minimal angezeigte Ordnung
'max_Order'	Maximal angezeigte Ordnung
'Order_division'	Rasterbreite zwischen den ganzzahligen Ordnungen
'Window'	Fenstertyp 0 = Rechteck 1 = Bartlett 2 = Blackman 3 = Hamming 4 = Hanning 5 = Blackman-Harris 6 = Flat top
‘Overlap'	Überlappungsfaktor
'DC-Suppression'	Gleichanteilunterdrückung

Beschreibung
Diese Funktion berechnet die Ordnungen (d.h. Vielfache einer Grundfrequenz ‘Freq‘) für ein Signal und liefert als Ergebnis einen Vektor mit den Ordnungen zwischen ‘min_Order‘ und ‘max_Order‘. Die Anzahl der Datenpunkte pro Ordnung wird durch den Paramter ‘Order_division‘ festgelegt.

Der Fenstertyp, der für die Berechnung verwendet wird, kann über den Parameter ‘Window‘ gesteuert werden. Der Überlappungsfaktor bestimmt die Überlappung der Zeitsegmente und kann zwischen 0 (keine Überlappung) und 1 (vollständige Überlappung) liegen. Optional ist es möglich die Gleichanteilunterdrückung mit dem Parameter ‘DC-Suppression‘ zu aktivieren.

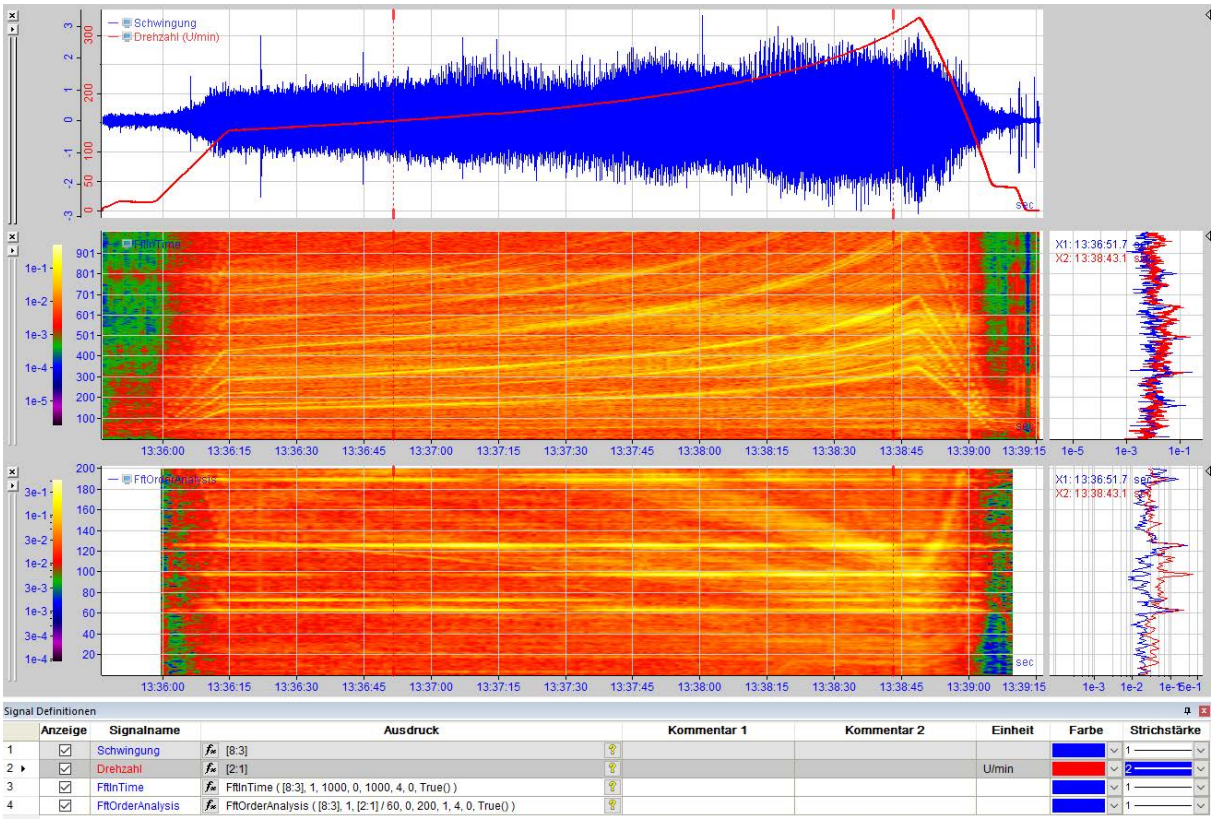
Im Gegensatz zur *FftInTime*-Funktion werden auf der Y-Achse nicht mehr die Zeit/Frequenz, sondern die Drehzahlfrequenz und deren Vielfache dargestellt, d.h. die Ordnungen. Die Frequenzachse wird dazu entsprechend der momentanen Drehzahl verzerrt, so dass die Ordnungen jetzt nicht mehr als Kurve, sondern als gerade Linien dargestellt werden. Je nach Funktion wird entweder ein Amplitudentrend (*FftOrderAnalysisAmpl*) oder ein Leistungstrend berechnet (*FftOrderAnalysisPower*).

Hinweis

Die Funktion liefert keine Resultate, wenn die Anzahl der Signalpunkte pro Umdrehung mehr als doppelt so hoch wie der gewählte Parameter ‘Time‘ ist.

Beispiel

Mit der Funktion *FftOrderAnalysis* können Sie die Berechnung der Ordnungsanalyse durchführen. Frequenzen, die der Motordrehzahl oder deren Vielfachen entsprechen, werden als Ordnungen bezeichnet. Die erste Ordnung entspricht der Frequenz der Motordrehzahl, die zweite Ordnung entspricht der Frequenz der ersten Ordnung multipliziert mit dem Faktor 2, usw. Bei der Ordnungsanalyse wird der Pegel oder der Pegelverlauf dieser Ordnungen berechnet.



Zwischen den einzelnen Signalpunkten erfolgt zur Berechnung der *FftOrderAnalysis* eine Interpolation.

14.3 FftPeaksInTimeAmpl / FftPeaksInTimePower

```
z.B. FftPeaksInTimeAmpl('Expression','Samples','#Peaks','min_Freq'=0,
'max_Freq','Window'=0,'Overlap'=0,'DC-Suppression'=0,'Zero-Padding'=0)
```

Argumente

'Expression'	Ausdruck, für den die Frequenzspitzen ausgewertet werden sollen
'Time'	Festlegung der verwendeten Zeit- bzw. Längenintervalle. Hier wird auf ein Intervall gerundet welches 2^N Samples enthält
'#Peaks'	Anzahl der angezeigten Spitzen
'min_Freq'	Optionaler Parameter für die minimal berücksichtigte Frequenz, d.h. Spitzen bei niedrigerer Frequenz werden nicht angezeigt
'max_Freq'	Optionaler Parameter für die maximal berücksichtigte Frequenz, d.h. Spitzen bei höherer Frequenz werden nicht angezeigt
'Window'	Fenstertyp: 0 = Rechteck 1 = Bartlett 2 = Blackman 3 = Hamming 4 = Hanning 5 = Blackman-Harris 6 = Flat top
‘Overlap'	Überlappungsfaktor
'DC-Suppression'	Gleichanteilunterdrückung
'Zero-Padding'	Ergänzen von Nullen

Beschreibung

Diese Funktion dient der Berechnung von Frequenzspitzen über gleitende Zeitintervalle, die durch den Parameter ‘Time‘ festgelegt werden. Dabei werden die ‘#Peaks‘, die höchsten Spitzen zwischen den Frequenzen ‘min_Freq‘ und ‘max_Freq‘ berechnet. Der Zeitverlauf der Frequenz und Spitzenwertepaare wird als Vektor zurückgegeben.

Dabei sind die Einträge nach folgendem Muster sortiert:

- Index 0: Frequenz mit der höchsten Spitze
- Index 1: Amplitude/Leistungen der höchsten Spitze
- Index 2: Frequenz mit der zweithöchsten Spitze
- Index 3: Amplitude/Leistungen der zweithöchsten Spitze
- Etc.

Tip

Um die gewünschten Werte aus dem Ergebnisarray auszulesen, können Sie die Funktion 'GetRows' benutzen.

Der Fenstertyp, der für die Berechnung verwendet wird, kann über den Parameter ‘Window‘ gesteuert werden. Der Überlappungsfaktor bestimmt die Überlappung der Zeitsegmente und kann zwischen 0 (keine Überlappung) und 1 (vollständige Überlappung) liegen. Optional ist es möglich, die Gleichanteilunterdrückung mit dem Parameter ‘DC-Suppression‘ zu aktivieren. Wenn der Parameter ‘Zero_Padding‘ auf 1 oder True() gesetzt ist, dann wird vor der Berechnung der FFT das letzte Fenster mit Nullen aufgefüllt.

14.4 FftAmpl / FftPower

```
z.B. FftAmpl('Expression','Samples','Window'=0,'DC-Suppression'=0,'Zero-Pad-
ding'=0)
```

Argumente

'Expression'	Ausdruck, für den die Fourier Transformierte berechnet werden soll
'Samples'	Anzahl der zu berücksichtigenden Messwerte und dadurch implizite Festlegung des verwendeten Zeit- bzw. Längenintervalls, abhängig von der Abtastrate
'Window'	Fenstertyp: 0 = Rechteck 1 = Bartlett 2 = Blackman 3 = Hamming 4 = Hanning 5 = Blackman-Harris 6 = Flat top
'DC-Suppression'	Gleichanteilunterdrückung
'Zero-Padding'	Ergänzen von Nullen

Beschreibung
Diese Funktionen berechnen die Amplitude bzw. Leistung der Fourier-Transformierten des Signals. Der verwendete Zeitabschnitt wird durch Runden der verwendeten Messpunkte ‘Samples‘ auf eine Zweierpotenz bestimmt.

Hinweis

Der Parameter ‘Samples‘ wird aufgerundet. Es müssen mindestens 128 Messpunkte verwendet werden.

Der Fenstertyp, der für die Berechnung verwendet wird, kann über den Parameter ‘Window‘ gesteuert werden. Optional ist es möglich die Gleichanteilunterdrückung mit dem Parameter ‘DC-Suppression‘ zu aktivieren. Wenn der Parameter ‘Zero_Padding‘ auf 1 oder True() gesetzt ist, dann wird zur Berechnung der FFT das Fenster mit Nullen aufgefüllt.

14.5 FftComplex

```
FftComplex('Expression','inv','normalize'=0)
```

Argumente

'Expression'	Ausdruck, für den die Fourier-Transformation berechnet werden soll
'inv'	Optionaler Parameter um eine inverse Fourier-Transformation zu ermöglichen
'normalize'	Optionaler Parameter um eine Normierung zu wählen

Beschreibung
Diese Funktion führt eine Fourier-Transformation für ein komplexes Signal über den gesamten Ausdruck aus und liefert als Ergebnis einen Vektor mir Real- und Imaginärteil der Fourier-Transformierten. Das Eingangssignal kann sowohl ein einzelnes Signal oder ein Vektor bestehend aus Real- und Imaginärteil sein. Dabei wird ein Rechteck-Fenster für die Berechnung verwendet.

Wenn der Parameter ‘inv‘ auf True() bzw. 1 gesetzt wird, wird eine inverse Fourier-Transformation berechnet. In diesem Fall erwartet die Funktion ein Eingangssignal, das entweder frequenzbasiert oder 1/Länge-basiert ist. Das Ergebnis der Operation ist dann entsprechend ein zeit- oder längenbasiertes Signal.

Für den Parameter ‘normalize‘ sind folgende Werte zulässig:

- **0**: Es wird keine Normierung durchgeführt.
- **1**: Das Ergebnis wird durch die Zahl der Samples dividiert. Bei einer inversen Transformation wird das Ergebnis nicht verändert.
- **2**: Das Ergebnis wird durch die Quadratwurzel der Anzahl Samples dividiert. Dies gilt sowohl für eine normale als auch eine inverse Transformation
- **Andere Werte**: Funktion wie mit Wert 1.

Die Anzahl der Frequenz-Samples wird durch die Anzahl der Samples des Eingangssignals bestimmt. Wenn N gerade ist, werden N/2+1 Frequenzpunkte berechnet, von denen der erste (Gleichanteil) und der letzte Punkt rein reell sind. Wenn N ungerade ist, dann werden (N+1)/2 Frequenzpunkte berechnet, für die der Gleichanteil rein reell ist.

14.6 FftReal / FftRealInverse

```
z.B. FftReal('Expression','normalize'=0)
```

Argumente

'Expression'	Ausdruck, für den die Fourier-Transformierte berechnet werden soll
'normalize'	Optionaler Parameter um die Normierung zu aktivieren

Beschreibung FftReal
Diese Funktion führt eine Fourier-Transformation für ein reelles Signal über den gesamten Ausdruck aus und liefert als Ergebnis einen Vektor mir Realteil und Imaginärteil der Fourier-Transformierten. Dabei wird ein Rechteck-Fenster verwendet.

Wenn der Parameter ‘normalize‘ True() bzw. 1 gesetzt ist, dann wird eine Normierung durchgeführt. Ist die Anzahl der Samples (N) des Signals ungerade, werden alle Frequenzwerte außer dem Gleichanteil durch N/2 dividiert. Ist N gerade, werden alle Frequenzwerte außer dem Gleichanteil und letztem Wert durch N/2 dividiert.

Die Anzahl der Frequenz-Samples wird durch die Anzahl der Samples des Eingangssignals bestimmt. Wenn N gerade ist, werden N/2+1 Frequenzpunkte berechnet, von denen der erste (Gleichanteil) und der letzte Punkt rein reell sind. Wenn N ungerade ist, dann werden (N+1)/2 Frequenzpunkte berechnet, für die der Gleichanteil rein reell ist.

Beschreibung FftRealInverse
Diese Funktion berechnet die inverse Fourier-Transformation, wie mit *FftReal* erstellt. Dabei ist das Ergebnis reell und das Eingangssignal muss dementsprechend ein Vektor bestehend aus Real- und Imaginärteil sein. Ansonsten funktioniert die Funktion genauso wie *FftReal*.

14.7 AWeighting / DbScale

AWeighting

```
AWeighting('Spectrum','Type')
```

Argumente

'Spectrum'	Spektrum
'Type'	Angabe des Typs

Beschreibung
Diese Funktion gewichtet ein Spektrum nach der sogenannten A-Bewertung. Dies ist ein Bewertungsfilter, welcher dem menschlichen Gehör entspricht.

Beispiel
Nach dem Anwenden dieser Funktion, kann ein Spektrum bezüglich der wahrnehmbaren Geräuschemission beurteilt werden.

DbScale

```
DbScale('Spectrum','Reference')
```

Argumente

'Spectrum'	Spektrum, welches logarithmisch skaliert werden soll
'Reference'	Optional

Beschreibung
Diese Funktion stellt eine logarithmische Skalierung in dB für ein Signalspektrum zur Verfügung. Ein sinnvolles Ergebnis kann nur erwartet werden, wenn als Eingang die Amplitude eines Spektrums gegeben ist.

Tipp

Funktionen, die eine solche Amplitude berechnen, sind: *FftAmpl, FftInTimeAmpl, FftOrderAnalysisAmpl*

14.8 IntSpectrum

```
IntSpectrum('Spectrum')
```

Beschreibung
Diese Funktion integriert ein gegebenes Spektrum. So kann beispielsweise aus der Frequenz eines Beschleunigungssensors die Vibrationsgeschwindigkeit berechnet werden.

15 Elektrische Funktionen

15.1 RMS / Eff

`RMS('Spectrum','Frequency') bzw. Eff('Spectrum','Frequency')`

Argumente

'Spectrum'	Messwert, für den der Effektivwert berechnet werden soll
'Frequency'	Grundfrequenz

Beschreibung
Diese Funktion berechnet den so genannten Root Mean Square-Wert bzw. den Effektivwert von 'Expression' mit der Grundfrequenz von 'Frequency':

$$E_{eff} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} e^2(n)}$$

e(n): Messpunkt n von Signal e ('expr')

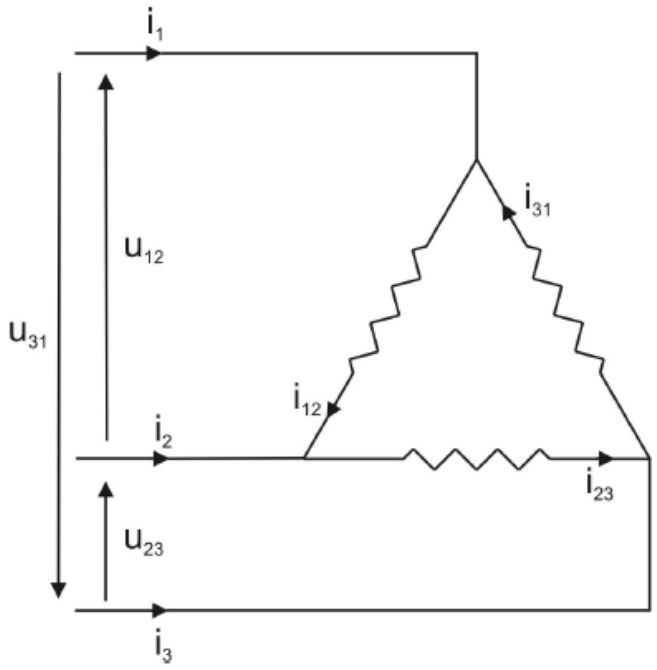
N: Anzahl der Messwerte pro Periode

Beispiel
Für einen Wechselspannungsverlauf mit einer Frequenz von 0,1 kHz, der von einer zweiten Wechselspannung mit 0,5 kHz überlagert wird, kann der Effektivwert der Spannung für beide Frequenzen ermittelt werden, indem die Funktion *Eff* mit zweitem Argument 0.1 bzw. 0.5 angewendet wird.

Hinweis

Zwischen den Funktionen *RMS* und *Eff* gibt es keinen Unterschied. Beide Funktionen werden aus Kompatibilitätsgründen von *ibaAnalyzer* unterstützt.

15.2 Dreieck-Funktionen



Die Dreieck-Funktionen benutzen die Anschlussspannungen und -ströme in einem Dreiecknetz, um die Leistungswerte zu berechnen.

Argumente

'u12', 'u13', 'u23'	Anschlussspannung (gleich den Phasenspannungen)
'i1', 'i2', 'i3'	Anschlussströme
'freq'	Grundfrequenz

Hinweis

Die Funktionen werden üblicherweise bei Dreiecknetzen eingesetzt, aber sie können auch für andere Netze verwendet werden, in denen Anschlussspannungen und -ströme gemessen werden können.

DeltaCollectiveUeff

```
DeltaCollectiveUeff('u12','u13','u23','freq')
```

Berechnet die gemeinsame Effektivspannung in einem Dreiecknetz:

$$U_{eff} = \sqrt{\frac{1}{3}(U_{12,eff}^2 + U_{23,eff}^2 + U_{31,eff}^2)}$$

$U_{xy,eff}$: der Effektivwert der Anschlussspannung Uxy

DeltaCollectiveIeff

`DeltaCollectiveIeff('i1','i2','i3','freq')`

Berechnet den gemeinsamen Effektivstrom in einem Dreiecknetz:

$$I_{eff} = \sqrt{\sum_{x=1}^{3} I_{x,eff}^2}$$

$I_{x,eff}$: der Effektivwert des Anschlussstroms ix

DeltaActiveP

`DeltaActiveP('u13','u23','i1','i2','freq')`

Berechnet die Wirkleistung in einem Dreiecknetz:

$$P = \frac{1}{N}\sum_{n=1}^{N}\left[u_{23}(n)i_2(n) + u_{13}(n)i_1(n)\right]$$

N : die Anzahl der Messwerte pro Periode

u_{xy} : die Spannung zwischen Anschluss x und y (u13 = -u31)

i_x : der Stom durch Anschluss x

DeltaApparentP

`DeltaApparentP('u12','u13','u23','i1','i2','i3','freq')`

Berechnet die Scheinleistung in einem Dreiecknetz:

$$S = U_{eff} I_{eff}$$

U_{eff} : die gemeinsame Effektivspannung

I_{eff} : der gemeinsame Effektivstrom

DeltaReactiveP

`DeltaReactiveP('u12','u13','u23','i1','i2','i3','freq')`

Berechnet die Blindleistung in einem Dreiecknetz:

$$Q = \sqrt{S^2 - P^2}$$

S : Scheinleistung

P : Wirkleistung

DeltaReactivePS

```
DeltaReactivePS ('u12', 'u13', 'u23', 'i1', 'i2', 'i3', 'freq')
```

Berechnet die vorzeichenbehaftete Blindleistung QS im Dreiecknetz:

DeltaActivePFactor

```
DeltaActivePFactor('u12','u13','u23','i1','i2','i3','freq')
```

Berechnet den Wirkleistungsfaktor in einem Dreiecknetz:

$$\cos\varphi = \frac{P}{S}$$

S : Scheinleistung

P : Wirkleistung

DeltaReactivePFactor

```
DeltaReactivePFactor('u12','u13','u23','i1','i2','i3','freq')
```

Berechnet den Blindleistungsfaktor in einem Dreiecknetz:

$$\tan\varphi = \frac{Q}{P}$$

Q : Blindleistung

P : Wirkleistung

DeltaReactivePFactorS

```
DeltaReactivePFactorS('u12','u13','u23','i1','i2','i3','freq')
```

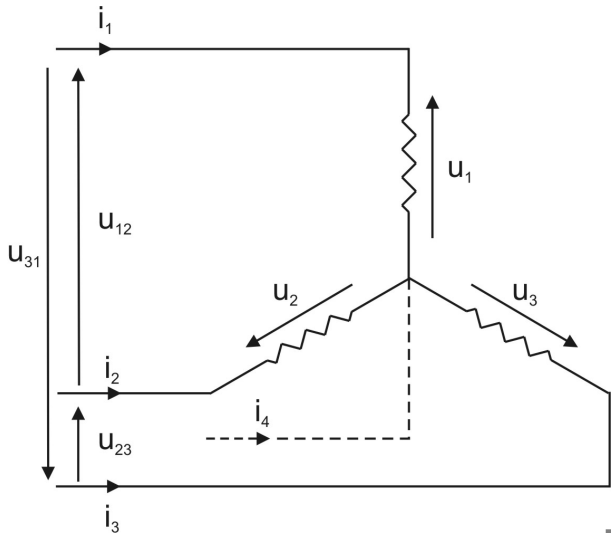
Berechnet den vorzeichenbehafteten Blindleistungsfaktor in einem Dreiecknetz:

$$\tan\varphi = \frac{Q_S}{P}$$

Q_S : vorzeichenbehaftete Blindleistung

P : Wirkleistung

15.3 Stern-Funktionen



Die Stern-Funktionen verwenden die Phasenspannungen und -ströme, um die verschiedenen Leistungswerte zu berechnen.

Argumente

'u1', 'u2', 'u3'	Phasenspannungen
'i1', 'i2', 'i3'	Phasenströme
'i4'	Sternpunktanschluss
'freq'	Grundfrequenz

Hinweis

Die Funktionen werden üblicherweise bei Sternnetzen eingesetzt, aber sie können auch für andere Netze verwendet werden, in denen Phasenspannungen und -ströme gemessen werden können.

StarCollectiveUeff

```
StarCollectiveUeff('u1','u2','u3','freq')
```

Berechnet die gemeinsame Effektivspannung in einem Sternnetz:

$$U_{eff} = \sqrt{\sum_{x=1}^{4} U_{x\_eff}^2}$$

U_{x_eff} : der Effektivwert der Phasenspannung ux

$u_4 = u_1 + u_2 + u_3$

StarCollectiveIeff

```
StarCollectiveIeff('i1','i2','i3','i4','freq')
```

Berechnet den gemeinsamen Effektivstrom in einem Sternnetz:

$$I_{eff} = \sqrt{\sum_{x=1}^{4} I_{x,eff}^2}$$

$I_{x,eff}$: der Effektivwert des Anschlussstroms ix

StarActiveP

```
StarActiveP('u1','u2','u3','i1','i2','i3','freq')
```

Berechnet die Wirkleistung in einem Sternnetz:

$$P = \sum_{x=1}^{3} \left(\frac{1}{N} \sum_{n=1}^{N} u_x(n) i_x(n) \right)$$

N : Anzahl der Messwerte pro Periode

u_x : Spannung der Phase x

i_x : Strom der Phase x

StarApparentP

```
StarApparentP('u1','u2','u3','i1','i2','i3','i4','freq')
```

Berechnet die Scheinleistung in einem Sternnetz:

$$S = U_{eff} I_{eff}$$

U_{eff} : die gemeinsame Effektivspannung

I_{eff} : der gemeinsame Effektivstrom

StarReactiveP

```
StarReactiveP('u1','u2','u3','i1','i2','i3','i4','freq')
```

Berechnet die Blindleistung in einem Sternnetz:

$$Q = \sqrt{S^2 - P^2}$$

S : Scheinleistung

P : Wirkleistung

StarReactivePS

```
StarReactivePS('u1','u2','u3','i1','i2','i3','i4','freq')
```

Berechnet die vorzeichenbehaftete Blindleistung QS im Sternnetz.

StarActivePFactor

```
StarActivePFactor('u1','u2','u3','i1','i2','i3','i4','freq')
```

Berechnet den Wirkleistungsfaktor in einem Sternnetz:

$$\cos\varphi = \frac{P}{S}$$

S : Scheinleistung

P : Wirkleistung

StarReactivePFactor

```
StarReactivePFactor('u1','u2','u3','i1','i2','i3','i4','freq')
```

Berechnet den Blindleistungsfaktor in einem Sternnetz:

$$\tan\varphi = \frac{Q}{P}$$

Q : Blindleistung

P : Wirkleistung

StarReactivePFactorS

```
StarReactivePFactorS('u1','u2','u3','i1','i2','i3','i4','freq')
```

Berechnet den vorzeichenbehafteten Blindleistungsfaktor in einem Sternnetz:

$$\tan\varphi = \frac{Q_S}{P}$$

Q_S : vorzeichenbehaftete Blindleistung

P : Wirkleistung

15.4 Harmonische Funktionen

HarmEff

```
HarmEff('u','Nharm','freq')
```

Berechnet den Effektivwert der 'NHarm'-ten Harmonischen des Signals 'u':

$$u_{Real,k} = \frac{2}{N}\sum_{n=0}^{N-1} u(n)\cos\frac{2\pi kn}{N}$$

$$u_{Imag,k} = \frac{2}{N}\sum_{n=0}^{N-1} u(n)\sin\frac{2\pi kn}{N}$$

$$U_k = \frac{\sqrt{u_{Real,k}^2 + u_{Imag,k}^2}}{\sqrt{2}}$$

u(n) : Messpunkt n von Signal u

$u_{Real,k}$: der Realteil der k-ten harmonischen Komponente von u

$u_{Imag,k}$: der Imaginärteil der k-ten harmonischen Komponente von u

U_k : der Effektivwert der k-ten harmonischen Komponente von u

HarmPhase

`HarmPhase('u','Nharm','freq')`

Berechnet die Phasenverschiebung der 'NHarm'-ten harmonischen Komponente des Signals 'u':

$$\varphi_k = -\operatorname{atan}\left(\frac{u_{Imag,k}}{u_{Real,k}}\right)$$

$u_{Real,k}$: der Realteil der k-ten harmonischen Komponente von u

$u_{Imag,k}$: der Imaginärteil der k-ten harmonischen Komponente von u

ϕ_k : die Phasenverschiebung der k-ten harmonischen Komponente von u

StarHarmUGeff

`StarHarmUGeff('u1','u2','u3','freq')`

Berechnet die effektive Gegensystemspannung UGeff.:

$$U_G = \frac{1}{3}\left[u_{1,1} + u_{2,1}\left(-\frac{2}{3}\pi\right) + u_{3,1}\left(-\frac{4}{3}\pi\right)\right]$$

$$U_{Geff} = \frac{\sqrt{U_{G,real}^2 + U_{G,imag}^2}}{\sqrt{2}}$$

$u_{x,1}$: Grundwellenzeiger (komplex) der Phasenspannung u_x

StarHarmUMeff

```
StarHarmUMeff('u1','u2','u3','freq')
```

Berechnet die Mitsystemspannnung U_{Meff}:

$$U_M = \frac{1}{3}\left[u_{1,1} + u_{2,1}\left(\frac{2}{3}\pi\right) + u_{3,1}\left(\frac{4}{3}\pi\right)\right]$$

$$U_{Meff} = \frac{\sqrt{U_{M,real}^2 + U_{M,imag}^2}}{\sqrt{2}}$$

$u_{x,1}$: Grundwellenzeiger (komplex) der Phasenspannung u_x

StarHarmUnSym

```
StarHarmUnSym('u1','u2','u3','freq')
```

Berechnet die Spannungsunsymmetrie in einem Sternnetz:

$$SYM = \frac{U_{Geff}}{U_{Meff}} \times 100$$

Das Ergebnis wird in % angegeben.

WeightedDistortionFactor

```
WeightedDistortionFactor('u','Nharm'=50,'freq')
```

Berechnet den gewichteten Klirrfaktor von 'u' (aller Phasen) mit 'NHarm' Harmonischen:

$$D_W = \frac{\sqrt{\sum_{n=2}^{Nharm} n^2 U_n^2}}{U_1}$$

U_n : Effektivwert der n-ten Harmonischen von u

UnweightedDistortionFactor

```
UnweightedDistortionFactor('u','Nharm'=50,'freq')
```

Berechnet den ungewichteten Klirrfaktor von 'u' (aller Phasen) mit 'NHarm' Harmonischen:

$$D_{UW} = \frac{\sqrt{\sum_{n=2}^{Nharm} U_n^2}}{\sqrt{\sum_{n=1}^{Nharm} U_n^2}}$$

U_n : Effektivwert der n-ten Harmonischen von u

15.4.1 TIF

```
TIF('u','Nharm'=50,'freq')
```

Berechnet den Telefon-Interferenzfaktor von 'u' unter Berücksichtigung der 'NHarm' Harmonischen:

$$TIF = \frac{1}{U_1}\sqrt{\sum_{n=2}^{Nharm}(K_n \times P_n \times U_n)^2}$$

K_n = 5*n*freq

P_n = Gewichtungsfaktor nach BTS (British Telephone System)

U_n : Effektivwert der Spannung der n-ten Harmonischen von u

U_1 : Effektivwert der Spannung der Grundwelle von u

16 Support und Kontakt

Support

Tel.: +49 911 97282-14

E-Mail: support@iba-ag.com

Hinweis

Wenn Sie Support benötigen, dann geben Sie bitte bei Softwareprodukten die Nummer des Lizenzcontainers an. Bei Hardwareprodukten halten Sie bitte ggf. die Seriennummer des Geräts bereit.

Kontakt

Hausanschrift

iba AG
Königswarterstraße 44
90762 Fürth
Deutschland

Tel.: +49 911 97282-0

E-Mail: iba@iba-ag.com

Postanschrift

iba AG
Postfach 1828
90708 Fürth

Warenanlieferung, Retouren

iba AG
Gebhardtstraße 10
90762 Fürth

Regional und weltweit

Weitere Kontaktadressen unserer regionalen Niederlassungen oder Vertretungen finden Sie auf unserer Webseite:

www.iba-ag.com