A2.4.4 Description of algorithms

Version: 2018-08-17, V0.2

This document describes detailed internal function of algorithms developed in TracePQM activity A2.3.2 and their uncertainty calculation developed in A2.3.5. The algorithm files are located in the TWM project [2] in folder "octprog/QWTB" They are all integrated in the copy of QWTB toolbox [1]. This document won't describe principle of the QWTB toolbox as it is documented on the project web page [1]. The method how the algorithms are called by the TWM toolbox, i.e. what input quantities they receive and what may be returned as a result is defined in the document [4].

In general, the goal of QWTB is to make a wrapper function (next it will be called just "wrapper") that translates the algorithm specific inputs and outputs to a unified format of input and output quantities. This is job of the so called algorithm wrappers: PSFE, SP-WFFT, etc., which are already present in the QWTB toolbox. These wrappers also may or may not contain some uncertainty calculation method or methods. However, non of these wrappers apply any HW component corrections defined by the TWM documents [4], [3]. Therefore, there is a second layer of wrappers (these will be called "TWM wrappers" in the text), which starts with "TWM-" prefix, e.g.: TWM-PSFE, TWM-PWRTDI, etc. The TWM wrappers contain all signal corrections defined by TWM. TWM wrappers perform the necessary TWM correction, they call either a QWTB wrapper (e.g. PSFE) or calculate the result by themselves and combines and returns the corrected results. Note some of the wrappers may call several other wrappers to achieve the desired result. This approach reduces duplication of code in the QWTB toolbox. One of these repeatedly called wrappers is "SP-WFFT" algorithm which is used for spectrum analysis.

References

[1] QWTB toolbox. `https://qwtb.github.io/qwtb/`.

[2] TWM tool. `https://github.com/smaslan/TWM`.

[3] Stanislav Mašláň. Activity A2.3.1 - Correction Files Reference Manual. `https://github.com/smaslan/TWM/tree/master/doc/A231CorrectionFilesReferenceManual.docx`.

[4] Stanislav Mašláň. Activity A2.3.2 - Algorithms Exchange Format. `https://github.com/smaslan/TWM/tree/master/doc/A232AlgorithmExchangeFormat.docx`.

1 TWM-PSFE - Phase Sensitive Frequency Estimator

TWM-PSFE is a TWM wrapper for the Phase Sensitive Frequency Estimator algorithm (PSFE) [1]. PSFE is an algorithm for estimating the frequency, amplitude, and phase of the fundamental component in harmonically distorted waveforms. The algorithm minimizes the phase difference between the sine model and the sampled waveform by effectively minimizing the influence of the harmonic components. It uses a three-parameter sine-fitting algorithm for all phase calculations. The resulting estimates show up to two orders of magnitude smaller sensitivity to harmonic distortions than the results of the four-parameter sine fitting algorithm.

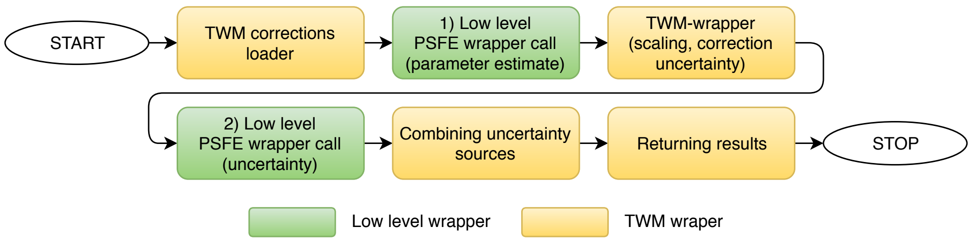

START
TWM corrections loader
1) Low level PSFE wrapper call (parameter estimate)
TWM-wrapper (scaling, correction uncertainty)
2) Low level PSFE wrapper call (uncertainty)
Combining uncertainty sources
Returning results
STOP
Low level wrapper
TWM wraper

Figure 1: Overview of the TWM-PSFE algorithm wrapper.

The algorithm implementation to the TWM structure consists of two levels: (i) wrapper “PSFE” and its uncertainty estimator; (ii) TWM wrapper “TWM-PSFE”. The overall structure is shown in the fig. 1.

The TWM wrapper accepts inputs and corrections (see [3] for details) specified in the table 1. List of output quantities is shown in the table 2. The TWM wrapper also accepts “calcset” options shown in the table 3.

Table 1: List of input quantities to the TWM-PSFE wrapper.

Name	**Default**	**Unc.**	**Description**
comp_timestamp	0	N/A	Enable compensation of phase shift by timestamp value: $\phi' = \phi - 2 \cdot \pi \cdot f\_est \cdot time\_stamp$.
y y_lo	N/A N/A	No No	Input sample data vector and complementary low-side input data vector *y_lo* for differential mode only.
Ts fs t	N/A N/A N/A	No No No	Sampling period or sampling rate or sample time vector. Note the wrapper always calculates in equidistant mode, so t is used just to calculate Ts.
lsb adc_nrng adc_bits lo_lsb lo_adc_nrng lo_adc_bits	N/A 1000 40 N/A 1000 40	No No No No No No	Either absolute ADC resolution *lsb* or nominal range value *adc_nrng* (e.g.: 5 V for 10 Vpp range) and *adc_bits* bit resolution of ADC.
adc_offset lo_adc_offset	0 0	Yes Yes	Digitizer input offset voltage.
adc_gain adc_gain_f adc_gain_a lo_adc_gain lo_adc_gain_f lo_adc_gain_a	1 [] [] 1 [] []	Yes No No Yes No No	Digitizer gain correction 2D table (multiplier).

Table 1: List of input quantities to the TWM-PSFE wrapper.

Name	Default	Unc.	Description
adc_phi	0	Yes	Digitizer phase correction 2D table (additive).
adc_phi_f	[]	No	
adc_phi_a	[]	No	
lo_adc_phi	0	Yes	
lo_adc_phi_f	[]	No	
lo_adc_phi_a	[]	No	
adc_freq	0	Yes	Digitizer timebase error correction: $f\_tb' = f\_tb \cdot (1 + adc\_freq.v)$ The effect on the estimated frequency is opposite: $f\_est' = f\_est/(1 + adc\_freq.v)$
adc_jitter	0	No	Digitizer sampling period jitter [s].
adc_aper	0	No	ADC aperture value [s].
adc_aper_corr	0	No	ADC aperture error correction enable: $A' = A \cdot pi \cdot adc\_aper \cdot f\_est / \sin(pi \cdot adc\_aper \cdot f\_est)$ $phi' = phi + pi \cdot adc\_aper \cdot f\_est$
lo_adc_aper	0		
time_stamp	0	Yes	Relative timestamp of the first sample y.
adc_sfdr	180	No	Digitizer SFDR 2D table.
adc_sfdr_f	[]	No	
adc_sfdr_a	[]	No	
lo_adc_sfdr	180	No	
lo_adc_sfdr_f	[]	No	
lo_adc_sfdr_a	[]	No	
adc_Yin_Cp	1e-15	Yes	Digitizer input admittance 1D table.
adc_Yin_Gp	1e-15	Yes	
adc_Yin_f	[]	No	
lo_adc_Yin_Cp	1e-15	Yes	
lo_adc_Yin_Gp	1e-15	Yes	
lo_adc_Yin_f	[]	No	
tr_type	""	No	Transducer type string ("rvd" or "shunt").
tr_gain	1	Yes	Transducer gain correction 2D table (multiplicative).
tr_gain_f	[]	No	
tr_gain_a	[]	No	
tr_phi	0	Yes	Transducer phase correction 2D table (additive).
tr_phi_f	[]	No	
tr_phi_a	[]	No	
tr_Zlo_Rp	1e3	Yes	RVD transducer low-side impedance 1D table. Note this is related to loading correction and it has effect only for RVD transducer and will work only if adc_Yin is defined as well.
tr_Zlo_Cp	1e-15	Yes	
tr_Zlo_f	[]	No	
tr_Zca_Rs	1e-9	Yes	Loading corrections: Transducer high side terminal series impedance 1D table.
tr_Zca_Ls	1e-12	Yes	
tr_Zca_f	[]	No	
tr_Zcal_Rs	1e-9	Yes	Loading corrections: Transducer low side terminal series impedance 1D table.
tr_Zcal_Ls	1e-12	Yes	
tr_Zcal_f	[]	No	
tr_Yca_Cp	1e-15	Yes	Loading corrections: Transducer output terminals shunting impedance.
tr_Yca_D	1e-12	Yes	
tr_Yca_f	[]	No	
tr_Zcam	1e-12	Yes	Loading corrections: Transducer output terminals mutual inductance 1D table.
tr_Zcam_f	[]	No	
Zcb_Rs	1e-9	Yes	Loading corrections: Cable series impedance 1D table.
Zcb_Ls	1e-12	Yes	
Zcb_f	[]	No	

Table 1: List of input quantities to the TWM-PSFE wrapper.

Name	**Default**	**Unc.**	**Description**
Ycb_Rs	1e-15	Yes	Loading corrections: Cable series impedance 1D table.
Ycb_Ls	1e-12	Yes	
Ycb_f	[]	No	

Table 2: List of output quantities of the TWM-PSFE wrapper. The uncertainty marked * is just a contribution of corrections, but PSFE contribution is not included and not validated.

Name	**Uncertainty**	**Description**
f	Yes	Estimated frequency [Hz].
A	Yes*	Estimated amplitude.
ph	Yes*	Estimated phase angle [rad].

Table 3: List of "calcset" options supported by the TWM-PSFE wrapper.

Name	**Description**
calcset.unc	Uncertainty calculation mode. Supported: "none" or "guf".
calcset.loc	Level of confidence [-].
calcset.verbose	Verbose level.

Block diagram of the internal structure of the TWM-PSFE wrapper is shown in the figure 2. The TWM wrapper partially supports differential transducer input (see [2] for definition). However, in the differential mode it only calculates frequency. The other parameters are ignored. Two differential inputs are directly subtracted $(y - y\_lo)$ in the differential mode. This is not usable for amplitude or phase estimation, but this is sufficient for frequency. The DC offset correction is applied directly to the time domain signal y. Next, the PSFE is called first time to obtain estimates of the unscaled waveform. The uncertainty is disabled, because not all required inputs to PSFE are available at this point. In single-ended mode follow corrections of the estimated signal parameters along with the calculation of the correction uncertainties. When uncertainty calculation is enabled, the additional inputs, such as SFDR and LSB are calculated and PSFE is called again, but this time with uncertainty estimation enabled. The returned estimates are ignored, but the returned uncertainties are combined with the correction contributions and combined with the estimates A, phi and f.

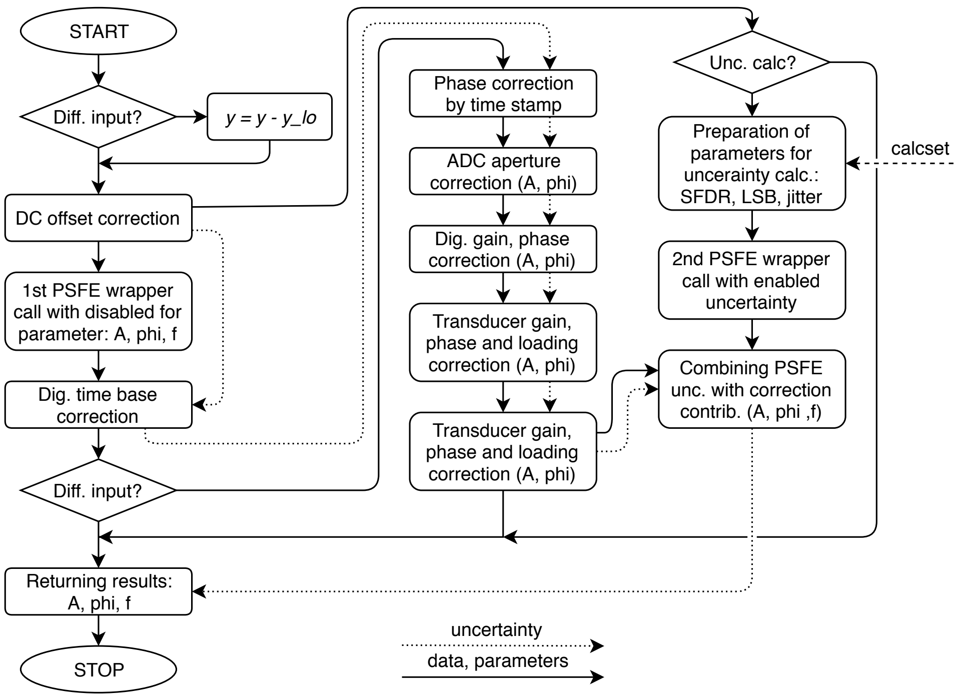

START
Diff. input?
y = y - y_lo
DC offset correction
1st PSFE wrapper call with disabled for parameter: A, phi, f
Dig. time base correction
Diff. input?
Returning results: A, phi, f
STOP
Phase correction by time stamp
ADC aperture correction (A, phi)
Dig. gain, phase correction (A, phi)
Transducer gain, phase and loading correction (A, phi)
Transducer gain, phase and loading correction (A, phi)
Unc. calc?
Preparation of parameters for unceraínty calc.: SFDR, LSB, jitter
calcset
2nd PSFE wrapper call with enabled uncertainty
Combining PSFE unc. with correction contrib. (A, phi ,f)
uncertainty
data, parameters

Figure 2: Detailed internal structure of the TWM-PSFE algorithm wrapper.

SIQ description of the low level PSFE, PSFE wrapper and uncertainty estimation goes here...

References

[1] Rado Lapuh. Estimating the fundamental component of harmonically distorted signals from noncoherently sampled data. *IEEE Transactions on Instrumentation and Measurement*, 64(6):1419–1424, June 2015.

[2] Stanislav Mašláň. Activity A2.3.1 - Correction Files Reference Manual. `https://github.com/smaslan/TWM/tree/master/doc/A231CorrectionFilesReferenceManual.docx`.

[3] Stanislav Mašláň. Activity A2.3.2 - Algorithms Exchange Format. `https://github.com/smaslan/TWM/tree/master/doc/A232AlgorithmExchangeFormat.docx`.

2 TWM-MODTDPS - Modulation analyzer in Time Domain, by quadrature Phase Shifting

TWM-MODTDPS is algorithm for calculation of the modulation parameters of non-coherently sampled signal in time domain. It was designed for basic estimation of the modulation parameters of a sinusoidal waveform modulated by sine wave or rectangular wave with duty cycle of approx. 50 %.

2.1 TWM wrapper parameters

The input quantities supported by the algorithm are shown in the table 4. Algorithm returns output quantities shown in the table 5. Calculation setup supported by the algorithm is shown in table 6.

Table 4: List of input quantities to the TWM-MODTDPS wrapper.
Details on the correction quantities can be found in [3].

Name	Default	Unc.	Description
wave_shape	"sine"	N/A	User string parameter that defines if the algorithm calculates "sine": sinusoidal modulation or "rect": rectangular modulation wave shape.
comp_err	0	N/A	Enable self-compensation of the algorithm error (non-zero value or "on" string).
y y_lo	N/A N/A	No No	Input sample data vector and complementary low-side input data vector y_lo for differential mode only.
Ts fs t	N/A N/A N/A	No No No	Sampling period or sampling rate or sample time vector. Note the wrapper always calculates in equidistant mode, so t is used just to calculate Ts.
lsb adc_nrng adc_bits lo_lsb lo_adc_nrng lo_adc_bits	N/A 1000 40 N/A 1000 40	No No No No No No	Either absolute ADC resolution lsb or nominal range value adc_nrng (e.g.: 5 V for 10 Vpp range) and adc_bits bit resolution of ADC.
adc_gain adc_gain_f adc_gain_a lo_adc_gain lo_adc_gain_f lo_adc_gain_a	1 [] [] 1 [] []	Yes No No Yes No No	Digitizer gain correction 2D table (multiplier).
adc_phi adc_phi_f adc_phi_a lo_adc_phi lo_adc_phi_f lo_adc_phi_a	0 [] [] 0 [] []	Yes No No Yes No No	Digitizer phase correction 2D table (additive).
adc_freq	0	Yes	Digitizer timebase error correction: $f\_tb' = f\_tb \cdot (1 + adc\_freq.v)$ The effect on the estimated frequency is opposite: $f\_est' = f\_est/(1 + adc\_freq.v)$
adc_jitter	0	No	Digitizer sampling period jitter [s].
adc_aper	0	No	ADC aperture value [s].
adc_aper_corr lo_adc_aper	0 0	No	ADC aperture error correction enable: $A' = A \cdot pi \cdot adc\_aper \cdot f\_est/\sin(pi \cdot adc\_aper \cdot f\_est)$ $phi' = phi + pi \cdot adc\_aper \cdot f\_est$
time_stamp	0	Yes	Relative timestamp of the first sample y.
time_shift_lo	0	Yes	Low-side channel time shift [s].

Table 4: List of input quantities to the TWM-MODTDPS wrapper.
Details on the correction quantities can be found in [3].

Name	**Default**	**Unc.**	**Description**
adc_sfdr	180	No	Digitizer SFDR 2D table.
adc_sfdr_f	[]	No	
adc_sfdr_a	[]	No	
lo_adc_sfdr	180	No	
lo_adc_sfdr_f	[]	No	
lo_adc_sfdr_a	[]	No	
adc_Yin_Cp	1e-15	Yes	Digitizer input admittance 1D table.
adc_Yin_Gp	1e-15	Yes	
adc_Yin_f	[]	No	
lo_adc_Yin_Cp	1e-15	Yes	
lo_adc_Yin_Gp	1e-15	Yes	
lo_adc_Yin_f	[]	No	
tr_type	“”	No	Transducer type string (“rvd” or “shunt”).
tr_gain	1	Yes	Transducer gain correction 2D table (multiplicative).
tr_gain_f	[]	No	
tr_gain_a	[]	No	
tr_phi	0	Yes	Transducer phase correction 2D table (additive).
tr_phi_f	[]	No	
tr_phi_a	[]	No	
tr_Zlo_Rp	1e3	Yes	RVD transducer low-side impedance 1D table. Note this is related to loading correction and it has effect only for RVD transducer and will work only if adc_Yin is defined as well.
tr_Zlo_Cp	1e-15	Yes	
tr_Zlo_f	[]	No	
tr_Zca_Rs	1e-9	Yes	Loading corrections: Transducer high side terminal series impedance 1D table.
tr_Zca_Ls	1e-12	Yes	
tr_Zca_f	[]	No	
tr_Zcal_Rs	1e-9	Yes	Loading corrections: Transducer low side terminal series impedance 1D table.
tr_Zcal_Ls	1e-12	Yes	
tr_Zcal_f	[]	No	
tr_Yca_Cp	1e-15	Yes	Loading corrections: Transducer output terminals shunting impedance.
tr_Yca_D	1e-12	Yes	
tr_Yca_f	[]	No	
tr_Zcam	1e-12	Yes	Loading corrections: Transducer output terminals mutual inductance 1D table.
tr_Zcam_f	[]	No	
Zcb_Rs	1e-9	Yes	Loading corrections: Cable series impedance 1D table.
Zcb_Ls	1e-12	Yes	
Zcb_f	[]	No	
Ycb_Rs	1e-15	Yes	Loading corrections: Cable series impedance 1D table.
Ycb_Ls	1e-12	Yes	
Ycb_f	[]	No	

Table 5: List of output quantities of the TWM-MODTDPS wrapper.

Name	**Uncertainty**	**Description**
f0	Yes	Frequency of the carrier [Hz].
A0	Yes	Amplitude of the carrier.
f_mod	Yes	Modulating frequency [Hz].
A_mod	Yes	Modulating amplitude.
mod	Yes	Modulating depth [%].
dVV	Yes	$\Delta V/V$ depth [%]. Alternative expression of *mod*.
cpm	Yes	Changes per minute. Alternative expression of f_mod.
env	No	Modulation envelope.
env_t	No	Modulation envelope *env* time vector.

Table 6: List of "calcset" options supported by the TWM-MODTDPS wrapper.

Name	**Description**
calcset.unc	Uncertainty calculation mode. Supported: "none" or "guf" for uncertainty estimator.
calcset.loc	Level of confidence [-].
calcset.verbose	Verbose level.

2.2 MODTDPS algorithm description

The overview of the TWM wrapper structure is shown in the fig. 3. The algorithm supports differential transducer inputs. The TWM wrapper first estimates carrier frequency $f0$ and mean amplitude of the modulated signal $A0$ by a PSFE algorithm. It uses these two values to obtain and apply gain and aperture error corrections for the high-side channel y (and for low-side channel y_lo in the differential mode).

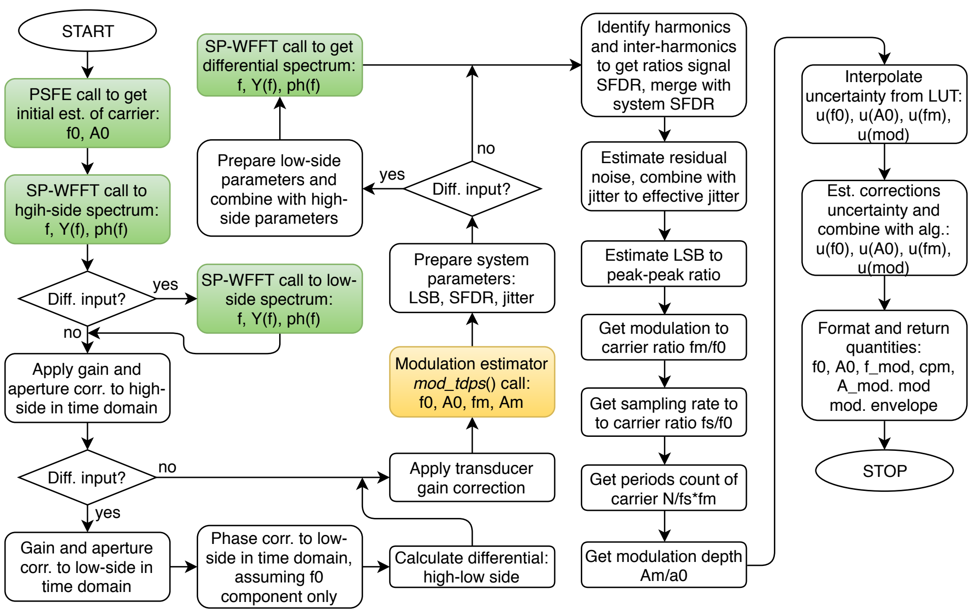

START
PSFE call to get initial est. of carrier: f0, A0
SP-WFFT call to hgih-side spectrum: f, Y(f), ph(f)
Diff. input?
yes
no
SP-WFFT call to low-side spectrum: f, Y(f), ph(f)
Apply gain and aperture corr. to high-side in time domain
Diff. input?
no
yes
Gain and aperture corr. to low-side in time domain
Phase corr. to low-side in time domain, assuming f0 component only
Calculate differential: high-low side
Apply transducer gain correction
Modulation estimator mod_tdps() call: f0, A0, fm, Am
Prepare system parameters: LSB, SFDR, jitter
Diff. input?
yes
no
Prepare low-side parameters and combine with high-side parameters
SP-WFFT call to get differential spectrum: f, Y(f), ph(f)
Identify harmonics and inter-harmonics to get ratios signal SFDR, merge with system SFDR
Estimate residual noise, combine with jitter to effective jitter
Estimate LSB to peak-peak ratio
Get modulation to carrier ratio fm/f0
Get sampling rate to to carrier ratio fs/f0
Get periods count of carrier N/fs*fm
Get modulation depth Am/a0
Interpolate uncertainty from LUT: u(f0), u(A0), u(fm), u(mod)
Est. corrections uncertainty and combine with alg.: u(f0), u(A0), u(fm), u(mod)
Format and return quantities: f0, A0, f_mod, cpm, A_mod. mod mod. envelope
STOP

Figure 3: Overview of the TWM-MODTDPS algorithm wrapper. Green cells are calls to other QWTB wrappers. Gold cells are calls of the local functions which are described in the text.

In the differential transducer mode, the wrapper also applies high-low side time shift correction and phase correction to the low-side input y_lo by time shifting it according to the estimate $f0$. This trivial phase synchronization of the high-low side phase obviously works only to one frequency $f0$ and it is dependent on its correct estimation, however it turned out to be sufficient for the purposes of this algorithm. Next, the wrapper calculates voltage difference $y = y - y\_lo$, so the differential is reduced to single ended input y. The transducer gain correction in the differential mode uses additional voltage vectors $Y(f0), phi(f0)$ and $Y\_lo(f0), phi\_lo(f0)$ obtained from the two spectra SP-WFFT. Although the vectors have absolute values distorted by the spectral leakage, their ratio stays fixed, so it is enough for the transducer transfer and loading correction function. At this point the signal is single ended and scaled.

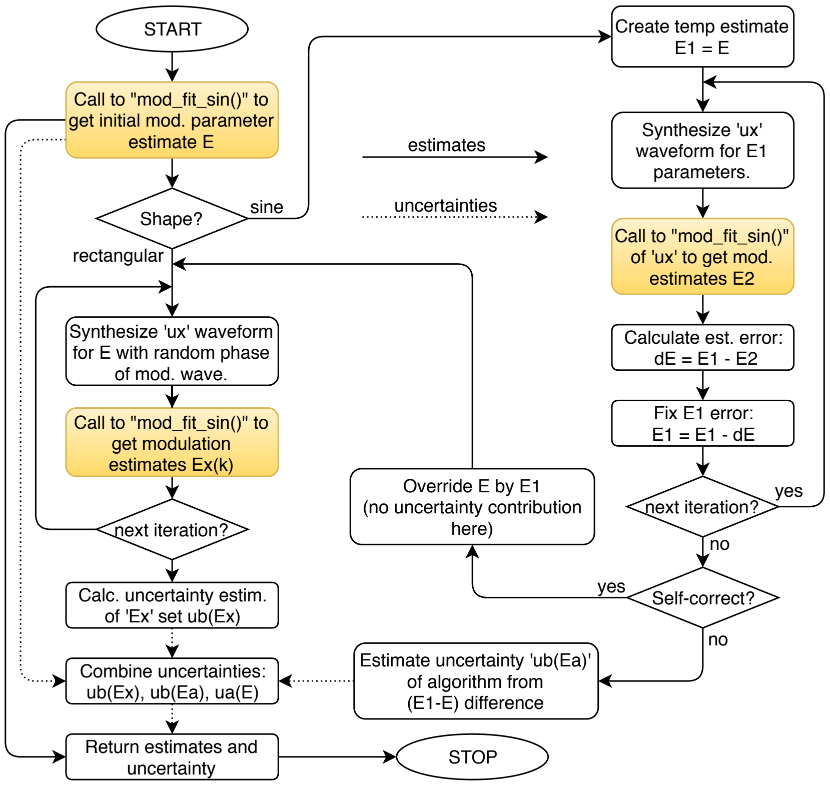

START
Call to "mod_fit_sin()" to get initial mod. parameter estimate E
Shape?
sine
rectangular
Synthesize 'ux' waveform for E with random phase of mod. wave.
Call to "mod_fit_sin()" to get modulation estimates Ex(k)
next iteration?
Calc. uncertainty estim. of 'Ex' set ub(Ex)
Combine uncertainties: ub(Ex), ub(Ea), ua(E)
Return estimates and uncertainty
STOP
estimates
uncertainties
Create temp estimate E1 = E
Synthesize 'ux' waveform for E1 parameters.
Call to "mod_fit_sin()" of 'ux' to get mod. estimates E2
Calculate est. error: dE = E1 - E2
Fix E1 error: E1 = E1 - dE
next iteration?
yes
no
Self-correct?
yes
no
Override E by E1 (no uncertainty contribution here)
Estimate uncertainty 'ub(Ea)' of algorithm from (E1-E) difference

Figure 4: Overview of the "mod_tdps()" function of the TWM-MODTDPS algorithm wrapper. Gold cells are calls of local functions of the algorithm.

The next step is the main algorithm for the estimation of modulation parameters "mod_tdps()" which is shown in the fig. 4. The algorithm internally calls the function "mod_fit_sin()" (see fig. 5), which does the parameter estimation itself.

The algorithm itself is based on the estimation of the carrier frequency $f0$ by means of PSFE algorithm [2]. Once the carrier $f0$ is known, the algorithm applies 90deg phase shift to the input signal y and builds two virtual quadrature signals:

$$ya(t) = y(t) + j \cdot y\left(t + \frac{pi}{2 \cdot f0}\right), \tag{1}$$

$$yb(t) = y(t) - j \cdot y\left(t - \frac{pi}{2 \cdot f0}\right), \tag{2}$$

(3)

The average of amplitudes of the signals $ya(t)$ and $yb(t)$ is roughly equal to the modulation envelope:

$$ev(t) = 0.5 \cdot (|ya(t)| + |yb(t)|) \tag{4}$$

The envelope $ev(t)$ is used as an input to the next call of the PSFE algorithm which returns modulation amplitude Am, modulation frequency fm and modulation phase phm. The algorithm differs for the sinusoidal and rectangular wave shape from this point. In the sinusoidal mode, the carrier amplitude

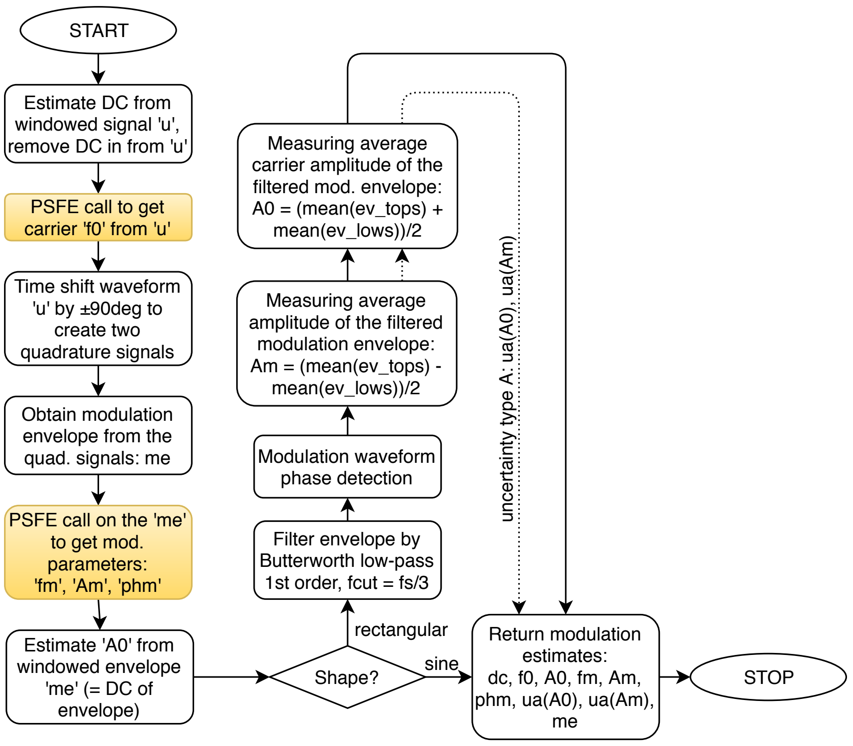

START
Estimate DC from windowed signal 'u', remove DC in from 'u'
PSFE call to get carrier 'f0' from 'u'
Time shift waveform 'u' by ±90deg to create two quadrature signals
Obtain modulation envelope from the quad. signals: me
PSFE call on the 'me' to get mod. parameters: 'fm', 'Am', 'phm'
Estimate 'A0' from windowed envelope 'me' (= DC of envelope)
Shape?
rectangular
sine
Filter envelope by Butterworth low-pass 1st order, fcut = fs/3
Modulation waveform phase detection
Measuring average amplitude of the filtered modulation envelope: Am = (mean(ev_tops) - mean(ev_lows))/2
Measuring average carrier amplitude of the filtered mod. envelope: A0 = (mean(ev_tops) + mean(ev_lows))/2
uncertainty type A: ua(A0), ua(Am)
Return modulation estimates: dc, f0, A0, fm, Am, phm, ua(A0), ua(Am), me
STOP

Figure 5: Overview of the "mod_fit_sin()" function of the TWM-MODTDPS algorithm wrapper. Gold cells are calls of local functions of the algorithm.

$A0$ is obtained as a DC value of the envelope $ev(t)$ using a windowed average method with Blackman window:

$$A0 = \frac{\sum_{t=1}^{N} w(t) \cdot e(t)}{\sum_{t=1}^{N} w(t)}, \tag{5}$$

where the $w(t)$ are coefficients of the Blackman window and N is samples count of the envelope. This trivial method obtains acceptable suppression of errors caused by the non-coherent window size if at least three modulating periods were recorded. In the rectangular mode, the $A0$ from PSFE cannot be used. Only the modulation frequency estimate fm and phase estimate phm are relevant. The envelope $ev(t)$ is filtered by a low-pass 1st order Butterworth filter with cutoff frequency $fs/3$. This reduces the noise caused by the harmonic and inter-harmonic spurs present in the envelope $ev(t)$ but does not distort the shape too much at high modulation frequencies. Next, the phase phm of the modulation wave is used to detect the tops and lows of the filtered envelope rectangular wave. It was experimentally decided to use 15 % to 30 % of the periods as a tops and 75 % to 85 % as lows. The modulation parameters are calculated according to formulas:

$$A0 = \frac{1}{2M} \sum_{m=1}^{M} [\text{tops}\{ev(t), m\} - \text{lows}\{ev(t), m\}], \tag{6}$$

$$Am = \frac{1}{2M} \sum_{m=1}^{M} [\text{tops}\{ev(t), m\} + \text{lows}\{ev(t), m\}], \tag{7}$$

where m is the period index and M is total count of modulation periods in the signal. The type A uncertainty estimate is calculated from the differences between the periods m.

In the sine wave mode, the algorithm also contains a self-correcting routine that is capable to reduce the inherent error of the algorithm itself (see diagram in fig. 4). The idea is following: First, the algorithm core function "mod_fit_sin()" is called on the real waveform data y to obtain the initial estimate E of the modulation parameters. Next, a new simulated waveform is synthesized so it has the modulation parameters E and core function "mod_fit_sin()" is called again on the waveform for $E1$ to obtain estimate

$E2$. Finally, an algorithm error $\mathrm{d}E = E2 - E1$ is calculated. The whole operation is repeated three times in a loop which was sufficient to get stable error $\mathrm{d}E$. The $\mathrm{d}E$ is either used as a correction to E (when self-correction is enabled) or is used to estimate algorithm error uncertainty contribution, when self-correction is disabled. This method significantly reduced the error of the algorithm even for high modulation frequencies. The performance was evaluated so the uncertainty calculation reflects sensitivity of this method to the imperfect input signal.

The "mod_tdps()" function automatically calculates estimate of maximum error caused by the uncertain phase shift of the modulating waveform. It is calculated by repeating the estimator 10 times for different phase shifts and calculating maximum error. This is part of the total uncertainty budget.

2.3 Uncertainty estimator

The algorithm is too complex for GUF uncertainty calculation. It is also relatively slow, so the Monte-Carlo uncertainty calculation for an interactive application would be too slow especially for waveforms longer than few thousand samples. Therefore, a fast uncertainty estimator was developed. The estimator is based on the massive lookup tables (LUT) that contains precalculated uncertainties for various combinations of the parameters of the input signal.

First step for creation of the estimator was selection of the relevant signal parameters. The set was chosen so it is minimalist, because each parameter means one more dimension of the simulation and thus additional data in the LUT. Selection is follows:

- **Modulating periods count**: The count of modulating signal periods in the recored waveform.
- **Samples per period of carrier**: The ratio of sampling rate and carrier frequency $fs/f0$.
- **Relative modulation frequency**: The ratio of the modulating frequency to carrier $fm/f0$.
- **Total SFDR**: Combination of system SFDR (corrections) and signal SFDR (harmonics and interhamonics).
- **Effective jitter**: Total effective sampling jitter in seconds. This also includes equivalent value of the residual RMS noise found in the signal. The jitter value is normalized to the carrier frequency.
- **Bit resolution**: The bits count per used peak-to-peak ADC range. This is theoretically replaceable by the jitter (resp. noise), but it may easily lead to nonlinear behaviour for low resolutions. Therefore, this parameter was simulated separately.
- **Modulation depth**: The ratio of the modulating amplitude to the carrier amplitude $Am/A0$.

The simulation ranges of the parameters were chosen according to the table 7. The ranges were chosen to cover the typical operating range, however most of the dependencies is extrapolable in one direction.

Table 7: Simulation ranges and steps of the parameters for uncertainty estimator of MODTDPS algorithm.

Name	**Description**
Modulating periods count	Log. space: 3 to 30, 6 steps
Samples per period of carrier	Log. space: 10 to 100, 5 steps
Relative modulation frequency	Log. space: 0.01 to 0.33, 8 steps
Total SFDR	List: [120; 80; 60; 30] dB, 4 steps
Effective jitter	Log. space: 10^{-9} to 10^{-2}, 5 steps
Bit resolution	Log. space: 6 to 24 bits, 6 steps
Modulation depth	Log. space: 0.01 to 0.99, 8 steps

Fro simplicity it was assumed all the parameters may be correlated, so all combinations of the seven parameters were generated (6x5x8x4x5x6x8 = 230400 combinations). At least 1000 Monte-Carlo (MC) iteration cycles of following sequence of operations was performed for each combination:

- Get one combination of simulation parameters E_{ref}.

- Randomize E_{ref} parameters in a small range (few percent), so each MC iteration generates a bit different signal. This is to prevent unfortunate selection of a combination E_{ref} where the uncertainty is exceptionally low, e.g. due to the coherent sampling.
- Generate other random parameters, such as DC offset, phase shift of the modulation signal, random spurs up to SFDR parameter value, etc.
- Synthesize modulated waveform of known parameters.
- Distort the waveform by: spurs, jitter, quantisation, etc.
- Perform estimation of the modulation parameters E_x by "mod_tdps()" algorithm. Note the uncertainties returned by the "mod_tdps()" itself are ignored, as they will be calculated on runtime during actual measurements.
- Compare estimates E_x to generated parameters E_{ref}: $\Delta E_x(k) = E_x - E_{ref}$.

The set of algorithm errors $\Delta E_x(k)$ from the MC iterations k for each combination of parameters is processed according to the GUM guide, supplement 1 [1]. The whole batch of combinations was processed on the supercomputer, so it took only three days per configuration ("sine", "rect", with or without self-corrections). The 1000 MC cycles was enough to obtain stable estimates. Output of the calculation is 7-dimensional matrix of uncertainties of modulation parameters: $f0$, fm, $A0$ and Am. The 7D array was manually inspected along various axes (= along simulation parameters), however it was not possible to find a simple empiric formulas that would cover full range of any axis. There was always some non-linearity dependent on the other axes. All tries resulted either in significant over or underestimation of uncertainty in some part of the parameter space.

Therefore, the whole 7D matrix was simply compressed to the log. space and 16bit integers (resolution better 0.005) and saved to a compressed MAT file as lookup table (LUT). The size of LUT is roughly 1.7 MBytes per configuration which is still acceptable and thus it was decided to not continue with further optimalisations. The LUT contains definitions of the axes (parameters), their permissible ranges, interpolation modes (linear or logarithmic) and definition of the estimator action, when the parameter is out of range (error or limit at max/min value). A multidimensional interpolator was developed which is capable to read the LUT and return interpolated values (or errors) of the quantities stored in the LUT. The usable range of parameters is shown in the table 8. Note the interpolator permits to extrapolate a outside the stated limits, which should prevent problems around limits. The additional permissible range is set to up to $\pm 5\,\%$ of given range.

Table 8: Permissible range of signal parameters for the uncertainty estimator. The values in parenthesis are permissible, but outside simulation range. The actions on min or max value is reached are: "error" - generate error; "const" - return value of uncertainty at min. or max. of simulated range.

Name	**Range**	**On min**	**On max**
Modulating periods count	3 to 30 (3 to ∞)	error	const
Samples per period of carrier	10 to 100 (10 to ∞)	error	const
Relative modulation frequency	0.01 to 0.33 (0 to 0.33)	const	error
Total SFDR	120 to 30 dB (∞ to 30 dB)	error	const
Effective jitter	10^{-9} to 10^{-2} ($10^{-\infty}$ to 10^{-2})	const	error
Bit resolution	6 to 24 bits (6 to ∞ bits)	error	const
Modulation depth	0.01 to 0.99	error	error

The estimator itself in the TWM-MODTDPS wrapper is based on the estimated modulation parameters and spectrum analysis of the input signal y. It obtains the parameters of the LUT axes by following procedure:

- Calculate the basic parameters from corrections and estimated modulation parameters: (i) Modulating periods count; (ii) Samples per period of carrier; (iii) Relative modulation frequency; (iv) Modulation depth; (v) Bit resolution.

- Perform spectrum analysis to obtain: (i) Harmonics (except the ones belonging to modulation sidebands); (ii) Interharmonics; (iii) RMS noise estimate. These value are used to calculate signal SFDR estimate and noise, which is converted to equivalent jitter at the carrier frequency $f0$.
- Interpolated the LUT table for given configuration to get the algorithm uncertainty.
- Calculate estimate of uncertainty of the corrections. This covers estimate of the error caused by the fact the signal scaling is happening at a single frequency spot $f0$ instead of complicated frequency dependent correction.
- Combine uncertainties: (i) Runtime calculated uncertainty from "mod_tdps()" itself; (ii) Uncertainty from the LUT table; (iii) Uncertainty of the corrections.

References

[1] JCGM. *Evaluation of measurement data - Supplement 1 to the "Guide to the expression of uncertainty in measurement" - Propagation of distributions using a Monte Carlo method.* Bureau International des Poids et Measures.

[2] Rado Lapuh. Estimating the fundamental component of harmonically distorted signals from noncoherently sampled data. *IEEE Transactions on Instrumentation and Measurement*, 64(6):1419–1424, June 2015.

[3] Stanislav Mašláň. Activity A2.3.2 - Algorithms Exchange Format. `https://github.com/smaslan/TWM/tree/master/doc/A232AlgorithmExchangeFormat.docx`.

3 TWM-FPNLSF - Four Parameter Non Linear Sine Fit

This algorithm fits a sine wave to the recorded data by means of non-linear least squares fitting method using 4 parameter (frequency, amplitude, phase and offset) model. An estimate of signal frequency is required with accuracy at least 500 ppm. Due to non-linear characteristic, convergence is not always achieved. When run in Matlab, function "lsqnonlin" in Optimization toolbox is used. When run in GNU Octave, function "leasqr" in GNU Octave Forge package optim is used. Therefore results can differ. The integrated uncertainty estimator was developed only for the GNU Octave version. This should be still kept in mind when using the algorithm with Matlab despite the Matlab version seems to give always more accurate results than GNU Octave!

3.1 TWM wrapper parameters

The input quantities supported by the algorithm are shown in the table 9. Algorithm returns output quantities shown in the table 10. Calculation setup supported by the algorithm is shown in table 11.

Table 9: List of input quantities to the TWM-FPNLSF wrapper.
Details on the correction quantities can be found in [3].

Name	Default	Unc.	Description
f_est	N/A	N/A	Initial estimate of the sine frequency. The estimate should be accurate to at least 500 ppm.
comp_timestamp	0	N/A	Enable compensation of phase shift by time stamp value: $phi' = phi - 2 \cdot pi \cdot f\_fit \cdot time\_stamp$.
y y_lo	N/A N/A	No No	Input sample data vector and complementary low-side input data vector *y_lo* for differential mode only.
Ts fs t	N/A N/A N/A	No No No	Sampling period or sampling rate or sample time vector. Note the wrapper always calculates in equidistant mode, so *t* is used just to calculate *Ts*.
lsb adc_nrng adc_bits lo_lsb lo_adc_nrng lo_adc_bits	N/A 1000 40 N/A 1000 40	No No No No No No	Either absolute ADC resolution *lsb* or nominal range value *adc_nrng* (e.g.: 5 V for 10 Vpp range) and *adc_bits* bit resolution of ADC.
adc_offset lo_adc_offset	0 0	Yes Yes	Digitizer input offset voltage.
adc_gain adc_gain_f adc_gain_a lo_adc_gain lo_adc_gain_f lo_adc_gain_a	1 [] [] 1 [] []	Yes No No Yes No No	Digitizer gain correction 2D table (multiplier).
adc_phi adc_phi_f adc_phi_a lo_adc_phi lo_adc_phi_f lo_adc_phi_a	0 [] [] 0 [] []	Yes No No Yes No No	Digitizer phase correction 2D table (additive).
adc_freq	0	Yes	Digitizer timebase error correction: $f\_tb' = f\_tb \cdot (1 + adc\_freq.v)$ The effect on the estimated frequency is opposite: $f\_est' = f\_est/(1 + adc\_freq.v)$
adc_jitter	0	No	Digitizer sampling period jitter [s].
adc_aper	0	No	ADC aperture value [s].

Table 9: List of input quantities to the TWM-FPNLSF wrapper.
Details on the correction quantities can be found in [3].

Name	**Default**	**Unc.**	**Description**
adc_aper_corr lo_adc_aper	0 0	No	ADC aperture error correction enable: $A' = A \cdot pi \cdot adc\_aper \cdot f\_est / \sin(pi \cdot adc\_aper \cdot f\_est)$ $phi' = phi + pi \cdot adc\_aper \cdot f\_est$
time_stamp	0	Yes	Relative timestamp of the first sample y.
time_shift_lo	0	Yes	Low-side channel time shift [s].
adc_sfdr adc_sfdr_f adc_sfdr_a lo_adc_sfdr lo_adc_sfdr_f lo_adc_sfdr_a	180 [] [] 180 [] []	No No No No No No	Digitizer SFDR 2D table.
adc_Yin_Cp adc_Yin_Gp adc_Yin_f lo_adc_Yin_Cp lo_adc_Yin_Gp lo_adc_Yin_f	1e-15 1e-15 [] 1e-15 1e-15 []	Yes Yes No Yes Yes No	Digitizer input admittance 1D table.
tr_type	“”	No	Transducer type string (“rvd” or “shunt”).
tr_gain tr_gain_f tr_gain_a	1 [] []	Yes No No	Transducer gain correction 2D table (multiplicative).
tr_phi tr_phi_f tr_phi_a	0 [] []	Yes No No	Transducer phase correction 2D table (additive).
tr_Zlo_Rp tr_Zlo_Cp tr_Zlo_f	1e3 1e-15 []	Yes Yes No	RVD transducer low-side impedance 1D table. Note this is related to loading correction and it has effect only for RVD transducer and will work only if adc_Yin is defined as well.
tr_Zca_Rs tr_Zca_Ls tr_Zca_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Transducer high side terminal series impedance 1D table.
tr_Zcal_Rs tr_Zcal_Ls tr_Zcal_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Transducer low side terminal series impedance 1D table.
tr_Yca_Cp tr_Yca_D tr_Yca_f	1e-15 1e-12 []	Yes Yes No	Loading corrections: Transducer output terminals shunting impedance.
tr_Zcam tr_Zcam_f	1e-12 []	Yes No	Loading corrections: Transducer output terminals mutual inductance 1D table.
Zcb_Rs Zcb_Ls Zcb_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Cable series impedance 1D table.
Ycb_Rs Ycb_Ls Ycb_f	1e-15 1e-12 []	Yes Yes No	Loading corrections: Cable series impedance 1D table.

Table 10: List of output quantities of the TWM-FPNLSF wrapper.

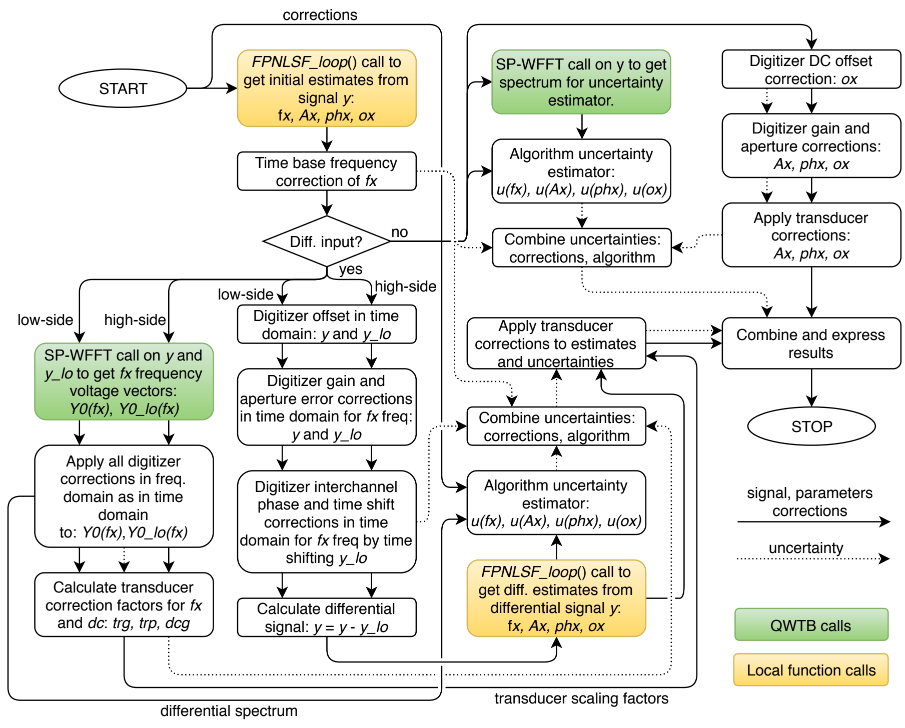
Name	Uncertainty	Description
f	Yes	Frequency of the carrier [Hz].
A	Yes	Amplitude of the carrier.
phi	Yes	Phase of main signal component [rad].
ofs	Yes	DC offset of signal.

Table 11: List of "calcset" options supported by the TWM-FPNLSF wrapper.

Name	Description
calcset.unc	Uncertainty calculation mode. Supported: "none" or "guf" for uncertainty estimator.
calcset.loc	Level of confidence [-].
calcset.verbose	Verbose level.

3.2 Algorithm description

The wrapper TWM-FPNSLF overview is shown in the fig. 6. It first calls the core function "FPNLSF_loop()" on the unscaled high-side input signal y to get initial estimate of the signal frequency fx. This is necessary to get gain and phase correcting coefficients. Follows the signal scaling in the time domain, i.e. application of the digitizer DC offset, gain, phase and aperture corrections.



corrections
START
FPNLSF_loop() call to get initial estimates from signal y: fx, Ax, phx, ox
Time base frequency correction of fx
Diff. input?
no
yes
low-side
high-side
low-side
high-side
SP-WFFT call on y and y_lo to get fx frequency voltage vectors: Y0(fx), Y0_lo(fx)
Apply all digitizer corrections in freq. domain as in time domain to: Y0(fx),Y0_lo(fx)
Calculate transducer correction factors for fx and dc: trg, trp, dcg
Digitizer offset in time domain: y and y_lo
Digitizer gain and aperture error corrections in time domain for fx freq: y and y_lo
Digitizer interchannel phase and time shift corrections in time domain for fx freq by time shifting y_lo
Calculate differential signal: y = y - y_lo
SP-WFFT call on y to get spectrum for uncertainty estimator.
Algorithm uncertainty estimator: u(fx), u(Ax), u(phx), u(ox)
Combine uncertainties: corrections, algorithm
Apply transducer corrections to estimates and uncertainties
Combine uncertainties: corrections, algorithm
Algorithm uncertainty estimator: u(fx), u(Ax), u(phx), u(ox)
FPNLSF_loop() call to get diff. estimates from differential signal y: fx, Ax, phx, ox
Digitizer DC offset correction: ox
Digitizer gain and aperture corrections: Ax, phx, ox
Apply transducer corrections: Ax, phx, ox
Combine and express results
STOP
signal, parameters corrections
uncertainty
QWTB calls
Local function calls
differential spectrum
transducer scaling factors

Figure 6: Overview of the TWM-FPNLSF algorithm wrapper.

The wrapper also allows differential input sensor connection. In this case it compensates the high-low side phase error by time shifting the low-side signal y_lo according to the estimated frequency component fx. Such a phase correction of course works only for the single frequency component fx, but the results were acceptable as it is the main signal component. Next, it calculates differential signal $y_d = y - y\_lo$. Only additional difference is the TWM defines relatively complex transducer loading corrections scheme (see [2]). This is ensured by the function "correction_transducer_loading()". However, the function operates in frequency domain, while FPNLSF operates in time domain and the algorithm expects non-coherent sampling. Therefore, an additional step is done to obtain scaling transducer factor. The windowed FFT of the high and low-side signals y and y_lo is calculated by SP-WFFT algorithm. The voltage vectors are obtained from the FFTs and used as an inputs to the "correction_transducer_loading()". Although the voltage vectors are distorted by the spectral leakage, their ratio stays unaffected, so the transducer scaling factor obtained from the "correction_transducer_loading()" is sufficiently accurate. At this point, the differential signals y and y_lo are reduced to single ended and scaled. The "FPNLSF_loop()" is called again on the differential signal y_d. Next calculation steps are identical for single ended and differential modes.

The FPNLSF algorithm itself and its uncertainty analysis was described in [4]. The basic principle is use of non-linear least square minimising algorithm to fit the input signal y by a four parameter sine wave model:

$$y_{\rm m} = o + A \cdot \sin\left(2\pi f t + \phi\right), \tag{8}$$

where o is DC offset, A is amplitude, t is time vector, f is sine frequency and ϕ is phase angle. This method is quite sensitive to harmonics and especially interharmonics and also requires good initial estimates for the minimising algorithm, however for clean signals it offers acceptable estimates of the fundamental component.

The core function of the TWM-FPNLSF wrapper is function "FPNLSF_loop()" whose structure is shown in fig. 7. The function accompanies the FPNLSF algorithm itself by several supporting functions. First major problem to solve was its sensitivity to the precision of initial estimate of the parameters, especially frequency f_est. It was merely impossible to perform the Monte Carlo (MC) uncertainty calculation of the FPNLSF itself as the FPNLSF minimising process often ended in a local minima, which is not always detectable, so the histogram of the MC calculation contained many far outliers, which made the uncertainty unusable. Therefore, the permissible range of initial estimate f_est was set to ±500 ppm from the actual signal frequency. The FPNLSF was placed in a retry loop that tries repeatedly run the FPNLSF with slightly randomised initial estimates until the fitted frequency f is within the ±500 ppm range. The retry loop also contains limit for the total retries count and total timeout, so it won't get locked up. The loop was also accompanied by initial zero cross estimation, which tries to obtain at least approximate initial phase estimate.

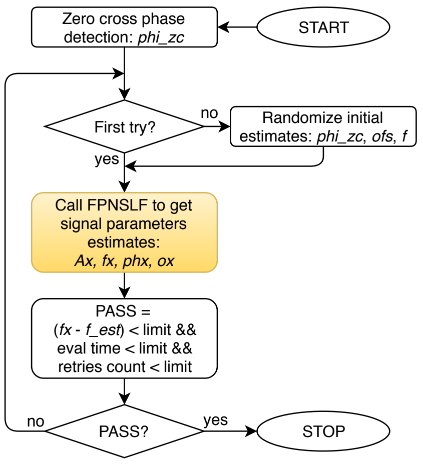

START
Zero cross phase detection: phi_zc
First try?
no
Randomize initial estimates: phi_zc, ofs, f
yes
Call FPNSLF to get signal parameters estimates: Ax, fx, phx, ox
PASS = (fx - f_est) < limit && eval time < limit && retries count < limit
PASS?
no
yes
STOP

Figure 7: The structure of the "FPNLSF_loop()" function of the TWM-FPNLSF algorithm wrapper.

3.3 Uncertainty estimator

The algorithm is too complex for GUF uncertainty calculation. It is also relatively slow, so the Monte-Carlo uncertainty calculation for an interactive application would be too slow especially for waveforms longer than few thousand samples. Therefore, a fast uncertainty estimator was developed. The estimator is based on a combination of the empirical formulas and the lookup tables (LUT) that contains precalculated uncertainties for various combinations of the input signal parameters.

First step for creation of the estimator was selection of the relevant signal parameters. The set was chosen so it is minimalist, because each parameter means one more dimension of the simulation and thus additional data in the LUT. Selection is follows:

- **Periods count**: The count of signal periods in the recored waveform.
- **Samples per period**: The ratio of sampling rate and carrier frequency $fs/f0$.
- **Total SFDR**: Combination of system SFDR (corrections) and signal SFDR (harmonics and interhamonics).
- **Effective jitter**: Total effective sampling jitter in seconds. This also includes equivalent value of the residual RMS noise found in the signal. The jitter value is normalized to the carrier frequency.
- **Bit resolution**: The bits count per used peak-to-peak ADC range. This is theoretically replaceable by the jitter (resp. noise), but it may easily lead to nonlinear behaviour for low resolutions. Therefore, this parameter was simulated separately.

The simulation ranges of the parameters were chosen according to the table 12. The ranges were chosen to cover the typical operating range, however most of the dependencies is extrapolable in one direction.

Table 12: Simulation ranges and steps of the parameters for uncertainty estimator of "FPNLSF_loop()" function.

Name	**Description**
Periods count	List: [10; 20; 50; 100], 4 steps
Samples per period	Log. space: 10 to 1000, 10 steps
Total SFDR	List: [180; 120; 80; 40; 30] dB, 4 steps
Effective jitter	Log. space: 10^{-9} to 10^{-2}, 9 steps
Bit resolution	Log. space: 4 to 24 bits, 8 steps

Fro simplicity it was assumed all the parameters may be correlated, so all combinations of the five parameters were generated (4x10x4x9x8 = 11520 combinations). 1000 Monte-Carlo (MC) iteration cycles of following sequence of operations was performed for each combination:

- Get one combination of simulation parameters E_{ref}.
- Randomize E_{ref} parameters in a small range (few percent), so each MC iteration generates a bit different signal. This is to prevent unfortunate selection of a combination E_{ref} where the uncertainty is exceptionally low, e.g. due to the coherent sampling.
- Generate other random parameters, such as DC offset, phase shift of the signal, random spurs up to SFDR parameter value, etc.
- Synthesize modulated waveform of known parameters.
- Distort the waveform by: spurs, jitter, quantisation, etc.
- Perform estimation of the signal parameters E_x by "FPNLSF_loop()" algorithm.
- Compare estimates E_x to generated parameters E_{ref}: $\Delta E_x(k) = E_x - E_{ref}$.

The set of algorithm errors $\Delta E_x(k)$ from the MC iterations k for each combination of parameters was processed according to the GUM guide, supplement 1 [1]. The whole batch of combinations was processed on the supercomputer, so it took only two days. The 1000 MC cycles was enough to obtain stable estimates. Output of the calculation was a 5-dimensional matrix of uncertainties of signal parameters estimates: frequency, amplitude, phase and DC offset. The 5D array was manually inspected along various axes (= along simulation parameters) to verify there are no extrema. As the array is relatively small it was decided to not look for empirical formulas to reduce the axes. Therefore, the whole 5D matrix was simply compressed to the log. space and 16bit integers (resolution better 0.005) and saved to a compressed MAT file as lookup table (LUT). The size of LUT is roughly 120 kBytes, which is still acceptable. The LUT contains definitions of the axes (parameters), their permissible ranges, interpolation modes (linear or logarithmic) and definition of the estimator action, when the parameter is out of range (error or limit at max/min value). A multidimensional interpolator was developed which is capable to read the LUT and return interpolated values (or errors) of the quantities stored in the LUT. The usable range of parameters is shown in the table 13. Note the interpolator permits to extrapolate a outside the stated limits, which should prevent problems around limits. The additional permissible range is set to up to ±5 % of given range.

Table 13: Permissible range of signal parameters for the uncertainty estimator of "FPNLSF_loop()" function. The values in parenthesis are permissible, but outside simulation range. The actions on min or max value is reached are: "error" - generate error; "const" - return value of uncertainty at min. or max. of simulated range.

Name	**Range**	**On min**	**On max**
Periods count	10 to 100 (10 to ∞)	error	const
Samples per period	10 to 1000 (10 to ∞)	error	const
Total SFDR	180 to 30 dB (∞ to 30 dB)	error	const
Effective jitter	10^{-9} to 10^{-2} ($10^{-\infty}$ to 10^{-2})	const	error
Bit resolution	4 to 24 bits (4 to ∞ bits)	error	const

The estimator itself in the TWM-FPNLSF wrapper is based on the estimated parameters of the signal returned by the "FPNLSF_loop()" and spectrum analysis of the input signal y. It obtains the parameters of the LUT axes by following procedure:

- Calculate the basic parameters from corrections and estimated parameters: (i) Periods count; (ii) Samples per period; (iii) Bit resolution.
- Perform spectrum analysis of y (or y_d for differential mode) to obtain: (i) Harmonics; (ii) Interharmonics; (iii) RMS noise estimate. These value are used to calculate signal SFDR estimate and noise, which is converted to the equivalent jitter at the fitted frequency fx.
- Interpolate the LUT table for given parameters to get the algorithm uncertainty.
- Calculate estimate of uncertainty of the corrections.
- Combine uncertainties: (i) Uncertainty from the LUT table; (ii) Uncertainty of the corrections.

Note the precalculated LUT was calculated on the GNU Octave system, which is using different minimising algorithm than Matlab. However, the long validation test was performed with generating many random signal parameters and no case where the calculated estimates were outside the uncertainty were found. In fact, the Matlab version seems to give much better results than GNU Octave. However, this fact should be still kept in mind, as there may be some case, where Matlab performs worse.

References

[1] JCGM. *Evaluation of measurement data - Supplement 1 to the "Guide to the expression of uncertainty in measurement" - Propagation of distributions using a Monte Carlo method.* Bureau International des Poids et Measures.

[2] Stanislav Mašláň. Activity A2.3.1 - Correction Files Reference Manual. `https://github.com/smaslan/TWM/tree/master/doc/A231CorrectionFilesReferenceManual.docx`.

[3] Stanislav Mašláň. Activity A2.3.2 - Algorithms Exchange Format. `https://github.com/smaslan/TWM/tree/master/doc/A232AlgorithmExchangeFormat.docx`.

[4] M. Šíra and S. Mašláň. Uncertainty analysis of non-coherent sampling phase meter with four parameter sine wave fitting by means of monte carlo. In *29th Conference on Precision Electromagnetic Measurements (CPEM 2014)*, pages 334–335, Aug 2014.

4 TWM-HCRMS - Half Cycle RMS algorithm

Algorithm for calculation of the so called half cycle RMS values or sliding window RMS values of a single phase waveform. It calculates RMS value of signal in length of one period with window step defined by the method of calculation. That is, according to the IEC 61000-3-40: (i) Class A - half-cycle step; (ii) Class S - "sliding window" step (20 windows per period for this implementation). Examples of the calculated values for the modes A and S are shown in fig. 8.

The algorithm is designed so it can handle non-coherent sampling and also it is capable to compensate slow frequency drifts. It uses PSFE and resampling technique to ensure coherent sampling internally. The used can enter signal frequency manually if coherent sampling was ensured by the HW.

In general, the algorithm will work better with higher sampling rates. At least 100 samples should be recorded per period of the fundamental component (= sampling rate 5 kSa/s for 50 Hz networks). The higher is better, because the RMS algorithm will better suppress the harmonic and interharmonic content.

The algorithm is for single-ended input only and it is equipped with fast uncertainty estimator.

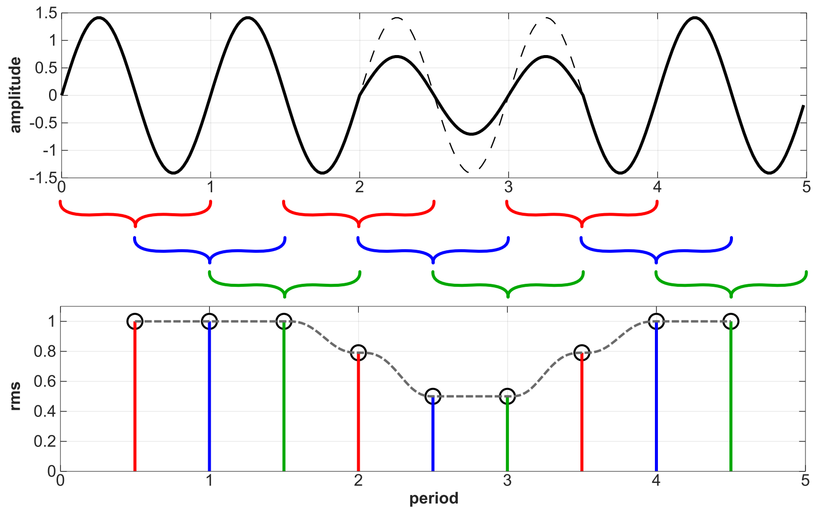


Figure 8: Example of Half Cycle RMS calculation for a "dip" event. The circles in RMS plot show values calculated according IEC 61000-3-40 "class A", the dashed line shows result of sliding window mode for "class S".

4.1 TWM wrapper parameters

The input quantities supported by the algorithm are shown in the table 14. Algorithm returns output quantities shown in the table 15. Calculation setup supported by the algorithm is shown in table 16.

Table 14: List of input quantities to the TWM-HCRMS wrapper.
Details on the correction quantities can be found in [1].

Name	Default	Unc.	Description
mode	"A"	N/A	Mode of calculation: "A" for class A or "S" for class S.
nom_f	N/A	N/A	Optional user defined frequency of the fundamental frequency. The algorithm will identify the fundamental frequency by itself when it is not assigned.
y	N/A	No	Input sample data vector.
Ts fs t	N/A N/A N/A	No No No	Sampling period or sampling rate or sample time vector. Note the wrapper always calculates in equidistant mode, so *t* is used just to calculate *Ts*.
adc_lsb adc_nrng adc_bits	N/A 1000 40	No No No	Either absolute ADC resolution *lsb* or nominal range value *adc_nrng* (e.g.: 5 V for 10 Vpp range) and *adc_bits* bit resolution of ADC.

Table 14: List of input quantities to the TWM-HCRMS wrapper.
Details on the correction quantities can be found in [1].

Name	**Default**	**Unc.**	**Description**
adc_offset	0	Yes	Digitizer input offset voltage.
adc_gain adc_gain_f adc_gain_a	1 [] []	Yes No No	Digitizer gain correction 2D table (multiplier).
adc_phi adc_phi_f adc_phi_a	0 [] []	Yes No No	Digitizer phase correction 2D table (additive).
adc_freq	0	Yes	Digitizer timebase error correction: $f\_tb' = f\_tb \cdot (1 + adc\_freq.v)$ The effect on the estimated frequency is opposite: $f\_est' = f\_est/(1 + adc\_freq.v)$
adc_jitter	0	No	Digitizer sampling period jitter [s].
adc_aper	0	No	ADC aperture value [s].
adc_aper_corr	0	No	ADC aperture error correction enable: $A' = A \cdot pi \cdot adc\_aper \cdot f\_est/\sin(pi \cdot adc\_aper \cdot f\_est)$ $phi' = phi + pi \cdot adc\_aper \cdot f\_est$
time_stamp	0	Yes	Relative timestamp of the first sample y.
adc_sfdr adc_sfdr_f adc_sfdr_a	180 [] []	No No No	Digitizer SFDR 2D table.
adc_Yin_Cp adc_Yin_Gp adc_Yin_f	1e-15 1e-15 []	Yes Yes No	Digitizer input admittance 1D table.
tr_type	“”	No	Transducer type string (“rvd” or “shunt”).
tr_gain tr_gain_f tr_gain_a	1 [] []	Yes No No	Transducer gain correction 2D table (multiplicative).
tr_phi tr_phi_f tr_phi_a	0 [] []	Yes No No	Transducer phase correction 2D table (additive).
tr_Zlo_Rp tr_Zlo_Cp tr_Zlo_f	1e3 1e-15 []	Yes Yes No	RVD transducer low-side impedance 1D table. Note this is related to loading correction and it has effect only for RVD transducer and will work only if adc_Yin is defined as well.
tr_Zca_Rs tr_Zca_Ls tr_Zca_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Transducer high side terminal series impedance 1D table.
tr_Zcal_Rs tr_Zcal_Ls tr_Zcal_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Transducer low side terminal series impedance 1D table.
tr_Yca_Cp tr_Yca_D tr_Yca_f	1e-15 1e-12 []	Yes Yes No	Loading corrections: Transducer output terminals shunting impedance.
tr_Zcam tr_Zcam_f	1e-12 []	Yes No	Loading corrections: Transducer output terminals mutual inductance 1D table.
Zcb_Rs Zcb_Ls Zcb_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Cable series impedance 1D table.
Ycb_Rs Ycb_Ls Ycb_f	1e-15 1e-12 []	Yes Yes No	Loading corrections: Cable series impedance 1D table.

Table 15: List of output quantities of the TWM-HCRMS wrapper.

Name	Uncertainty	Description
t	Yes	Time vector of the calculated samples [s].
env	Yes	Calculated half-cycle RMS values $env(t)$.
f0	Yes	Average detected fundamental frequency.

Table 16: List of "calcset" options supported by the TWM-HCRMS wrapper.

Name	Description
calcset.unc	Uncertainty calculation mode. Supported: "none" or "guf" for uncertainty estimator.
calcset.loc	Level of confidence [-].
calcset.verbose	Verbose level.
calcset.dbg_plots	Non-zero value to enable plotting of debugging/signal analysis plots.

4.2 Algorithm description

The overview of the wrapper TWM-HCRMS structure is shown in the diagram in fig. 9. It starts with signal scaling and corrections. First step is digitizer timebase correction. Follows removal of the digitizer DC offset. Next, the signal y is split into DC and AC components. Next, the wrapper calls PSFE to estimate fundamental frequency $f0\_est$ of the signal y unless used defined f_nom in algorithm parameters. The frequency $f0\_est$ is used to obtain and apply the gain, phase, aperture error corrections and transducer corrections to DC and AC components separately. Note the AC corrections are applied in time domain and applies only for the $f0\_est$ frequency. No frequency dependent corrections were implemented as the required accuracy of the algorithm is not critical for the PQ events detection. The scaled DC and AC components are merged back to the single time domain signal y, which is ready for the processing.

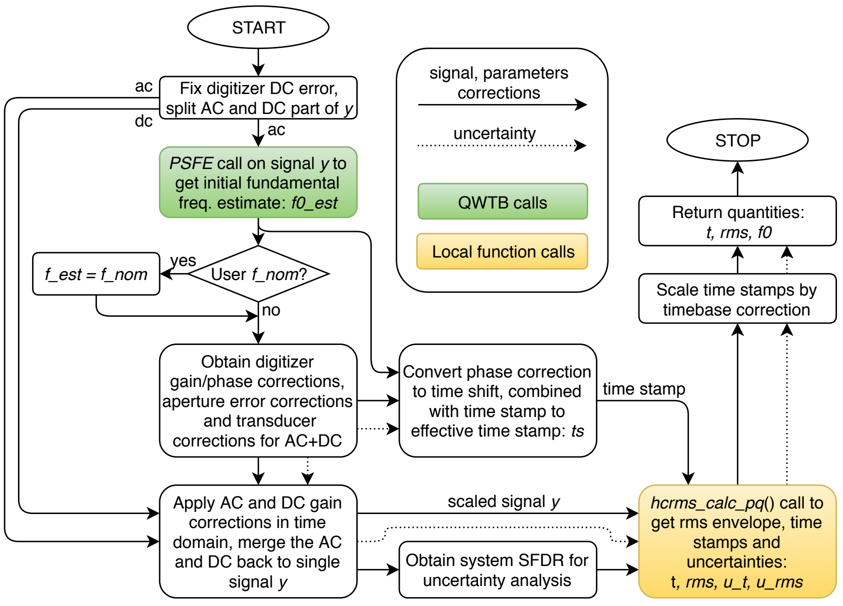

START
Fix digitizer DC error, split AC and DC part of y
ac
dc
ac
PSFE call on signal y to get initial fundamental freq. estimate: f0_est
User f_nom?
yes
f_est = f_nom
no
Obtain digitizer gain/phase corrections, aperture error corrections and transducer corrections for AC+DC
Convert phase correction to time shift, combined with time stamp to effective time stamp: ts
time stamp
Apply AC and DC gain corrections in time domain, merge the AC and DC back to single signal y
scaled signal y
Obtain system SFDR for uncertainty analysis
hcrms_calc_pq() call to get rms envelope, time stamps and uncertainties: t, rms, u_t, u_rms
Scale time stamps by timebase correction
Return quantities: t, rms, f0
STOP
signal, parameters corrections
uncertainty
QWTB calls
Local function calls

Figure 9: Overview of the TWM-HCRMS wrapper for evaluation of the RMS envelope in time.

The processing itself is performed by function "hcrms_calc_pq()" which is shown in the fig. 10. The first step of the algorithm is detection of the fundamental frequency and resampling to coherent sampling if user did not defined f_nom parameter. The algorithm uses PSFE algorithm called 200 times using a sliding window of size $N/200$ with step $N/200$. So the development of the fundamental frequency

$f0(t)$ in time is found (see example in fig. 11, top-left). The time development $f0(t)$ is filtered and the outliers caused by the PQ events are removed based on the simple heuristic algorithm. The removed portions usually happens on the edges of the "dip"-like events. The missing parts are replaced by the interpolation, so the frequency $f0(t)$ is known in full range of the processing time t. The $f0(t)$ is used to calculate resampling coefficients to achieve pseudo-coherent sampling in the full duration of the signal. The resampling by the dynamic frequency ratio is performed using ordinary spline interpolation, which seems to produce the least harmonic distortion of the resampled signal yx. The samples count per period of the resampled signal is chosen to be divisible by factor 2 for class A (so each period can be split to half) or by 20 for class to S (the algorithm calculates only 20 sliding windows per period).

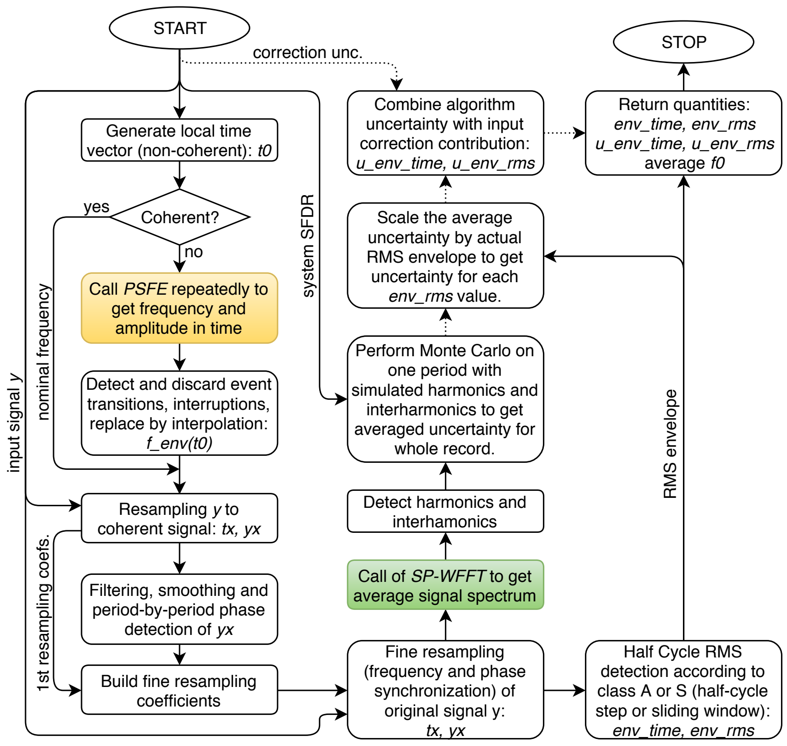

START
correction unc.
STOP
Generate local time vector (non-coherent): t0
Combine algorithm uncertainty with input correction contribution: u_env_time, u_env_rms
Return quantities: env_time, env_rms u_env_time, u_env_rms average f0
yes
Coherent?
no
system SFDR
Scale the average uncertainty by actual RMS envelope to get uncertainty for each env_rms value.
Call PSFE repeatedly to get frequency and amplitude in time
nominal frequency
input signal y
Detect and discard event transitions, interruptions, replace by interpolation: f_env(t0)
Perform Monte Carlo on one period with simulated harmonics and interharmonics to get averaged uncertainty for whole record.
RMS envelope
Resampling y to coherent signal: tx, yx
Detect harmonics and interharmonics
1st resampling coefs.
Filtering, smoothing and period-by-period phase detection of yx
Call of SP-WFFT to get average signal spectrum
Build fine resampling coefficients
Fine resampling (frequency and phase synchronization) of original signal y: tx, yx
Half Cycle RMS detection according to class A or S (half-cycle step or sliding window): env_time, env_rms

Figure 10: Detailed structure of the main half-cycle RMS calculator and uncertainty evaluator.

The next step is period-by-period phase detection and synchronization of yx. This is almost useless when there are at least 100 samples per period, however the first resampling is not absolutely precise, so this additional step improves the coherent sampling of each period. The wrapper first filters the resampled yx by a very narrow passband filter to yxf, which removes the harmonics. Next, the yxf is split per periods and send to FFT, which calculates phase error of each period $phi\_p(p)$ (see example in fig. 11, top-right). Heuristic algorithm discards the parts of $phi\_p(p)$ affected by the PQ events same as for the first resampling step and the missing parts are replaced by the interpolation and it also upsamples the phase value for each time sample to $phi\_p(t)$. The $phi\_p(t)$ is finally used to fine tune the first resampling coefficients and the resampling is performed again on the original data y to get synchronised signal yx. Example of the phase detection after the resampling is shown in fig. 11, bottom-left.

Follows the main RMS calculation algorithm which calculates RMS value with step 1/2-period (class A) or 1/20-period (class S) by ordinary non-windowed discrete RMS method:

$$rms(p) = \sqrt{\frac{1}{N1T} \cdot \sum_{k=1}^{N} yx(k + p \cdot N1T)^2}, \tag{9}$$

where k is sample index, p is window offset in periods and $N1T$ is length of the period in samples.

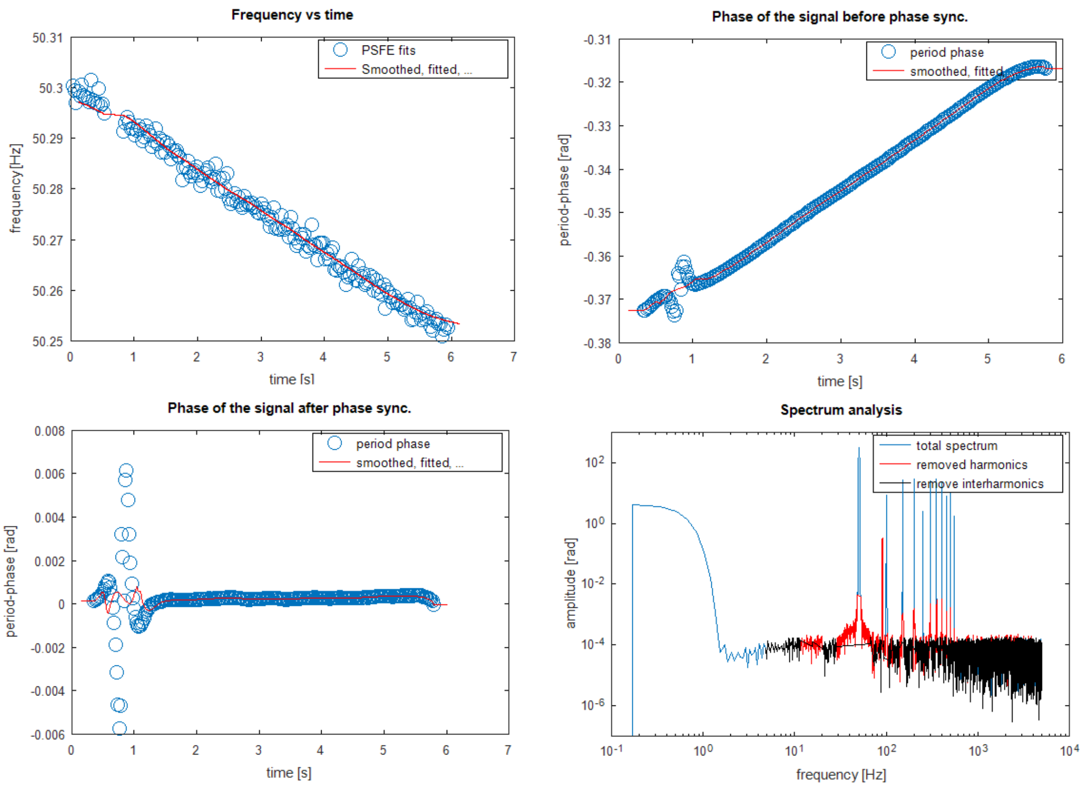


Figure 11: Debug plots showing the intermediated stages of TWM-HCRMS signal processing.

4.3 Uncertainty calculation

The algorithm is relatively straightforward and not excessively slow, so the calculation of uncertainty is performed on runtime when the "guf" option in "calcset" is selected. The core of the calculation is based on the spectrum analysis of the resampled signal yx. The spectrum is used to identify dominant harmonic and interharmonic components (example is shown in fig. 11, bottom-right). Follows a small Monte Carlo loop which simulates the effect of harmonics and interharmonics on the RMS value of one period of the signal. It also simulates uncertain quality of the resampling, i.e. non-perfect coherency. The loop uses just 200 cycles and it is fast as it is performed on one period only.

The previous steps were performed on the conditions identified from the averaged spectrum of the whole record. Therefore, if the record contained PQ events, such as "dip" or "swell", the estimated uncertainty will be inaccurate. So the calculated uncertainty is scaled proportionally for each period by the actual RMS level. This simple method based on average spectrum analysis showed good agreement with repeated calculation for each single period, so it was decided to use it as a solution of choice. The only disadvantage is the uncertainty around event edges is larger than it may be, however RMS method does not allow exact localisation of the events, so tries to fix this often lead to the underestimation.

The final step is combining the uncertainty coming from the corrections with the uncertainty of the algorithm and assigning it to the particular RMS samples. Note the uncertainty estimator also assigns the uncertainty to the timestamps of each RMS value, however these are almost irrelevant as the technique for detecting the PQ events introduces uncertainty orders of magnitude higher.

References

[1] Stanislav Mašláň. Activity A2.3.2 - Algorithms Exchange Format. `https://github.com/smaslan/TWM/tree/master/doc/A232AlgorithmExchangeFormat.docx`.

5 TWM-InDiSwell - Interruption, Dip, Swell event detector

This algorithm detects power quality events “dip”, “swell” and “interruption” for a single phase systems according to the IEC 61000-3-40, “class A” (half-cycle step) or “class S” (sliding window). It returns relative event time, duration and its residual RMS value in percents relative to the entered nominal level *nom_rms*. Note the result provided for the classes A and S should be identical as long as the event is synchronised with the nominal frequency. However that is rarely the case of real life situations, so the selection must be made depending on the prescription for the given PQ meter test or PQ event calibrator.

The algorithm internally uses RMS envelope detector TWM-HCRMS, so the accuracy of the detection depends on its properties. In general, the algorithm will work better with higher sampling rates. At least 100 samples should be recorded per period of the fundamental component (= sampling rate 5 kSa/s for 50 Hz networks). The higher is better, because the RMS algorithm will better suppress the harmonic and interharmonic content.

The algorithm is for single-ended input only and it is equipped with fast uncertainty estimator.

Example of the detected event as plotted by the algorithm is shown in the fig. 12.

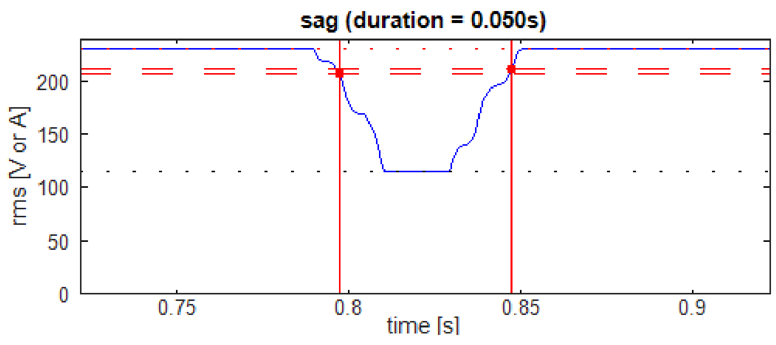


Figure 12: Example of the “dip” event evaluated according to the class S of IEC 61000-3-40.

5.1 TWM wrapper parameters

The input quantities supported by the algorithm are shown in the table 17. Algorithm returns output quantities shown in the table 18. Calculation setup supported by the algorithm is shown in table 19.

Table 17: List of input quantities to the TWM-InDiSwell wrapper.
Details on the correction quantities can be found in [1].

Name	Default	Unc.	Description
mode	“A”	N/A	Mode of calculation: “A” for class A or “S” for class S.
nom_f	N/A	N/A	Optional user defined frequency of the fundamental frequency. The algorithm will identify the fundamental frequency by itself when it is not assigned.
nom_rms	230	N/A	Optional user defined nominal RMS value of the network. The event thresholds will be related to this value.
sag_thresh	90	N/A	Optional threshold value for “sag” (resp. “dip”) event evaluation. It is percent of nominal level *nom_rms*.
swell_thresh	110	N/A	Optional threshold value for “swell” event evaluation. It is percent of nominal level *nom_rms*.
int_thresh	10	N/A	Optional threshold value for “interruption” event evaluation. It is percent of nominal level *nom_rms*.
hyst	2	N/A	Detection hysteresis in percent of nominal level *nom_rms*.
plot	0	N/A	Enables plotting of the detected events. One plot per event type will be generated with detection levels, RSM envelope and markers of the event.
y	N/A	No	Input sample data vector.

Table 17: List of input quantities to the TWM-InDiSwell wrapper.
Details on the correction quantities can be found in [1].

Name	**Default**	**Unc.**	**Description**
Ts fs t	N/A N/A N/A	No No No	Sampling period or sampling rate or sample time vector. Note the wrapper always calculates in equidistant mode, so t is used just to calculate Ts.
adc_lsb adc_nrng adc_bits	N/A 1000 40	No No No	Either absolute ADC resolution lsb or nominal range value adc_nrng (e.g.: 5 V for 10 Vpp range) and adc_bits bit resolution of ADC.
adc_offset	0	Yes	Digitizer input offset voltage.
adc_gain adc_gain_f adc_gain_a	1 [] []	Yes No No	Digitizer gain correction 2D table (multiplier).
adc_phi adc_phi_f adc_phi_a	0 [] []	Yes No No	Digitizer phase correction 2D table (additive).
adc_freq	0	Yes	Digitizer timebase error correction: $f\_tb' = f\_tb \cdot (1 + adc\_freq.v)$ The effect on the estimated frequency is opposite: $f\_est' = f\_est/(1 + adc\_freq.v)$
adc_jitter	0	No	Digitizer sampling period jitter [s].
adc_aper	0	No	ADC aperture value [s].
adc_aper_corr	0	No	ADC aperture error correction enable: $A' = A \cdot pi \cdot adc\_aper \cdot f\_est/\sin(pi \cdot adc\_aper \cdot f\_est)$ $phi' = phi + pi \cdot adc\_aper \cdot f\_est$
time_stamp	0	Yes	Relative timestamp of the first sample y.
adc_sfdr adc_sfdr_f adc_sfdr_a	180 [] []	No No No	Digitizer SFDR 2D table.
adc_Yin_Cp adc_Yin_Gp adc_Yin_f	1e-15 1e-15 []	Yes Yes No	Digitizer input admittance 1D table.
tr_type	“”	No	Transducer type string (“rvd” or “shunt”).
tr_gain tr_gain_f tr_gain_a	1 [] []	Yes No No	Transducer gain correction 2D table (multiplicative).
tr_phi tr_phi_f tr_phi_a	0 [] []	Yes No No	Transducer phase correction 2D table (additive).
tr_Zlo_Rp tr_Zlo_Cp tr_Zlo_f	1e3 1e-15 []	Yes Yes No	RVD transducer low-side impedance 1D table. Note this is related to loading correction and it has effect only for RVD transducer and will work only if adc_Yin is defined as well.
tr_Zca_Rs tr_Zca_Ls tr_Zca_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Transducer high side terminal series impedance 1D table.
tr_Zcal_Rs tr_Zcal_Ls tr_Zcal_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Transducer low side terminal series impedance 1D table.
tr_Yca_Cp tr_Yca_D tr_Yca_f	1e-15 1e-12 []	Yes Yes No	Loading corrections: Transducer output terminals shunting impedance.
tr_Zcam tr_Zcam_f	1e-12 []	Yes No	Loading corrections: Transducer output terminals mutual inductance 1D table.

Table 17: List of input quantities to the TWM-InDiSwell wrapper.
Details on the correction quantities can be found in [1].

Name	Default	Unc.	Description
Zcb_Rs Zcb_Ls Zcb_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Cable series impedance 1D table.
Ycb_Rs Ycb_Ls Ycb_f	1e-15 1e-12 []	Yes Yes No	Loading corrections: Cable series impedance 1D table.

Table 18: List of output quantities of the TWM-InDiSwell wrapper.

Name	Uncertainty	Description
t	Yes	Time vector of the calculated samples [s].
env	Yes	Calculated half-cycle RMS values $env(t)$.
f0	No	Average detected fundamental frequency.
sag_start	Yes	Sag (dip) event start relative time stamp [s].
sag_dur	Yes	Sag (dip) event duration [s].
sag_res	Yes	Sag (dip) event residual RMS level [%].
swell_start	Yes	Swell event start relative time stamp [s].
swell_dur	Yes	Swell event duration [s].
swell_res	Yes	Swell event residual RMS level [%].
int_start	Yes	Interruption event start relative time stamp [s].
int_dur	Yes	Interruption event duration [s].
int_res	Yes	Interruption event residual RMS level [%].

Table 19: List of "calcset" options supported by the TWM-InDiSwell wrapper.

Name	Description
calcset.unc	Uncertainty calculation mode. Supported: "none" or "guf" for uncertainty estimator.
calcset.loc	Level of confidence [-].
calcset.verbose	Verbose level.
calcset.dbg_plots	Non-zero value to enable plotting of debugging/signal analysis plots of the TWM-HCRMS, which is used internally by the TWM-InDiSwell.

5.2 Algorithm description

The algorithm TWM-InDiSWell internally uses algorithm TWM-HCRMS to calculate RMS envelope of the signal. Therefore all input signal conditioning of y is performed in the TWM-HCRMS. The TWM-HCRMS output RMS values and corresponding time stamps are used to detect events according to the preset thresholds. The detection method follow the IEC 61000-3-40 standard. The start of event is assigned to the first RMS sample whose value exceeds the threshold. End of event is assigned to the first sample whose RMS value returned below the $(threshold - hysteresis)$, which prevents multiple events detection around the threshold. Flowchart of the algorithm is shown in fig. 13.

5.3 Uncertainty calculation

The main component of uncertainty is the output of TWM-HCRMS. However, due to the principle of detection, especially for class A, the uncertainty of the event start can never be lower than half of the period, as that is the resolution of the RMS detector. The duration uncertainty cannot be lower than

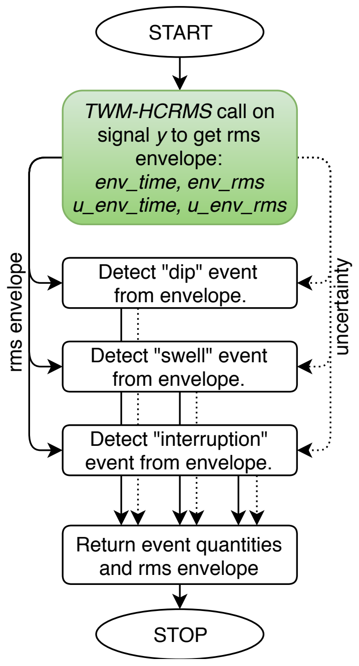

START
TWM-HCRMS call on signal *y* to get rms envelope:
env_time, env_rms
u_env_time, u_env_rms
rms envelope
uncertainty
Detect "dip" event from envelope.
Detect "swell" event from envelope.
Detect "interruption" event from envelope.
Return event quantities and rms envelope
STOP

Figure 13: Flowchart of the algorithm wrapper TWM-InDiSwell. Note the green cells are calls to another QWTB/TWM wrappers.

one full period, as the same resolution applies to the end of the event. The resolution in the class S mode is higher, however the uncertainty remains the same as that is the requirement of the IEC standard.

References

[1] Stanislav Mašláň. Activity A2.3.2 - Algorithms Exchange Format. `https://github.com/smaslan/TWM/tree/master/doc/A232AlgorithmExchangeFormat.docx`.

6 TWM-THDWFFT - THD from Windowed FFT

This algorithm is designed for calculation of the harmonics and Total Harmonic Distortion (THD) of the non-coherently sampled signal. It uses windowed FFT to detect the harmonic amplitudes, which limits the achievable accuracy of the harmonics detection due to the window scalloping effect. However, the algorithm was initially designed for THD calculation of low-distortion signals, where the accuracy was not critical. The relative expanded uncertainty of of the harmonics is at least 0.015 % (or 0.005 % after highly experimental correction method). On the other hand, the algorithm was designed to compensate the spectral leakage of the noise to the harmonics near noise level, so it offers decent accuracy for the very low distortions.

The algorithm supports direct processing of a multiple records which are used to produce averaged spectrum before the main calculation. This possibility should be preferred instead of repeated call of the algorithm for each record. The input to the algorithm is only single-ended.

The algorithm returns: (i) Full spectrum; (ii) Identified harmonics; (iii) THD coefficients according various definitions; (iv) RMS noise estimate; (v) THD+Noise estimate.

Example of the algorithm output is shown in the fig. 14.

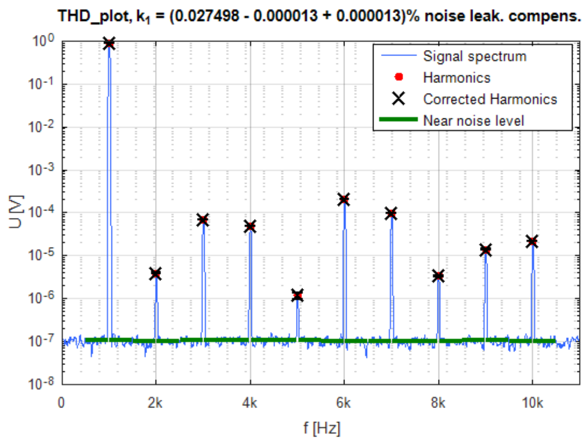


Figure 14: Example of the TWM-THDWFFT algorithm output.

6.1 TWM wrapper parameters

The input quantities supported by the algorithm are shown in the table 20. Algorithm returns output quantities shown in the table 21. Calculation setup supported by the algorithm is shown in table 22.

Table 20: List of input quantities to the TWM-THDWFFT wrapper. Details on the correction quantities can be found in [4].

Name	**Default**	**Unc.**	**Description**
f0	N/A	N/A	Optional user defined frequency of fundamental component. Do not assigned to enable auto detection.
f0_mode	“PSFE”	N/A	Optional selection of the fundamental frequency auto detection mode.
scallop_fix	0	N/A	Non-zero value to enable experimental window scalloping error correction. It will try to use known scalloping error of the window at given frequency to correct the error, however it will work only for stable signals when the fundamental frequency detection is accurate.
H	10	N/A	Optional limit of maximum harmonics count to analyse (including fundamental). Note the high values will significantly increase calculation time!

Table 20: List of input quantities to the TWM-THDWFFT wrapper. Details on the correction quantities can be found in [4].

Name	**Default**	**Unc.**	**Description**
band	inf	N/A	Optional bandwidth limit which can reduce the harmonics count to analyse. This also affects the bandwidth of the noise calculation.
plot	0	N/A	Non-zero value, "on", "true" or "enabled" string enables plotting of the detected harmonics.
y	N/A	No	Input matrix of the samples. One column per record (the algorithm can directly calculate average of multiple records).
Ts fs t	N/A N/A N/A	No No No	Sampling period or sampling rate or sample time vector. Note the wrapper always calculates in equidistant mode, so t is used just to calculate Ts.
adc_lsb adc_nrng adc_bits	N/A 1000 40	No No No	Either absolute ADC resolution lsb or nominal range value adc_nrng (e.g.: 5 V for 10 Vpp range) and adc_bits bit resolution of ADC.
adc_jitter	0	No	Digitizer sampling period jitter [s].
adc_aper_corr	0	No	ADC aperture error correction enable: $A' = A \cdot pi \cdot adc\_aper \cdot f\_est / \sin(pi \cdot adc\_aper \cdot f\_est)$ $phi' = phi + pi \cdot adc\_aper \cdot f\_est$
adc_aper	0	No	ADC aperture value [s].
adc_gain adc_gain_f adc_gain_a	1 [] []	Yes No No	Digitizer gain correction 2D table (multiplier).
adc_freq	0	Yes	Digitizer timebase error correction: $f\_tb' = f\_tb \cdot (1 + adc\_freq.v)$ The effect on the estimated frequency is opposite: $f\_est' = f\_est/(1 + adc\_freq.v)$
adc_sfdr adc_sfdr_f adc_sfdr_a	180 [] []	No No No	Digitizer SFDR 2D table.
adc_Yin_Cp adc_Yin_Gp adc_Yin_f	1e-15 1e-15 []	Yes Yes No	Digitizer input admittance 1D table.
tr_type	""	No	Transducer type string ("rvd" or "shunt").
tr_gain tr_gain_f tr_gain_a	1 [] []	Yes No No	Transducer gain correction 2D table (multiplicative).
tr_Zlo_Rp tr_Zlo_Cp tr_Zlo_f	1e3 1e-15 []	Yes Yes No	RVD transducer low-side impedance 1D table. Note this is related to loading correction and it has effect only for RVD transducer and will work only if adc_Yin is defined as well.
tr_Zca_Rs tr_Zca_Ls tr_Zca_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Transducer high side terminal series impedance 1D table.
tr_Zcal_Rs tr_Zcal_Ls tr_Zcal_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Transducer low side terminal series impedance 1D table.
tr_Yca_Cp tr_Yca_D tr_Yca_f	1e-15 1e-12 []	Yes Yes No	Loading corrections: Transducer output terminals shunting impedance.
tr_Zcam tr_Zcam_f	1e-12 []	Yes No	Loading corrections: Transducer output terminals mutual inductance 1D table.
Zcb_Rs Zcb_Ls Zcb_f	1e-9 1e-12 []	Yes Yes No	Loading corrections: Cable series impedance 1D table.

Table 20: List of input quantities to the TWM-THDWFFT wrapper. Details on the correction quantities can be found in [4].

Name	Default	Unc.	Description
Ycb_Rs	1e-15	Yes	Loading corrections: Cable series impedance 1D table.
Ycb_Ls	1e-12	Yes	
Ycb_f	[]	No	

Table 21: List of output quantities of the TWM-THDWFFT wrapper. Note the uncertainty "No" means the algorithm may return some uncertainty but it should be ignored because it is either incomplete or not validated.

Name	Uncertainty	Description
H	No	Harmonics count analysed.
noise_bw	No	Bandwidth used for the noise estimation [Hz].
thd	Yes	Total Harmonic Distortion referenced to the fundamental.
thd2	Yes	Total Harmonic Distortion referenced to the RMS value.
thdn	No	Total Harmonic Distortion + Noise referenced to the fundamental.
thdn2	No	Total Harmonic Distortion + Noise referenced to the RMS value.
noise	No	RMS noise estimate.
h	Yes	Amplitudes of the harmonics.
f	No	Frequencies of the harmonics h.
spec_a	No	Full spectrum from the windowed FFT.
spec_f	No	Frequencies of the spectrum components $spec\_a$.
thd_raw	No	thd without noise spectrum leakage correction.
thd2_raw	No	$thd2$ without noise spectrum leakage correction.

Table 22: List of "calcset" options supported by the TWM-THDWFFT wrapper.

Name	Description
calcset.unc	Uncertainty calculation mode. Supported: "none" or "guf" for uncertainty estimator. Note the algorithm is internally made in such a way it always calculates the uncertainty, so this option should have no effect in current version.
calcset.loc	Level of confidence [-].
calcset.verbose	Verbose level.

6.2 Algorithm description and uncertainty evaluation

The whole algorithm is extended and improved version of the THD analyser presented in [5]. The overview of the algorithm wrapper structure and internal functions is shown in the fig. 15. The wrapper start by a call to the top level function "thd_wfft()", which performs entire calculation and uncertainty estimation. Next, the wrapper may optionally plot graph showing the identified harmonics and near spectrum. Note the wrapper reduces the asymmetric uncertainty limits to symmetric as the TWM was not designed for such a case. This has no effect when the level of harmonics is at least twice the noise level. It will expand the uncertainty only for very small harmonic levels near noise level.

The "thd_wfft()" itself internally does just two steps: (i) Calculating spectra of input records and estimates their fundamental frequency (function "thd_proc_waves()"); (ii) Initiates main evaluation function "thd_eval_thd()".

The function "thd_proc_waves()" first detects fundamental frequency of each record in y. It contains several modes of detection. The simplest is zero crossing, however it is very unreliable. Another options if FPNLSF [6], which may fail when initial estimate from zero cross detector is poor. Last and best option (default) is PSFE [3], which is capable to identify the fundamental frequency with good accuracy

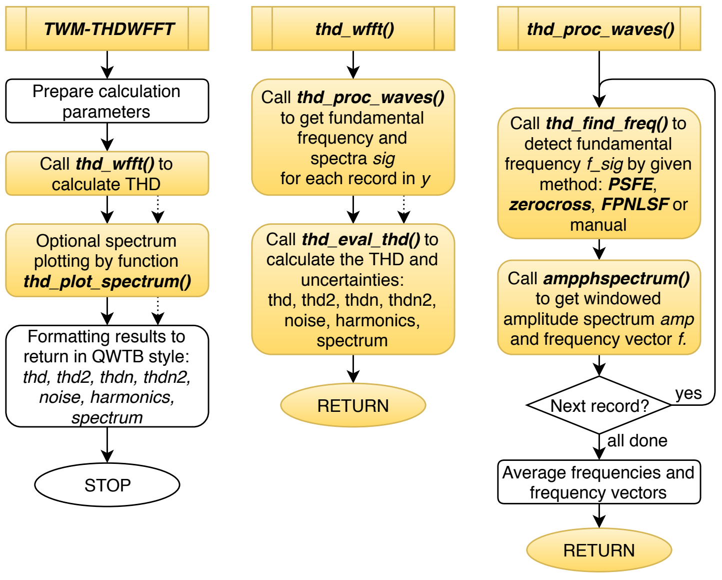

TWM-THDWFFT
Prepare calculation parameters
Call thd_wfft() to calculate THD
Optional spectrum plotting by function thd_plot_spectrum()
Formatting results to return in QWTB style: thd, thd2, thdn, thdn2, noise, harmonics, spectrum
STOP
thd_wfft()
Call thd_proc_waves() to get fundamental frequency and spectra sig for each record in y
Call thd_eval_thd() to calculate the THD and uncertainties: thd, thd2, thdn, thdn2, noise, harmonics, spectrum
RETURN
thd_proc_waves()
Call thd_find_freq() to detect fundamental frequency f_sig by given method: PSFE, zerocross, FPNLSF or manual
Call ampphspectrum() to get windowed amplitude spectrum amp and frequency vector f.
Next record?
yes
all done
Average frequencies and frequency vectors
RETURN

Figure 15: Flow chart of the algorithm wrapper TWM-THDWFFT. Note the rounded gold cells are calls to other local functions which are shown in another diagram or mentioned in the text.

even with strong harmonic content. User may also override the auto detection by manual entry of the fundamental frequency. The next step is calculation of amplitude spectrum for each record y using a windowed FFT. The widest, flattest window with highest suppression was chosen for the goal - Flattop HFT248D based on the [1]. This window offer side lobes suppression by 248 dB and scalloping error only 0.0104 % for range ±0.5 DFT bin.

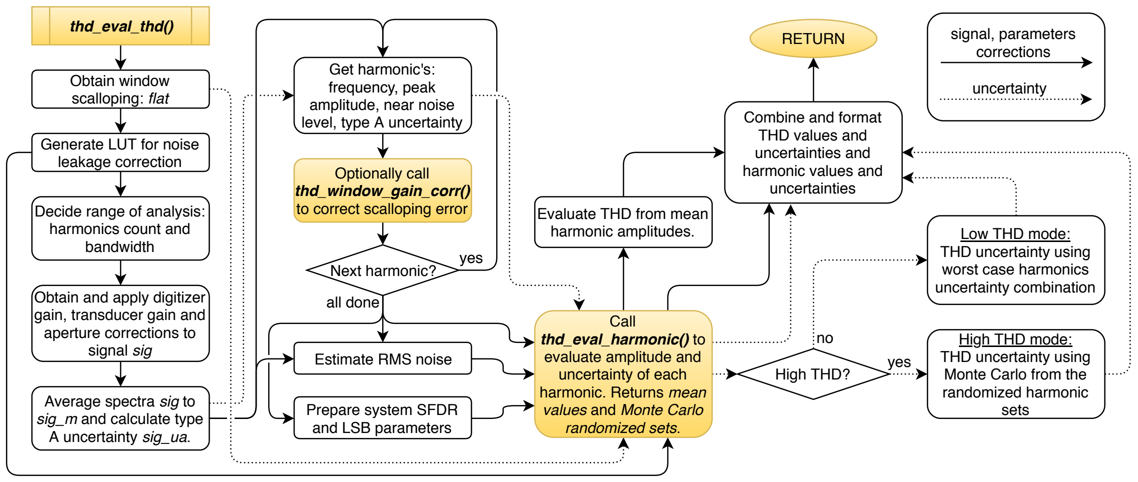

thd_eval_thd()
Obtain window scalloping: flat
Generate LUT for noise leakage correction
Decide range of analysis: harmonics count and bandwidth
Obtain and apply digitizer gain, transducer gain and aperture corrections to signal sig
Average spectra sig to sig_m and calculate type A uncertainty sig_ua.
Get harmonic's: frequency, peak amplitude, near noise level, type A uncertainty
Optionally call thd_window_gain_corr() to correct scalloping error
Next harmonic?
yes
all done
Estimate RMS noise
Prepare system SFDR and LSB parameters
Call thd_eval_harmonic() to evaluate amplitude and uncertainty of each harmonic. Returns mean values and Monte Carlo randomized sets.
Evaluate THD from mean harmonic amplitudes.
Combine and format THD values and uncertainties and harmonic values and uncertainties
RETURN
High THD?
no
yes
Low THD mode: THD uncertainty using worst case harmonics uncertainty combination
High THD mode: THD uncertainty using Monte Carlo from the randomized harmonic sets
signal, parameters corrections
uncertainty

Figure 16: Flow chart of the main algorithm function "thd_eval_thd()" for the TWM-THDWFFT algorithm.

The internal structure of the evaluation function "thd_eval_thd()" is shown in fig. 16. The function does following steps:

1. Obtaining parameters of the window function Flattop HFT248D used for the processing.
2. Generation of lookup table (LUT), which will be used for the numeric solver that compensates spectrum leakage of the noise to the harmonic DFT bin (details below).
3. Decision of how many harmonics to analyse based on the user limits (H and $bandwidth$).

4. Application of all gain corrections to scale the spectra from "thd_proc_waves()" to actual levels.
5. Averaging of the spectra and type A uncertainty calculation.
6. Detection of harmonics. The algorithm picks the harmonics from the average spectrum one by one. It searches the highest DFT bin in preset range for each estimated harmonics frequency. It also extracts the nearby noise level which is needed for compensation of the noise spectral leakage.
7. The parameters required for the uncertainty evaluation of each harmonics are obtained (system SFDR and LSB).
8. Evaluation of the harmonic values and uncertainties using function "thd_eval_harmonic()" (see below). This returns mean harmonic levels and calculated uncertainties and also randomized harmonic levels, because it internally uses Monte Carlo.
9. Calculation of the THD coefficients from the mean harmonic amplitudes according to various definition and calculation of their uncertainties using one of the methods (see below).

The evaluation of the THD coefficients in the step 9) is performed according to the several definitions. The most common is so called "fundamental referenced" THD:

$$thd = \frac{\sqrt{U_2^2 + U_3^2 + \cdots + U_M}}{U_1}, \tag{10}$$

where U_x is mean harmonic voltage and x is harmonic index and M is harmonics count. The next is RMS value referenced mode, which uses total RMS of the signal in the denominator:

$$thd2 = \frac{\sqrt{U_2^2 + U_3^2 + \cdots + U_M^2}}{\sqrt{U_1^2 + U_2^2 + U_3^2 + \cdots + U_M^2}}. \tag{11}$$

The results should be very close for low distortion signals. Next result is combined fundamental referenced THD and noise THD+N:

$$thdn = \frac{\sqrt{U_2^2 + U_3^2 + \cdots + U_M + U_{\text{noise}}^2}}{U_1}, \tag{12}$$

where the U_{noise} is RMS noise in specified bandwidth (parameter *band*). Last definition is RMS referenced THD+N:

$$thdn2 = \frac{\sqrt{U_2^2 + U_3^2 + \cdots + U_M^2 + U_{\text{noise}}^2}}{\sqrt{U_1^2 + U_2^2 + U_3^2 + \cdots + U_M^2 + U_{\text{noise}}^2}}. \tag{13}$$

The algorithm also returns the same four coefficient without the noise leakage correction, however those are just informative.

The uncertainty evaluation for the THD coefficients uses heuristic approach. The THD coefficients are calculated from the mean values from step 8) ignoring the uncertainty and its distribution. The uncertainty calculation method depends on the "is_high" obtained in step 8), which is set when the weighted average of the harmonic amplitudes is significantly above noise. So two case occur:

1. *is_high* = *true*: The distribution of the uncertainty of the harmonics is near Gaussian so the randomized amplitudes from step 8) are passed to the THD formulas above and the THD is evaluated using Monte Carlo and function "scovint()" (follows GUM guide [2]).
2. *is_high* = *false*: The distribution of the uncertainty of the harmonics is very asymmetric, so the Monte Carlo would lead to large bias in the mean value of THD. Therefore the THD uncertainty is evaluated using the worst case combination of the harmonic uncertainties from step 8):

$$[thd_{\text{MAX}}, thd_{\text{MIN}}] = \left[\frac{\sqrt{\sum\limits_{m=2}^{M} U_{m_{\text{MAX}}}^2}}{U_{1_{\text{MIN}}}}, \frac{\sqrt{\sum\limits_{m=2}^{M} U_{m_{\text{MIN}}}^2}}{U_{1_{\text{MAX}}}} \right] \tag{14}$$

where:

$$U_{m_{\mathrm{MAX}}} = U_m + U_+(U_m), \tag{15}$$
$$U_{m_{\mathrm{MIN}}} = U_m - U_-(U_m). \tag{16}$$

The reported asymmetric uncertainties were calculated according to:

$$[U_+(thd), U_-(thd)] = [thd_{\mathrm{MAX}} - thd, thd - k_{\mathrm{MIN}}]\,. \tag{17}$$

The evaluation of the uncertainty of each harmonic is performed by the function "thd_eval_harmonic()" shown in fig. 17. This is simple heuristic function that calculates uncertainty distribution of each harmonic component depending on how close it is to the noise level. This is necessary, because the distribution for harmonics well above the noise level will be near Gaussian, whereas the possible value of the harmonic near noise level may be anywhere in the noise or slightly above. The result of this approach is very asymmetric distribution that cannot be processed using GUF method. Therefore the calculation is performed by Monte Carlo with 10000 cycles (defined as fixed option in the TWM-THDWFFT wrapped). The performance is acceptable as long as no more than 50 harmonics are analysed. The resulting randomised set of harmonic amplitudes is returned in full for further processing. However, the function also calculates the uncertainty limits for each harmonic for given level of confidence by function "scovint()" (implemented according to [2]).

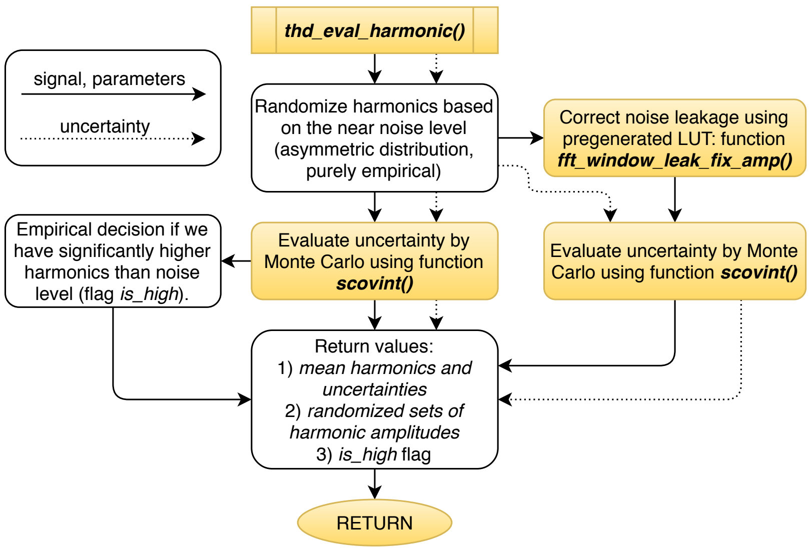

thd_eval_harmonic()
signal, parameters
uncertainty
Randomize harmonics based on the near noise level (asymmetric distribution, purely empirical)
Correct noise leakage using pregenerated LUT: function fft_window_leak_fix_amp()
Empirical decision if we have significantly higher harmonics than noise level (flag is_high).
Evaluate uncertainty by Monte Carlo using function scovint()
Evaluate uncertainty by Monte Carlo using function scovint()
Return values:
1) mean harmonics and uncertainties
2) randomized sets of harmonic amplitudes
3) is_high flag
RETURN

Figure 17: Flow chart of the function "thd_eval_harmonic()" of the TWM-THDWFFT algorithm.

The function "thd_eval_harmonic()" also repeats the same calculation once more with the mentioned noise leakage correction. The problem related to wide window functions such as Flattop HFT248D is the not only the harmonic power leaks to the more DFT bins, but also the noise energy near the harmonic leaks to the harmonic DFT bin. This effect is normally not considered, when the narrower windows are used and when the harmonic is several times larger than the noise. However, this algorithm uses very wide window Flattop HFT248D and it was designed to operate near noise level. The apparent gain of the detected harmonic can be obtained by the following procedure:

1. generation of sine wave $x(t)$ with amplitude U_m,
2. addition of gaussian noise with level U_{noise} to the $x(t)$,
3. windowing of the $x(t)$ by selected window function (Flattop HFT248D),
4. reading the amplitude U_{x} from amplitude spectrum of $X(f)$ of signal $x(t)$.

Alternatively the same result can be obtained by means of Monte Carlo method from equation:

$$U_\mathrm{x} = \frac{1}{I}\sum_{i=1}^{I}\left|U_m + U_\mathrm{noise}\sum_{k=1}^{K} W_k \cdot \mathrm{e}^{-j2\pi \mathrm{R}(i,k)}\right| \tag{18}$$

where K is number of coefficients of window function amplitude spectrum W_k and I is number of MC iterations (at least 10^4). The $R(i,k)$ is uniformly distributed random number generator from 0 to 1. The right sum term represents a vector sum of a noise vectors with random angle and amplitude weighted by window spectrum coefficients W_k. The resulting gain vs. noise to signal ratio is shown in fig. 18.

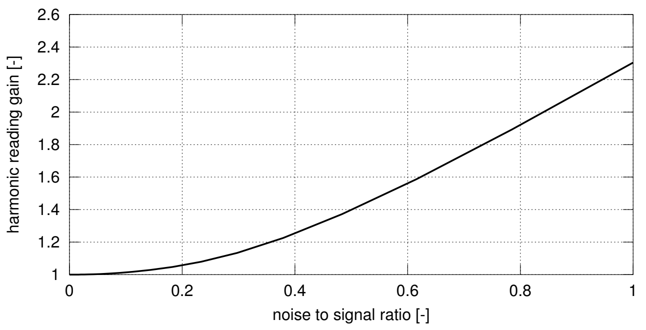


Figure 18: Error of the harmonics amplitude measurement using FFT with window Flattop HFT248D. Note the "noise" means amplitude of the noise in surrounding DFT bins, not RMS noise.

.

The direct inverse evaluation from the detected to actual harmonic level is nto possible, so the algorithm uses iterative function based on the precalculated LUT with the gain error (the dependence in fig. 18). The correction itself is performed by the function "fft_window_leak_fix_amp()", which takes the harmonic level, noise level detected around (assuming the noise is the same for all related DFT bins). Effect of this correction is shown in fig. 19.

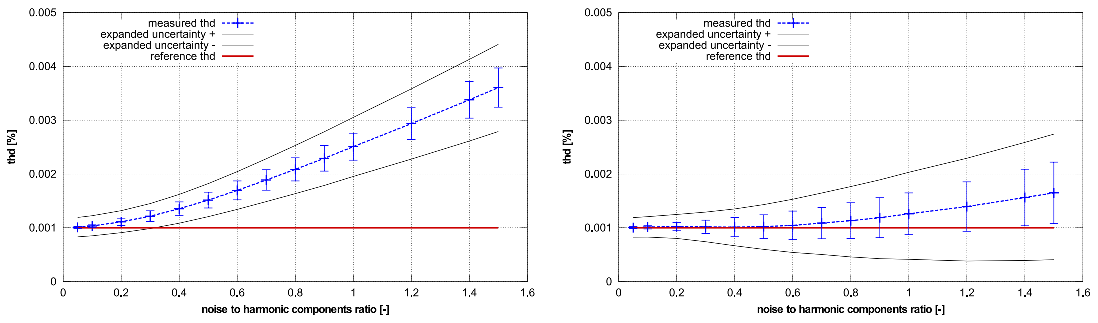


Figure 19: Deviation of THDWFFT algorithm from simulated THD level 10 ppm for various noise to higher harmonic ratios. The simulated waveform has 10 harmonic components with amplitudes $U_\mathrm{m} = \{0.9, 3\cdot10^{-6}, 3\cdot10^{-6}, \ldots\}$ V. Left graph shows results without noise spectral leakage correction, right graph shows the same dependence with corrected values. The error bars show the standard deviation of a repeated simulations.

References

[1] Gerhard Heinzel, A. Rűdiger, and R. Schilling. Spectrum and spectral density estimation by the discrete fourier transform (dft), including a comprehensive list of window functions and some new flat-top windows. Technical report, Max-Planck-Institut für Gravitationsphysik (Albert-Einstein-Institut) Teilinstitut Hannover, Feb 2005.

[2] JCGM. *Evaluation of measurement data - Supplement 1 to the "Guide to the expression of uncertainty in measurement" - Propagation of distributions using a Monte Carlo method.* Bureau International des Poids et Measures.

[3] Rado Lapuh. Estimating the fundamental component of harmonically distorted signals from noncoherently sampled data. *IEEE Transactions on Instrumentation and Measurement*, 64(6):1419–1424, June 2015.

[4] Stanislav Mašláň. Activity A2.3.2 - Algorithms Exchange Format. `https://github.com/smaslan/TWM/tree/master/doc/A232AlgorithmExchangeFormat.docx`.

[5] Stanislav Mašláň and Martin Šíra. Automated non-coherent sampling thd meter with spectrum analyser. In *Proceedings CPEM*, 2014.

[6] M. Šíra and S. Mašláň. Uncertainty analysis of non-coherent sampling phase meter with four parameter sine wave fitting by means of monte carlo. In *29th Conference on Precision Electromagnetic Measurements (CPEM 2014)*, pages 334–335, Aug 2014.

7 TWM-PWRTDI - Power by Time Domain Integration

7.1 TWM wrapper parameters

The input quantities supported by the algorithm are shown in the table 23. Algorithm returns output quantities shown in the table 24. Calculation setup supported by the algorithm is shown in table 25.

Table 23: List of input quantities to the TWM-PWRTDI wrapper.
Details on the correction quantities can be found in [1].

Name	Default	Unc.	Description
ac_coupling	0	N/A	Enables virtual AC coupling of the wattmeter. This option will cause the DC value will be ignored.
u u_lo	N/A N/A	No No	Input voltage sample data vector and complementary low-side input data vector *i_lo* (for differential mode only).
i i_lo	N/A N/A	No No	Input current sample data vector and complementary low-side input data vector *i_lo* (for differential mode only).
Ts fs t	N/A N/A N/A	No No No	Sampling period or sampling rate or sample time vector. Note the wrapper always calculates in equidistant mode, so t is used just to calculate Ts.
time_shift	0	Yes	Timeshift between voltage channel u and current channel i.
u_time_shift_lo i_time_shift_lo	0 0	Yes Yes	Time shift between high-side channel u low-side channel *u_lo* (or i and *i_lo* for current).
u_lsb u_adc_nrng u_adc_bits u_lo_lsb u_lo_adc_nrng u_lo_adc_bits i_lsb i_adc_nrng i_adc_bits i_lo_lsb i_lo_adc_nrng i_lo_adc_bits	N/A 1000 40 N/A 1000 40 N/A 1000 40 N/A 1000 40	No No No No No No No No No No No No	Either absolute ADC resolution *lsb* or nominal range value *adc_nrng* (e.g.: 5 V for 10 Vpp range) and *adc_bits* bit resolution of ADC.
u_adc_offset u_lo_adc_offset i_adc_offset i_lo_adc_offset	0 0 0 0	Yes Yes Yes Yes	Digitizer input offset voltage.
u_adc_gain u_adc_gain_f u_adc_gain_a u_lo_adc_gain u_lo_adc_gain_f u_lo_adc_gain_a i_adc_gain i_adc_gain_f i_adc_gain_a i_lo_adc_gain i_lo_adc_gain_f i_lo_adc_gain_a	1 [] [] 1 [] [] 1 [] [] 1 [] []	Yes No No Yes No No Yes No No Yes No No	Digitizer gain correction 2D table (multiplier).

Table 23: List of input quantities to the TWM-PWRTDI wrapper.
Details on the correction quantities can be found in [1].

Name	**Default**	**Unc.**	**Description**
u_adc_phi	0	Yes	Digitizer phase correction 2D table (additive).
u_adc_phi_f	[]	No	
u_adc_phi_a	[]	No	
u_lo_adc_phi	0	Yes	
u_lo_adc_phi_f	[]	No	
u_lo_adc_phi_a	[]	No	
i_adc_phi	0	Yes	
i_adc_phi_f	[]	No	
i_adc_phi_a	[]	No	
i_lo_adc_phi	0	Yes	
i_lo_adc_phi_f	[]	No	
i_lo_adc_phi_a	[]	No	
adc_freq	0	Yes	Digitizer timebase error correction: $f\_tb' = f\_tb \cdot (1 + adc\_freq.v)$ The effect on the estimated frequency is opposite: $f\_est' = f\_est/(1 + adc\_freq.v)$
u_adc_jitter	0	No	Digitizer sampling period jitter [s].
u_lo_adc_jitter	0	No	
i_adc_jitter	0	No	
i_lo_adc_jitter	0	No	
u_adc_aper	0	No	ADC aperture value [s].
u_lo_adc_aper	0	No	
i_adc_aper	0	No	
i_lo_adc_aper	0	No	
u_adc_aper_corr	0	No	ADC aperture error correction enable:
u_lo_adc_aper	0	No	$A' = A \cdot pi \cdot adc\_aper \cdot f\_est/\sin(pi \cdot adc\_aper \cdot f\_est)$
i_adc_aper_corr	0	No	$phi' = phi + pi \cdot adc\_aper \cdot f\_est$
i_lo_adc_aper	0	No	
u_adc_sfdr	180	No	Digitizer SFDR 2D table.
u_adc_sfdr_f	[]	No	
u_adc_sfdr_a	[]	No	
u_lo_adc_sfdr	180	No	
u_lo_adc_sfdr_f	[]	No	
u_lo_adc_sfdr_a	[]	No	
i_adc_sfdr	180	No	
i_adc_sfdr_f	[]	No	
i_adc_sfdr_a	[]	No	
i_lo_adc_sfdr	180	No	
i_lo_adc_sfdr_f	[]	No	
i_lo_adc_sfdr_a	[]	No	
u_adc_Yin_Cp	1e-15	Yes	Digitizer input admittance 1D table.
u_adc_Yin_Gp	1e-15	Yes	
u_adc_Yin_f	[]	No	
u_lo_adc_Yin_Cp	1e-15	Yes	
u_lo_adc_Yin_Gp	1e-15	Yes	
u_lo_adc_Yin_f	[]	No	
i_adc_Yin_Cp	1e-15	Yes	
i_adc_Yin_Gp	1e-15	Yes	
i_adc_Yin_f	[]	No	
i_lo_adc_Yin_Cp	1e-15	Yes	
i_lo_adc_Yin_Gp	1e-15	Yes	
i_lo_adc_Yin_f	[]	No	

Table 23: List of input quantities to the TWM-PWRTDI wrapper.
Details on the correction quantities can be found in [1].

Name	**Default**	**Unc.**	**Description**
u_tr_type	“”	No	Transducer type string (“rvd” or “shunt”).
i_tr_type			
u_tr_gain	1	Yes	Transducer gain correction 2D table (multiplicative).
u_tr_gain_f	[]	No	
u_tr_gain_a	[]	No	
i_tr_gain	1	Yes	
i_tr_gain_f	[]	No	
i_tr_gain_a	[]	No	
u_tr_phi	0	Yes	Transducer phase correction 2D table (additive).
u_tr_phi_f	[]	No	
u_tr_phi_a	[]	No	
i_tr_phi	0	Yes	
i_tr_phi_f	[]	No	
i_tr_phi_a	[]	No	
u_tr_Zlo_Rp	1e3	Yes	RVD transducer low-side impedance 1D table. Note this is related to loading correction and it has effect only for RVD transducer and will work only if *adc_Yin* is defined as well.
u_tr_Zlo_Cp	1e-15	Yes	
u_tr_Zlo_f	[]	No	
i_tr_Zlo_Rp	1e3	Yes	
i_tr_Zlo_Cp	1e-15	Yes	
i_tr_Zlo_f	[]	No	
u_tr_Zca_Rs	1e-9	Yes	Loading corrections: Transducer high side terminal series impedance 1D table.
u_tr_Zca_Ls	1e-12	Yes	
u_tr_Zca_f	[]	No	
i_tr_Zca_Rs	1e-9	Yes	
i_tr_Zca_Ls	1e-12	Yes	
i_tr_Zca_f	[]	No	
u_tr_Zcal_Rs	1e-9	Yes	Loading corrections: Transducer low side terminal series impedance 1D table.
u_tr_Zcal_Ls	1e-12	Yes	
u_tr_Zcal_f	[]	No	
i_tr_Zcal_Rs	1e-9	Yes	
i_tr_Zcal_Ls	1e-12	Yes	
i_tr_Zcal_f	[]	No	
u_tr_Yca_Cp	1e-15	Yes	Loading corrections: Transducer output terminals shunting impedance.
u_tr_Yca_D	1e-12	Yes	
u_tr_Yca_f	[]	No	
i_tr_Yca_Cp	1e-15	Yes	
i_tr_Yca_D	1e-12	Yes	
i_tr_Yca_f	[]	No	
u_tr_Zcam	1e-12	Yes	Loading corrections: Transducer output terminals mutual inductance 1D table.
u_tr_Zcam_f	[]	No	
i_tr_Zcam	1e-12	Yes	
i_tr_Zcam_f	[]	No	
u_Zcb_Rs	1e-9	Yes	Loading corrections: Cable series impedance 1D table.
u_Zcb_Ls	1e-12	Yes	
u_Zcb_f	[]	No	
i_Zcb_Rs	1e-9	Yes	
i_Zcb_Ls	1e-12	Yes	
i_Zcb_f	[]	No	

Table 23: List of input quantities to the TWM-PWRTDI wrapper.
Details on the correction quantities can be found in [1].

Name	Default	Unc.	Description
u_Ycb_Rs	1e-15	Yes	Loading corrections: Cable series impedance 1D table.
u_Ycb_Ls	1e-12	Yes	
u_Ycb_f	[]	No	
i_Ycb_Rs	1e-15	Yes	
i_Ycb_Ls	1e-12	Yes	
i_Ycb_f	[]	No	

Table 24: List of output quantities of the TWM-FPNLSF wrapper.

Name	Uncertainty	Description
U	Yes	RMS voltage [V].
I	Yes	RMS current [A].
P	Yes	Active power [W].
S	Yes	Apparent power [VA].
Q	Yes	Reactive power [VAr].
phi_ef	Yes	Effective phase angle arccos PF [rad].
Udc	Yes	DC voltage component [V].
Idc	Yes	DC current component [A].
Pdc	Yes	DC power component [W].
spec_U	No	Voltage channel spectrum [V].
spec_I	No	Current channel spectrum [A].
spec_S	No	Apparanet power spectrum [VA].
spec_f	No	Frequency vector of *spec_U*, *spec_I* and *spec_S*.

Table 25: List of "calcset" options supported by the TWM-PWRTDI wrapper.

Name	Description
calcset.unc	Uncertainty calculation mode. Supported: "none", "guf" for uncertainty estimator, "mcm" for Monte Carlo.
calcset.loc	Level of confidence [-].
calcset.verbose	Verbose level.

7.2 Algorithm description

References

[1] Stanislav Mašláň. Activity A2.3.2 - Algorithms Exchange Format. `https://github.com/smaslan/TWM/tree/master/doc/A232AlgorithmExchangeFormat.docx`.