A2.4.4 Description of algorithms

Version: 2018-08-23, V0.3, Stanislav Mašláň

Note this is just a first draft!

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.

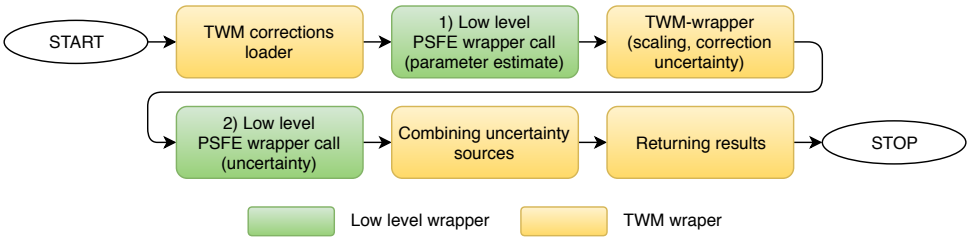

Figure 1: Overview of the TWM-PSFE algorithm wrapper.

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.

1.1 TWM wrapper parameters

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_sfdr	180	No	Transducer SFDR 2D table.
tr_sfdr_f	[]	No	
tr_sfdr_a	[]	No	
tr_Zlo_Rp	1e3	Yes	RVD transducer low-side impedance 1D table. Note this is
tr_Zlo_Cp	1e-15	Yes	related to loading correction and it has effect only for RVD
tr_Zlo_f	[]	No	transducer and will work only if adc_Yin is defined as well.
tr_Zca_Rs	1e-9	Yes	Loading corrections: Transducer high side terminal series
tr_Zca_Ls	1e-12	Yes	impedance 1D table.
tr_Zca_f	[]	No	
tr_Zcal_Rs	1e-9	Yes	Loading corrections: Transducer low side terminal series
tr_Zcal_Ls	1e-12	Yes	impedance 1D table.
tr_Zcal_f	[]	No	
tr_Yca_Cp	1e-15	Yes	Loading corrections: Transducer output terminals shunting
tr_Yca_D	1e-12	Yes	impedance.
tr_Yca_f	[]	No	
tr_Zcam	1e-12	Yes	Loading corrections: Transducer output terminals mutual
tr_Zcam_f	[]	No	inductance 1D table.

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

Name	Default	Unc.	Description
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 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.

1.2 PSFE algorithm description

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.

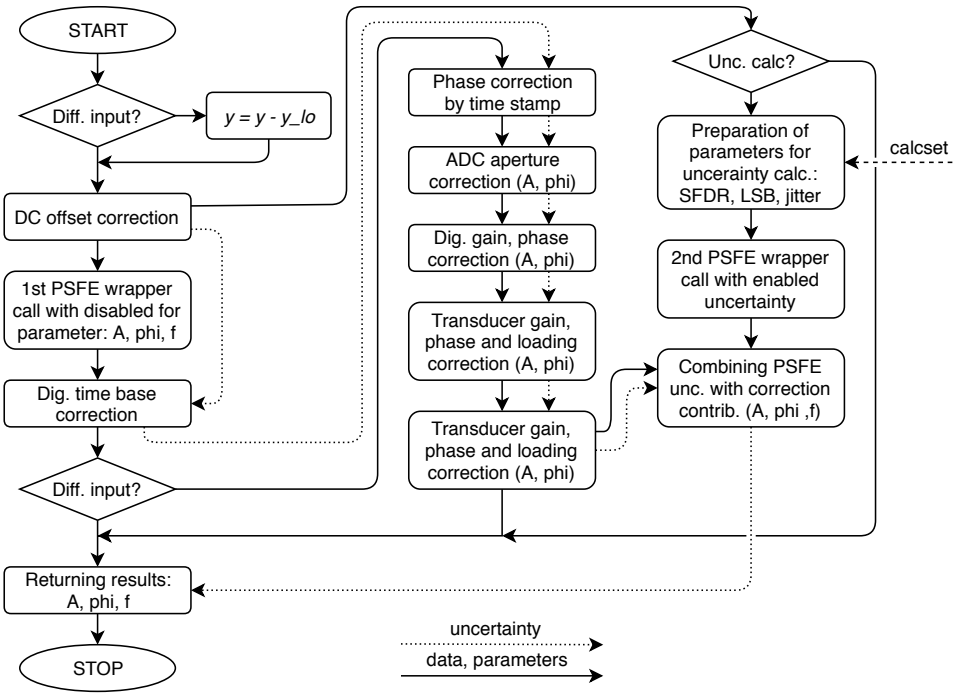

Figure 2: Detailed internal structure of the TWM-PSFE algorithm wrapper.

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_sfdr	180	No	Transducer SFDR 2D table.
tr_sfdr_f	[]	No	
tr_sfdr_a	[]	No	
tr_Zlo_Rp	1e3	Yes	RVD transducer low-side impedance 1D table. Note this is
tr_Zlo_Cp	1e-15	Yes	related to loading correction and it has effect only for RVD
tr_Zlo_f	[]	No	transducer and will work only if adc_Yin is defined as well.
tr_Zca_Rs	1e-9	Yes	Loading corrections: Transducer high side terminal series
tr_Zca_Ls	1e-12	Yes	impedance 1D table.
tr_Zca_f	[]	No	
tr_Zcal_Rs	1e-9	Yes	Loading corrections: Transducer low side terminal series
tr_Zcal_Ls	1e-12	Yes	impedance 1D table.
tr_Zcal_f	[]	No	
tr_Yca_Cp	1e-15	Yes	Loading corrections: Transducer output terminals shunting
tr_Yca_D	1e-12	Yes	impedance.
tr_Yca_f	[]	No	
tr_Zcam	1e-12	Yes	Loading corrections: Transducer output terminals mutual
tr_Zcam_f	[]	No	inductance 1D table.
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).

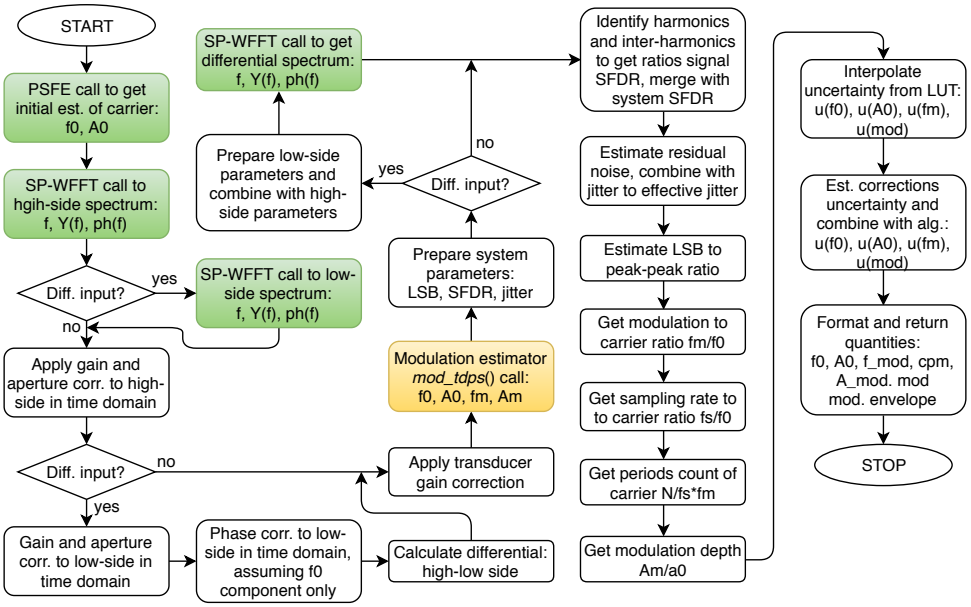

Figure 3: Overview of the TWM-MODTDPS algorithm wrapper. Green cells are calls to other QWTB wrappers. Gold cells are calls of the local functions which are described in the text.

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.

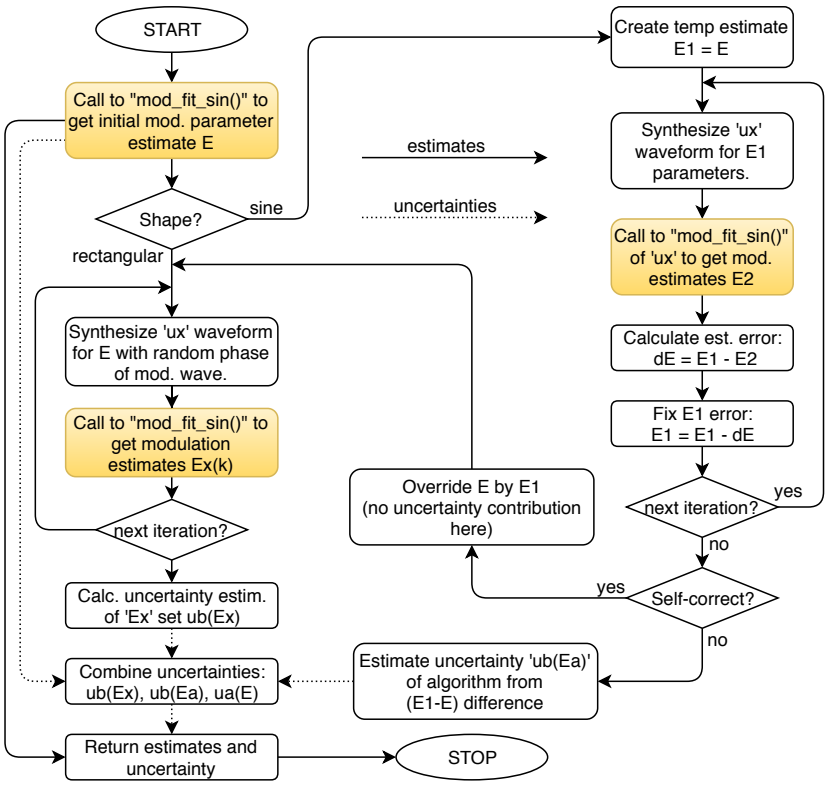

Figure 4: Overview of the "mod_tdps()" function of the TWM-MODTDPS algorithm wrapper. Gold cells are calls of local functions of the algorithm.

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:

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

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

$$\tag{3}$$

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

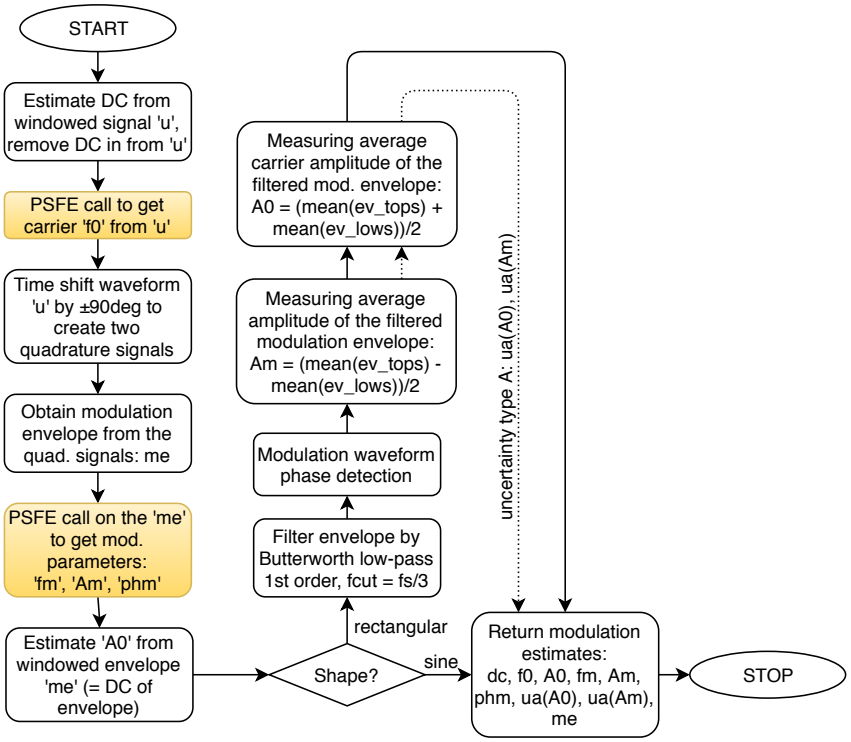

Figure 5: Overview of the "mod_fit_sin()" function of the TWM-MODTDPS algorithm wrapper. Gold cells are calls of local functions of the algorithm.

$A0$ 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 ±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	0	No	ADC aperture error correction enable:
lo_adc_aper	0		$A' = A \cdot pi \cdot adc\_aper \cdot f\_est / \sin(pi \cdot adc\_aper \cdot f\_est)$
			$phi' = phi + pi \cdot adc\_aper \cdot f\_est$
time_stamp	0	Yes	Relative timestamp of the first sample y.
time_shift_lo	0	Yes	Low-side channel time shift [s].
adc_sfdr	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_sfdr	180	No	Transducer SFDR 2D table.
tr_sfdr_f	[]	No	
tr_sfdr_a	[]	No	
tr_Zlo_Rp	1e3	Yes	RVD transducer low-side impedance 1D table. Note this is
tr_Zlo_Cp	1e-15	Yes	related to loading correction and it has effect only for RVD
tr_Zlo_f	[]	No	transducer and will work only if adc_Yin is defined as well.
tr_Zca_Rs	1e-9	Yes	Loading corrections: Transducer high side terminal series
tr_Zca_Ls	1e-12	Yes	impedance 1D table.
tr_Zca_f	[]	No	
tr_Zcal_Rs	1e-9	Yes	Loading corrections: Transducer low side terminal series
tr_Zcal_Ls	1e-12	Yes	impedance 1D table.
tr_Zcal_f	[]	No	
tr_Yca_Cp	1e-15	Yes	Loading corrections: Transducer output terminals shunting
tr_Yca_D	1e-12	Yes	impedance.
tr_Yca_f	[]	No	
tr_Zcam	1e-12	Yes	Loading corrections: Transducer output terminals mutual
tr_Zcam_f	[]	No	inductance 1D table.
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 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.

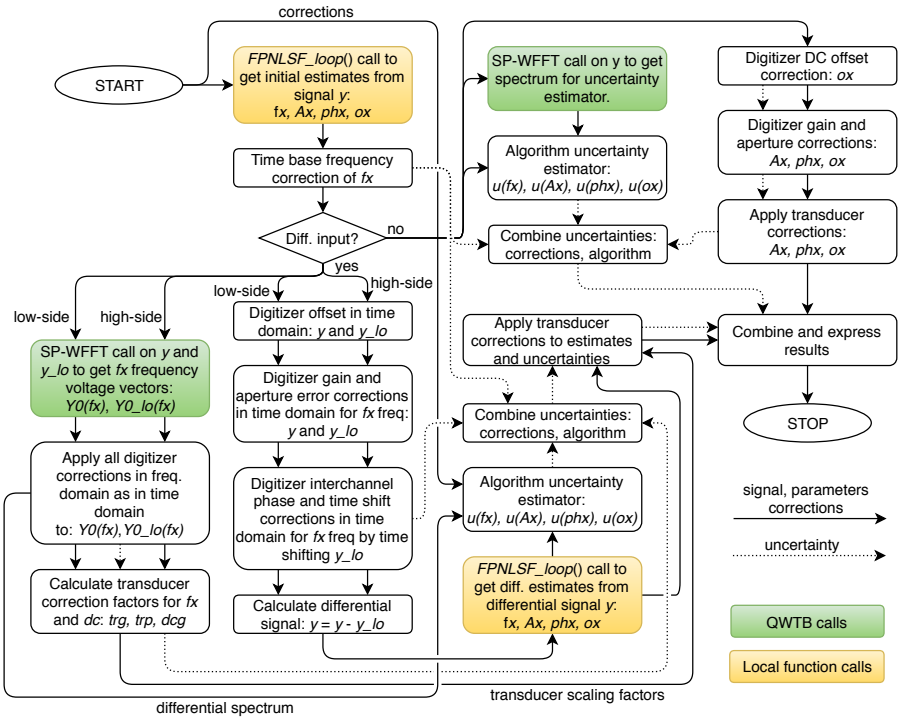

Figure 6: Overview of the TWM-FPNLSF algorithm wrapper.

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:

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

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.

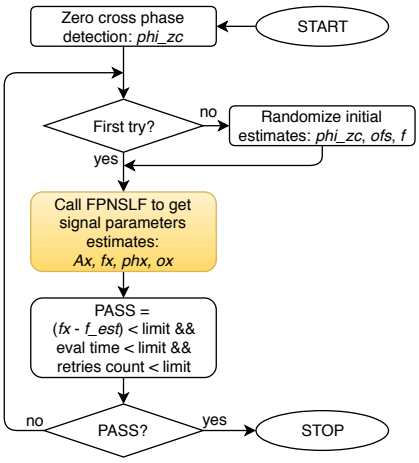

Figure 7: The structure of the "FPNLSF_loop()" function of the TWM-FPNLSF algorithm wrapper.

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 $\pm 5\,\%$ 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.

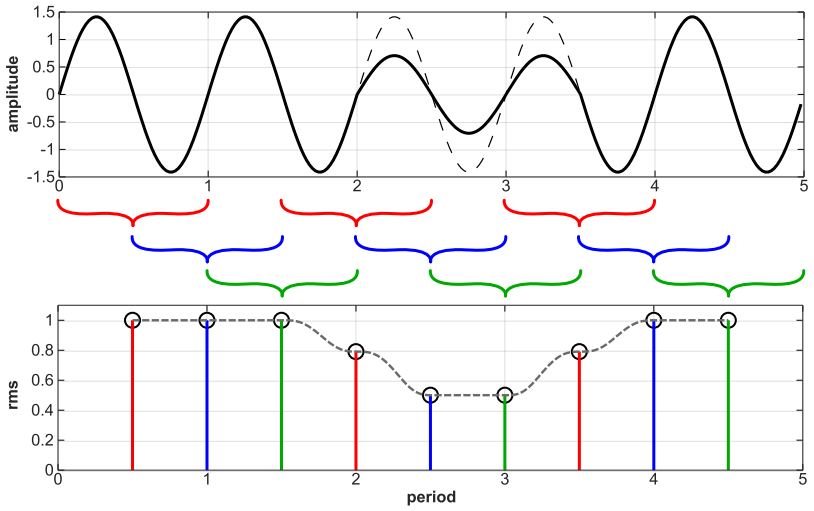

Figure 8: Example of Half Cycle RMS calculation for a "dip" event. The circles in RMS plot show values calculated according IEC 61000-3-40 "class A", the dashed line shows result of sliding window mode for "class S".

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	1	Yes	Digitizer gain correction 2D table (multiplier).
adc_gain_f	[]	No	
adc_gain_a	[]	No	
adc_phi	0	Yes	Digitizer phase correction 2D table (additive).
adc_phi_f	[]	No	
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: $A' = A \cdot pi \cdot adc\_aper \cdot f\_est/\sin(pi \cdot adc\_aper \cdot f\_est)$ $phi' = phi + pi \cdot adc\_aper \cdot f\_est$
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	
adc_Yin_Cp	1e-15	Yes	Digitizer input admittance 1D table.
adc_Yin_Gp	1e-15	Yes	
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_sfdr	180	No	Transducer SFDR 2D table.
tr_sfdr_f	[]	No	
tr_sfdr_a	[]	No	
tr_Zlo_Rp	1e3	Yes	RVD transducer low-side impedance 1D table. Note this is
tr_Zlo_Cp	1e-15	Yes	related to loading correction and it has effect only for RVD
tr_Zlo_f	[]	No	transducer and will work only if adc_Yin is defined as well.
tr_Zca_Rs	1e-9	Yes	Loading corrections: Transducer high side terminal series
tr_Zca_Ls	1e-12	Yes	impedance 1D table.
tr_Zca_f	[]	No	
tr_Zcal_Rs	1e-9	Yes	Loading corrections: Transducer low side terminal series
tr_Zcal_Ls	1e-12	Yes	impedance 1D table.
tr_Zcal_f	[]	No	
tr_Yca_Cp	1e-15	Yes	Loading corrections: Transducer output terminals shunting
tr_Yca_D	1e-12	Yes	impedance.
tr_Yca_f	[]	No	
tr_Zcam	1e-12	Yes	Loading corrections: Transducer output terminals mutual
tr_Zcam_f	[]	No	inductance 1D table.
Zcb_Rs	1e-9	Yes	Loading corrections: Cable series impedance 1D table.
Zcb_Ls	1e-12	Yes	
Zcb_f	[]	No	

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

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.

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).

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

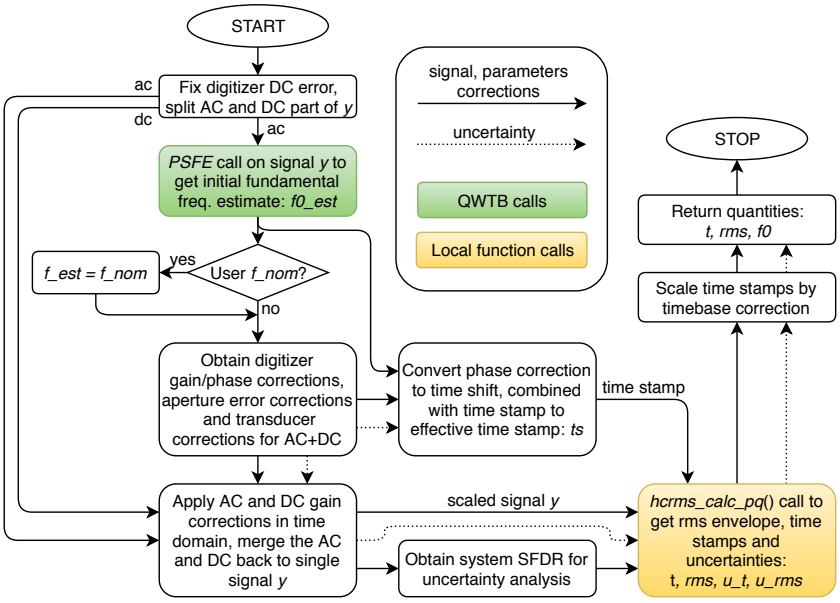

Figure 9: Overview of the TWM-HCRMS wrapper for evaluation of the RMS envelope in time.

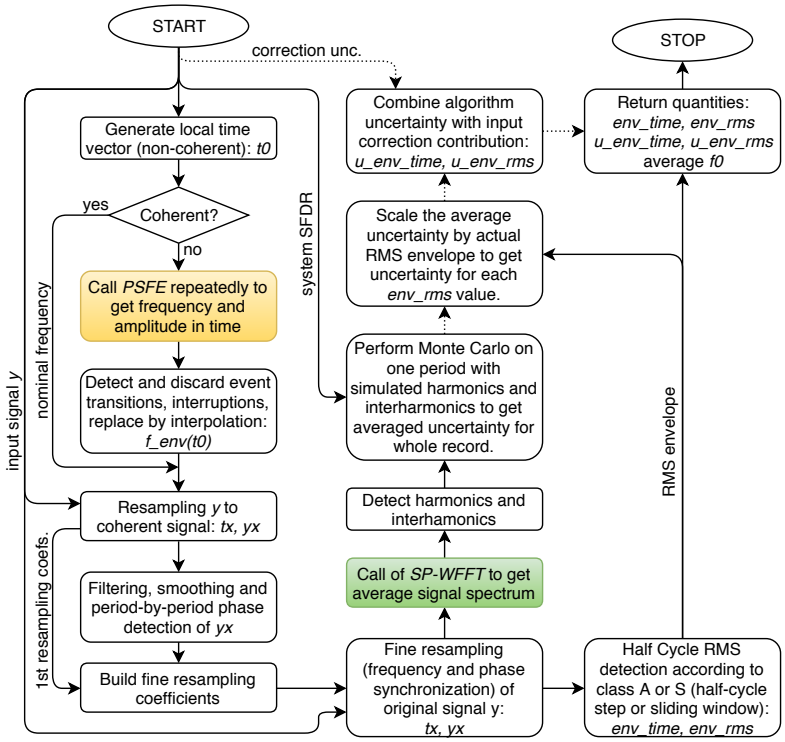

Figure 10: Detailed structure of the main half-cycle RMS calculator and uncertainty evaluator.

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.

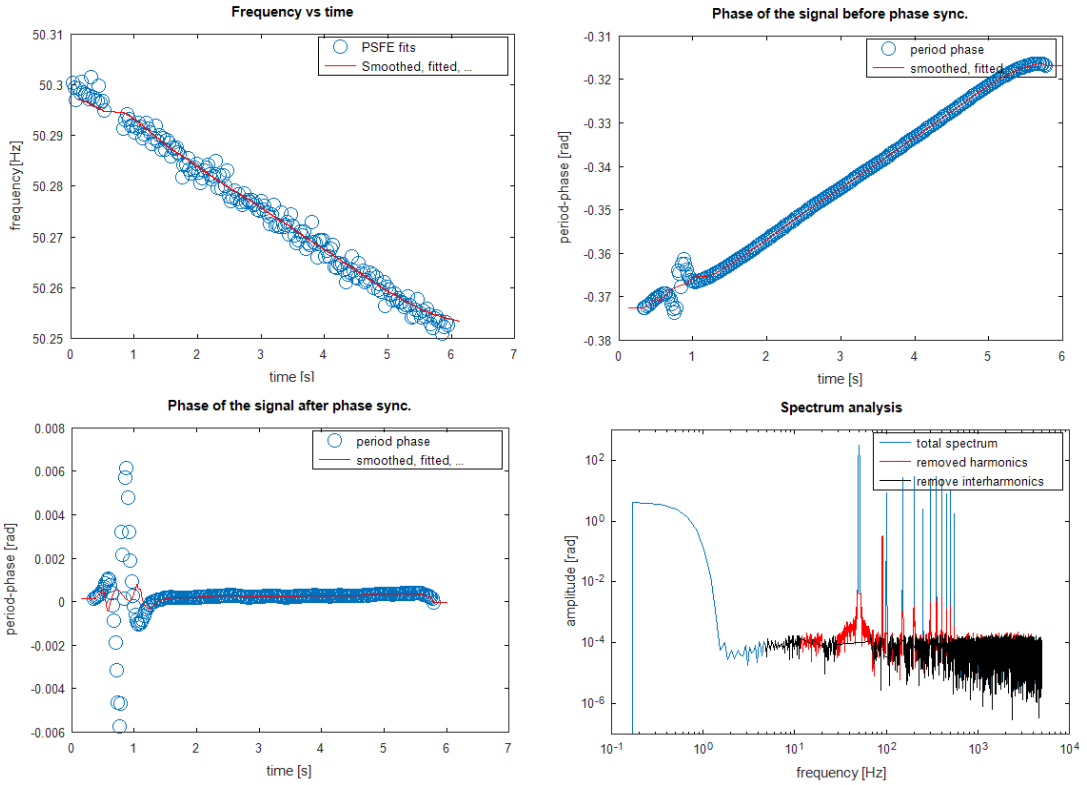

Figure 11: Debug plots showing the intermediated stages of TWM-HCRMS signal processing.

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.

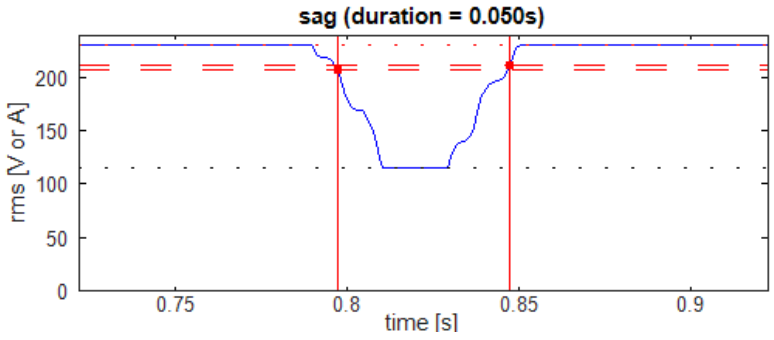

Figure 12: Example of the "dip" event evaluated according to the class S of IEC 61000-3-40.

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_sfdr tr_sfdr_f tr_sfdr_a	180 [] []	No No No	Transducer SFDR 2D table.
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.

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
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 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

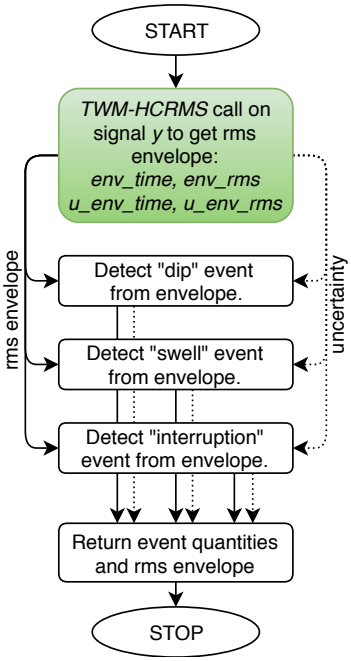

Figure 13: Flowchart of the algorithm wrapper TWM-InDiSwell. Note the green cells are calls to another QWTB/TWM wrappers.

period, as that is the resolution of the RMS detector. The duration uncertainty cannot be lower than 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.

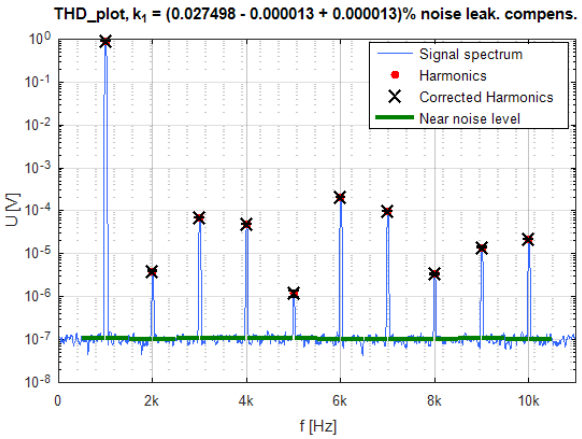

Figure 14: Example of the TWM-THDWFFT algorithm output.

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 assign 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_sfdr tr_sfdr_f tr_sfdr_a	180 [] []	No No No	Transducer SFDR 2D table.
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 20: List of input quantities to the TWM-THDWFFT wrapper. Details on the correction quantities can be found in [4].

Name	Default	Unc.	Description
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 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	*thd*2 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()".

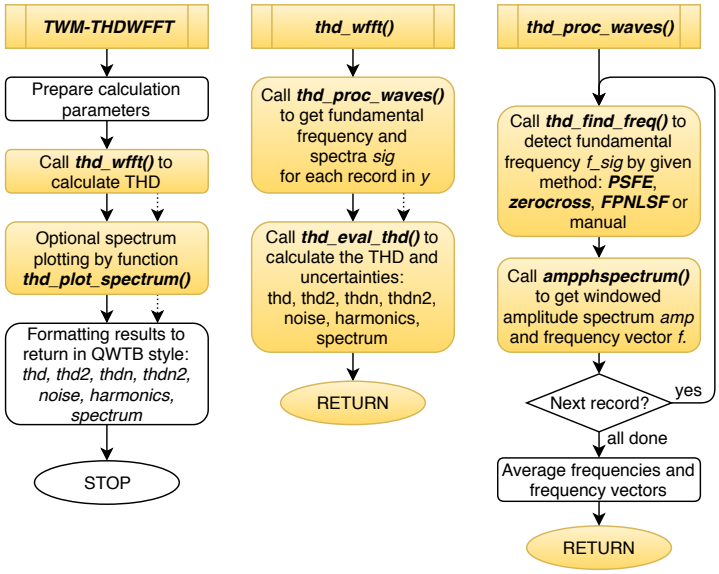

Figure 15: Flow chart of the algorithm wrapper TWM-THDWFFT. Note the rounded gold cells are calls to other local functions which are shown in another diagram or mentioned in the text.

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 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.

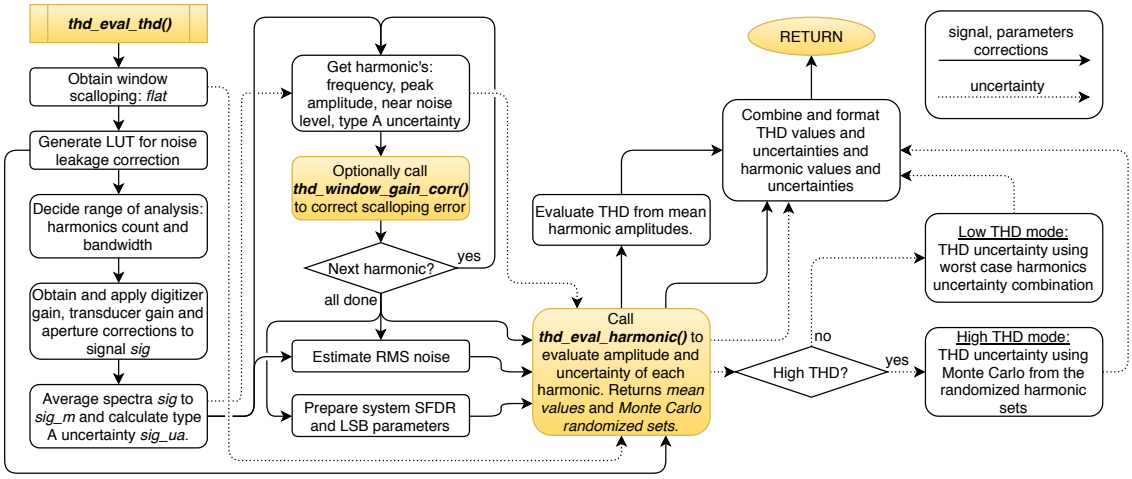

Figure 16: Flow chart of the main algorithm function "thd_eval_thd()" for the TWM-THDWFFT algorithm.

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 *band*width).
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:

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

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

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

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):

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

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]).

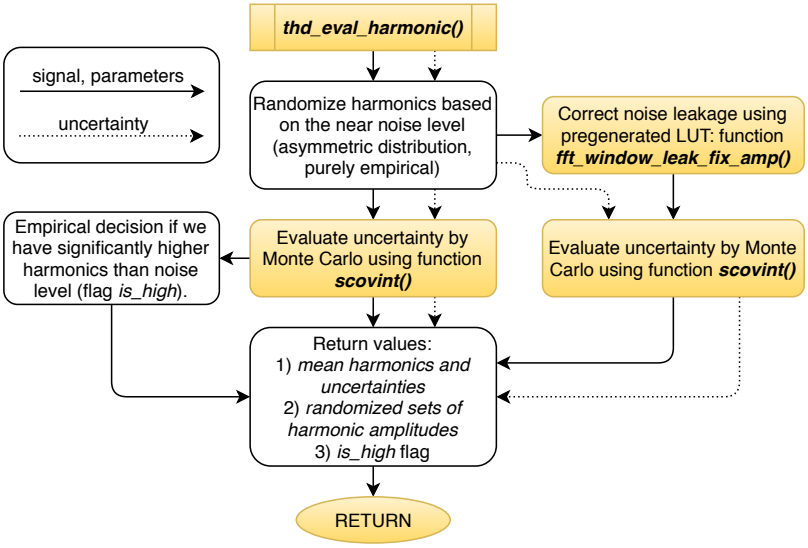

Figure 17: Flow chart of the function "thd_eval_harmonic()" of the TWM-THDWFFT algorithm.

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.

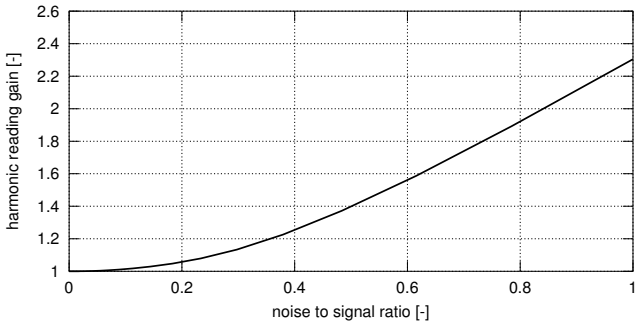

Figure 18: Error of the harmonics amplitude measurement using FFT with window Flattop HFT248D. Note the "noise" means amplitude of the noise in surrounding DFT bins, not RMS noise.

.

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.

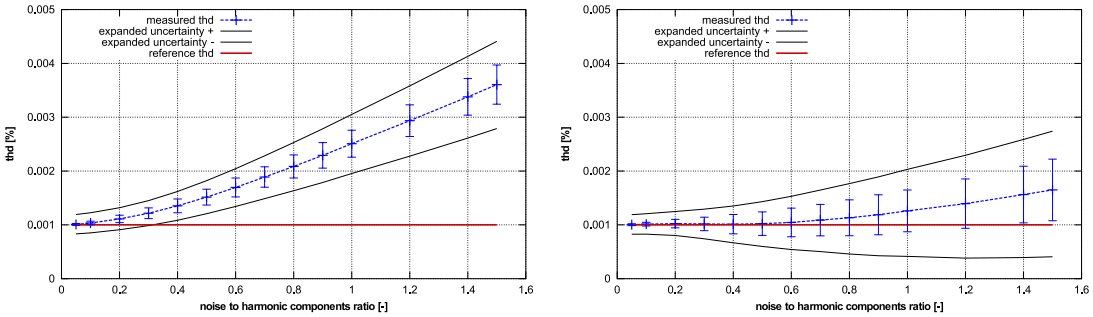

Figure 19: Deviation of THDWFFT algorithm from simulated THD level 10 ppm for various noise to higher harmonic ratios. The simulated waveform has 10 harmonic components with amplitudes $U_{\mathrm{m}} = \{0.9, 3\cdot 10^{-6}, 3\cdot 10^{-6}, \dots\}\,\mathrm{V}$. Left graph shows results without noise 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

TWM-PWRTDI is an algorithm for calculation of power parameters using a time domain integration of $u(t) \cdot i(t)$ product. It is based on the use of window function to eliminate effects of non-coherent sampling as was demonstrated in [1] and [4]. Therefore, it does work even for non-coherently sampled waveforms. The algorithm can easily reach errors below 1 µW/VA with proper selection of a sampling rate and windows size.

The algorithm can calculate all basic parameters: active power P, reactive power Q, apparent power S, RMS voltage U, RMS current I and power factor PF. It also returns DC components separately: Udc, Idc and Pdc. User may choose optional AC coupling mode by setting parameter "$ac_coupling = 1$" in which case the U, I, P, Q, S and PF will be calculated without the AC component.

Note the windowed RMS method itself can calculate power in any quadrant, however it is not able to distinguish all four quadrants. The quadrant identification (proper signs for P and Q) is obtained from a complementary windowed FFT algorithm which is running along the main RMS calculation. Note the quadrant selection may fail around $PF = 0$ (the absolute values will be correct).

The algorithm uses following definitions for the power components: (i) The AC power components P, Q and S are related by equation:

$$S^2 = P^2 + Q^2. \tag{19}$$

(ii) Power factor PF is calculated including DC components according to equation:

$$PF = \frac{P}{S}. \tag{20}$$

(iii) The sign of Q is calculated using harmonic components method according Budenau definition:

$$\mathrm{sing}(Q) = \mathrm{sign}\left\{\sum_{h=1}^{H}(U(h) \cdot I(h) \cdot \sin\phi(h))\right\}, \tag{21}$$

where h is harmonic index, H is harmonics count, $U(h)$, $I(h)$ and $\phi(h)$ are harmonic voltage, current and phase shift. Note the absolute value of Q is still calculated from AC components following equation 19. Only the sign of Q is decided from the Budenau definition 21.

The TWM-PWRTDI algorithm wrapper is able to use single-ended or differential input sensors for voltage channel, current channel or both. The algorithm is also equipped by a fast uncertainty estimator and the Monte Carlo uncertainty calculation method for more accurate but slower uncertainty evaluation.

TODO: Brief windowed RMS description by JV: something on the windowing, selection of window (Blackman Harris), limitations, ...

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 [5].

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.

Table 23: List of input quantities to the TWM-PWRTDI wrapper.
Details on the correction quantities can be found in [5].

Name	**Default**	**Unc.**	**Description**
u_time_shift_lo	0	Yes	Time shift between high-side channel u low-side channel u_lo (or i and i_lo for current).
i_time_shift_lo	0	Yes	
u_lsb	N/A	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_nrng	1000	No	
u_adc_bits	40	No	
u_lo_lsb	N/A	No	
u_lo_adc_nrng	1000	No	
u_lo_adc_bits	40	No	
i_lsb	N/A	No	
i_adc_nrng	1000	No	
i_adc_bits	40	No	
i_lo_lsb	N/A	No	
i_lo_adc_nrng	1000	No	
i_lo_adc_bits	40	No	
u_adc_offset	0	Yes	Digitizer input offset voltage.
u_lo_adc_offset	0	Yes	
i_adc_offset	0	Yes	
i_lo_adc_offset	0	Yes	
u_adc_gain	1	Yes	Digitizer gain correction 2D table (multiplier).
u_adc_gain_f	[]	No	
u_adc_gain_a	[]	No	
u_lo_adc_gain	1	Yes	
u_lo_adc_gain_f	[]	No	
u_lo_adc_gain_a	[]	No	
i_adc_gain	1	Yes	
i_adc_gain_f	[]	No	
i_adc_gain_a	[]	No	
i_lo_adc_gain	1	Yes	
i_lo_adc_gain_f	[]	No	
i_lo_adc_gain_a	[]	No	
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	

Table 23: List of input quantities to the TWM-PWRTDI wrapper.
Details on the correction quantities can be found in [5].

Name	**Default**	**Unc.**	**Description**
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	
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_sfdr	180	No	Transducer SFDR 2D table.
u_tr_sfdr_f	[]	No	
u_tr_sfdr_a	[]	No	
i_tr_sfdr	180	No	
i_tr_sfdr_f	[]	No	
i_tr_sfdr_a	[]	No	

Table 23: List of input quantities to the TWM-PWRTDI wrapper.
Details on the correction quantities can be found in [5].

Name	**Default**	**Unc.**	**Description**
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	
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-PWRTDI 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.mcm.method	Monte Carlo evaluation mode: "singlecore" - single core evaluation, "multicore" - Parallel evaluation using "parcellfun" for GNU Octave or "parfor" for Matlab "multistation" - Multicore evaluation using "multicore" package (GNU Octave only yet).
calcset.mcm.repeats	Monte Carlo iterations count. Use at least 100 to get any usable estimate.
calcset.mcm.proc_no	Number of parallel instances to use for the paralleled modes. Use zero value to not start any server processes for the "multistation" mode. This option expects user started the server processes manually in the shared folder. This option causes less overhead for the batch processing or runtime calculations.
calcset.mcm.tmpdir	Jobs sharing folder for the "multistation" mode. This should be an absolute path to the sharing folder. Keep in mind the package "multicore" will erase the content of this folder before each new calculation!
calcset.mcm.user_fun	User function to call in the "multistation" mode after startup of the serve processes. Example: "calcset.mcm.user_fun = @coklbind2". Leave empty to not execute any function.
calcset.loc	Level of confidence [-].
calcset.verbose	Verbose level.

7.2 Algorithm description

The TWM-PWRTDI wrapper starts with automatic cropping of the input data to a size of multiple of two, which is required by the algorithm core. Next, the wrapper creates two virtual channels, one for voltage and one for current, because the applied corrections are identical for both so the code duplication is minimised. It also creates virtual sub-channels for the low-side signals *u_lo* and *i_lo* (differential mode). Next, the wrapper calculates windowed spectra of each input signal u and i (and *u_lo*, *i_lo*) and also splits the time domain signals u and i (and *u_lo*, *i_lo*) to AC and DC components which are treated separately.

The next step are the frequency dependent corrections of the input signals. Note the windowed RMS (WRMS) algorithm itself operates in time domain, so the frequency dependent corrections of the gain and phase must be applied in time domain to u and i (and u_lo, i_lo). However, the uncertainty calculator/estimator needs a spectrum, i.e. frequency domain representation of the time domain signals. Therefore, the correcting code does two things at once: (i) It calculates total combined gain and phase corrections for each virtual channel but does not apply it to time domain data u and i (and u_lo, i_lo) yet; (ii) It applies the same corrections to the frequency domain representation only. At the same time the spectra of the eventual differential pair u and u_lo (and i and i_lo) are merged to a single single-ended spectrum of u and i.

In the next step, the wrapper calls the main WRMS function "proc_wrms()", which applies the calculated corrections in time domain to u, i (and u_lo, i_lo), calculates the differential time domain signal (for differential mode only) and calculates the AC RMS parameters U, I, AC power P (see below) and the DC components U_dc and I_dc. Follows the uncertainty calculation (see below) of the RMS and active power quantities and expression of the other desired quantities and their uncertainties. For simplicity the uncertainty calculator calculates only uncertainty of the RMS voltage, RMS current a active power. The remaining uncertainties are calculated from these three. The definitions of particular parameters is shown in the table 26. Note the Q_bud quantity in the table is reactive power estimated from the FFT spectra of voltage and current according the Budenau definition:

$$Q\text{\_bud} = \sum_{h=1}^{H} (0.5 \cdot U(h) \cdot I(h) \cdot \sin\phi(h)), \tag{22}$$

where h is harmonic component index, H is total harmonics count, $U(h)$ and $I(h)$ are harmonic voltage and current amplitudes and $\phi(h)$ is voltage to current phase shift. The purpose of the $(sign)(Q\text{\_bud})$ term is to distinguish correct polarity of Q, because the WRMS algorithm itself cannot distinguish all four quadrants of power. One solution would be to use Hilbert transformation on either voltage or current waveform and repeat the WRMS calculation. This would allow to calculate the Q directly, however such solution seemed to complex. Therefore, instead of the Hilbert transform, the sign of Q was obtained from the Budenau's definition. Such solution should work reliably if power factor is $|PF| > 0.05$ and there are no excessive harmonics.

Table 26: Definitions of the TWM-PWRTDI output quantities based on the basic quantities U, I, P, U_dc and I_dc. Note the input quantities marked * are obtained from the other output quantity.

Returned quantity	**AC coupled definition**	**DC coupled definition**
DC voltage Udc	U_dc	U_dc
DC current Idc	I_dc	I_dc
DC power component Pdc	$U\_dc \cdot I\_dc$	$U\_dc \cdot I\_dc$
RMS voltage U	U	$\sqrt{U^2 + U\_dc^2}$
RMS current I	I	$\sqrt{I^2 + I\_dc^2}$
Active power P	P	$P + U \cdot I$
Reactive power Q	$\sqrt{(UI)^2 - P^2} \cdot \text{sign}(Q\text{\_bud})$	$\sqrt{(U \cdot I)^2 - P^2} \cdot \text{sign}(Q\text{\_bud})$
Apparent power S	$(UI)^2$	$\sqrt{U^2 + U\_dc^2} \cdot \sqrt{I^2 + I\_dc^2}$
Power factor PF	$P/S*$	$P/S*$
Effective phase angle phi_ef	$\arctan2(Q*, P)$	$\arctan2(Q*, P)$

7.2.1 Windowed RMS function "proc_wrms()"

The core of the algorithm is function "proc_wrms()". Before the algorithm itself is described, it must be noted the "proc_wrms()" function was made in such a way it can be used for a Monte Carlo uncertainty evaluation in a parallel manner. I.e. the function is called once for each Monte Carlo iteration (see below). This affected its structure.

The function can be executed either on the input time domain signals u, i (and u_lo, i_lo) which is used for evaluation of the power parameters and it can be also called repeatedly in the simulator mode, where it calculates the power of synthesized signals with prescribed parameters randomized by

input uncertainties (Monte Carlo calculation mode). The synthesizer of the waveforms is included in this function, so the following section will show its function as well.

The function starts with selection of the mode of operation. For the simulated mode, it overrides input signals u, i (and u_lo, i_lo) by a synthesized ones with known spectral components and known total power, power factor, etc. It also applies various distortions, e.g. noise, quantisation errors, ...

Follows the calculation part which is common for both modes of operation. First, it applies the time domain frequency dependent correction of each virtual channel (u, i, u_lo and i_lo)). This is done by the "td_fft_filter()". Note the function "td_fft_filter()" itself also estimates the phase errors it causes, which is used only in the calculation mode (they are ignored in the simulating mode). The DC corrections to the previously split DC components are also applied in this step.

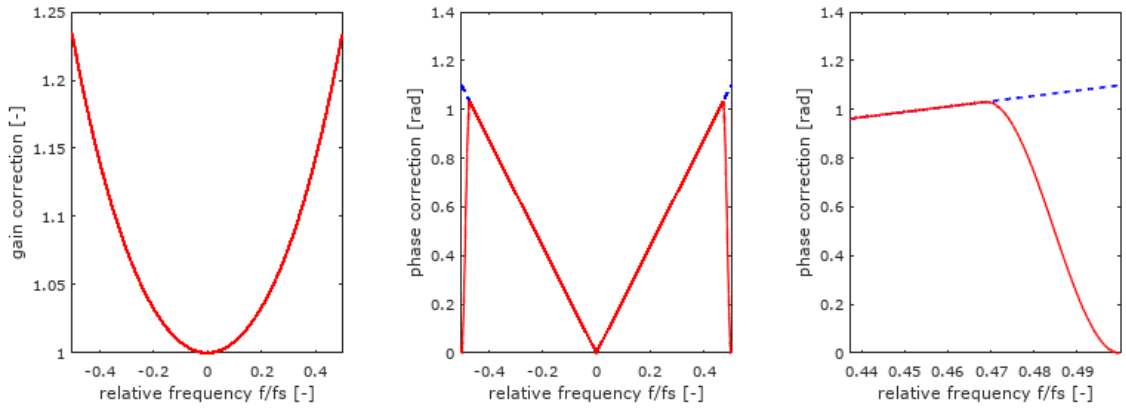

Figure 20: Example of FFT filter profile. From left right: Gain correction, Phase correction, Detail of phase correction. Blue - desired correction, red - applied correction.

The function "td_fft_filter()" is critical for correct functionality of the whole algorithm. The function is a frequency domain filter executed per smaller windows. The function does following steps:

1. Building a filter spectral profile from the gain/phase correction data for the positive frequencies.
2. Empirical post-processing of the filter profile to suppress the errors caused by high phase angle correction. This step applies weighting mask to the phase of the filter at its ends (at DC and Nyquist frequency). So both "ends" of the filter have zero phase. This drastically reduces inherent errors of this filtering method. See fig. 20 for an example.
3. Estimation of the filter post-processing errors on the phase.
4. Extending the filter profile to negative frequencies.
5. Performing the spectral filtering per small overlapping windows by a function "sFreqDep_PG_Comp(y, fft_size, filter)":
   (a) Obtaining Hann window function coefficients w of size $fft\_size/2$.
   (b) Extend Hann by zero padding: $w = [\text{zeros}(1, fft\_size/4), w, \text{zeros}(1, fft\_size/4)]$ to a total size of fft_size.
   (c) Window fft_size input samples $y(offset : offset + fft\_size - 1)$ by the window w.
   (d) Performing FFT of the windowed portion of y to get spectrum $\hat{Y}$.
   (e) Applying $filter$ profile to the spectrum $\hat{Y}$.
   (f) Performing inverse-FFT of the $\hat{Y}$ to get time domain representation y_frame.
   (g) Extracting $< fft\_size/4; fft\_size \cdot 3/4 >$ samples of y_frame.
   (h) Merging y_frame samples with overlapping samples of output signal yf and storing result to $yf(offset + fft\_size/2)$.

(i) Move $offset$ to next frame by $fft\_size/4$ samples and repeat from step 5c for all possible frames.

6. Removing invalid portions of the filtered signal ($fft\_size/2$ samples from beginning and end of signal).

TODO: Verify by JV

In the next step the "proc_wrms()" evaluates the RMS parameters U, I and P according the following definitions:

$$U = \frac{1}{W} \cdot \sqrt{\sum_{k=1}^{N} w(k)^2 \cdot u(k)^2}, \tag{23}$$

$$I = \frac{1}{W} \cdot \sqrt{\sum_{k=1}^{N} w(k)^2 \cdot i(k)^2}, \tag{24}$$

$$P = \frac{1}{W^2} \cdot \sqrt{\sum_{k=1}^{N} w(k)^2 \cdot u(k) \cdot i(k)}, \tag{25}$$

$$W = \sqrt{\sum_{k=1}^{N} w(k)^2}, \tag{26}$$

where k is a sample index, N is total samples count and $w(k)$ is k-th sample of the window function Blackman-Harris. For clarity, the Blackman-Harris window used in the WRMS algorithm is defined by [2]:

$$w(k) = 0.35875 - 0.48829 \cdot \cos\frac{2\pi k}{N} + 0.14128 \cdot \cos\frac{4\pi k}{N} - 0.01168 \cdot \cos\frac{6\pi k}{N}. \tag{27}$$

Note the coefficients are exact (not rounded). This window was selected because the calculation errors of the RMS parameters due to the non-coherency decays fastest with growing number of signal periods (see [1]).

Last step of the function "proc_wrms()" is performed only for the simulation mode. The function calculates the deviation of calculated power from the theoretical synthesized power (used for Monte Carlo evaluation).

7.3 Uncertainty calculator and estimator

The TWM-PWRTDI algorithm wrapper is equipped by two modes of uncertainty evaluation: (i) The ordinary Monte Carlo (MC) calculator; (ii) Fast estimator based on several precalculated lookup tables (LUT). The MC mode is more accurate, however it may take up to several minutes to perform even just a few hundreds of iterations. The calculation time drastically rises with the length of the record. Thus the fast estimator was created as well.

The WRMS algorithm itself can calculate a power of any voltage and current waveforms. However, calculation of uncertainty for general non-periodic waveforms would be extremely complex and slow. Therefore, the uncertainty calculation is based on the analysis of spectral components obtained from the average spectrum of the whole digitized waveform. This simplification should not have any effect, as the algorithm is primarily intended for a calibration of stationary signals.

The first step of the uncertainty evaluation for both modes is spectral analysis of the voltage and current spectra $U\mathrm{h}(fh)$, $I\mathrm{h}(fh)$ and phase shifts between the voltage and current harmonic components $ph(fh)$. These spectra along with their uncertainties were obtained in the corrections section (see section 7.2). The algorithm identifies the fundamental component of power by searching the dominant voltage harmonic assuming the fundamental voltage harmonic should be always present. The algorithm then searches up to 100 spectral components $Ux(fh)$, $Ix(fh)$ and $phx(fh)$, whose current or voltage amplitude exceeds certain threshold relative to the fundamental harmonic. These spectral components are successively removed from the spectra $U\mathrm{h}(fh)$ and $I\mathrm{h}(fh)$. Whatever is left after the removal is considered and later used as a residual RMS noise estimate. The identified spectral components will be

marked $Ux(h)$, $Ix(h)$ (amplitudes) and $phx(h)$ in the following text. Their uncertainties coming from the correction uncertainties will be marked $u\_Ux(h)$, $u\_Ix(h)$, $u\_phx(h)$. The h is index of harmonic from one to H.

The identified spectral components $Ux(h)$, $Ix(h)$ and $phx(h)$ are used to calculate estimate of the RMS and power parameters following the Budenau definition:

$$U\_rms = \sqrt{\sum_{h=1}^{H} 0.5 \cdot Ux(h)^2}, \tag{28}$$

$$I\_rms = \sqrt{\sum_{h=1}^{H} 0.5 \cdot Ix(h)^2}, \tag{29}$$

$$P = 0.5 \cdot \sum_{h=1}^{H} Ix(h) \cdot Ix(h) \cdot \cos(phx(h)), \tag{30}$$

$$Q = 0.5 \cdot \sum_{h=1}^{H} Ix(h) \cdot Ix(h) \cdot \sin(phx(h)), \tag{31}$$

$$S = 0.5 \cdot \sum_{h=1}^{H} Ix(h) \cdot Ix(h), \tag{32}$$

$$\tag{33}$$

Following algorithm structure differ for Monte Carlo and Estimator.

7.3.1 Monte Carlo uncertainty calculator

The MC uncertainty calculator is very straightforward. It starts with preparation of the two virtual channels. One for voltage and one for current. Each channel contains a list of the identified spectral components $Ux(h)$, $Ix(h)$ and $phx(h)$ to synthesize with assigned uncertainties coming from the corrections $u\_Ux(h)$, $u\_Ix(h)$, $u\_phx(h)$. Each channel have also assigned RMS noise, system SFDR and LSB of the ADC. Then the WRMS processing function "proc_wrms()" is called in the simulator mode repeatedly for each iteration of MC. The returned randomised lists of RMS voltage, RMS current and active power are evaluated according GUM annex 1 [3] using a function "scovint()" to obtain absolute final uncertainties of quantities U, I and P. The uncertainties of the other returned quantities are evaluated in the wrapper (see above) from these three uncertainties.

Note the MC evaluator itself uses function "qwtb_mcm_exec()". This function is internally designed to enable parallel calculation of the MC iteration cycles. It is offers three modes of parallelisation:

1. **calcset.mcm.method = 'singlecore'**: Single core calculation.
2. **calcset.mcm.method = 'multicore'**: Multicore operation using "parcellfun()" from "parallel" package for GNU Octave or "parfor" for Matlab. Note the use of Matlab's "parfor" for parallelisation is just a user wish. Actual parallelisation mode is decided by Matlab. The package "parcellfun()" implementation does work only for Linux. Windows implementation was not functional at least up to GNU Octave version 4.2.2.
3. **calcset.mcm.method = 'multistation'**: Multiprocess/multistation calculation using "multicore" package for GNU Octave (Matlab is not supported yet). Note the The "multistation" method requires to define shared folder path for the job files. Otherwise it will create the shared folder in temp folder, which may not be appreciated by the SSD disks owners. The mode "multistation" also have one specific feature. It can initiate the user function after startup of the server processes. The function is defined in the "calcset.mcm.user_fun" variable. The example of the use for this optional input is CMI's supercomputer "Čokl" [6] which requires to call a special script to assign server processes to particular CPU cores.

See table 25 for list of the additional parameters. Note at least 100 iterations is the absolute minimum for which the MC mode provides any usable uncertainty estimates. The processing time for an evaluation

at 4 cores with 1000 cycles and $N = 10000$ input samples is typically below one minute. However, the situation may change drastically when high count of harmonic components is presents in the signal.

7.3.2 Fast uncertainty estimator

The uncertainty estimator is significantly more complex than the Monte Carlo calculator. It consists of four separate estimation routines which contributes to the final uncertainty: (i) Uncertainty of time-domain frequency dependent filter "td_fft_filter()"; (ii) Uncertainty of a system SFDR; (iii) Uncertainty WRMS algorithm for a single tone signal; (iv) Uncertainty of WRMS mutual harmonic effects. These are executed in order. The estimation routine (i) is affecting steps (ii), (iii) and (iv). The obtained uncertainty components from the particular steps are combined to the total uncertainty of U, I and P using following equations:

$$u\_U = \sqrt{u\_U\_st^2 + u\_U\_sp^2 + u\_U\_sfdr^2 + \sum_{h=1}^{H} 0.5 u_U x(h)^2}, \quad (34)$$

$$u\_I = \sqrt{u\_I\_st^2 + u\_I\_sp^2 + u\_I\_sfdr^2 + \sum_{h=1}^{H} 0.5 u_I x(h)^2}, \quad (35)$$

$$u\_P = \sqrt{u\_P\_st^2 + u\_P\_sp^2 + u\_P\_sfdr^2 + \sum_{h=1}^{H} 0.5 u_P x(h)^2}, \quad (36)$$

where the components are described in the following sections. The uncertainties of the other returned quantities are evaluated in the wrapper (see above) from these three uncertainties.

7.3.2.1 Uncertainty of time-domain frequency dependent filter "td_fft_filter()"

The first estimator quantifies the errors introduced by the frequency dependent gain and phase filter "td_fft_filter()". The uncertainty estimate is calculated using a complementary function "td_fft_filter_unc()", which is called for voltage and current channel separately. This estimator is far most problematic. The "td_fft_filter()" is using the time-¿frequency-¿time conversion (filtering in frequency domain per small sized windows). The filtering in the frequency domain can cause hardly predictable errors when high phase shift corrections are applied. This is especially problematic if there are significant spectral components near the Nyquist frequency or near DC. The function have too many degrees of freedom to create a simple but reliable estimator. Therefore, the only usable solution found to this problem was to perform a small scale Monte Carlo (MC). In particular, 10 cycles of following steps are performed:

1. Synthesize waveform with spectral components $Ux(h)$ (resp. $Ix(h)$) with random phase angles $\phi(h)$ and with uncertain frequency ± 1 DFT bin, because accuracy of the frequency estimation from FFT spectral analysis is limited.
2. Perform the frequency dependent filtering by "td_fft_filter()" with calculated channel gain and phase correction data and calculate spectrum.
3. Perform the frequency dependent correction with the same correction data in frequency domain to the spectral components $Ux(h)$ (resp. $Ix(h)$) and the generated phase angles $\phi(h)$.
4. Compare the difference of the spectral components from 3) and 2) to estimate the filter error.

The set of error estimates is processed to find a maximum probable amplitude and phase error of each spectral component. Surprisingly this simple solution provides reliable estimates.

This gain uncertainties $u\_fa\_U(h)$, $u\_fa\_I(h)$ and phase uncertainties $u\_fp\_U(h)$, $u\_fp\_I(h)$ obtained by this estimator are used to expand the correction uncertainties $u\_Ux(h)$, $u\_Ix(h)$ and $u\_phx(h)$ before following estimator steps:

$$u\_Ux(h) = \sqrt{u\_Ux(h)^2 + u\_fa\_U(h)^2}, \quad (37)$$

$$u\_Ix(h) = \sqrt{u\_Ix(h)^2 + u\_fa\_I(h)^2}, \quad (38)$$

$$u\_phx(h) = \sqrt{u\_phx(h)^2 + u\_fp\_U(h)^2 + u\_fp\_I(h)^2}. \quad (39)$$

7.3.2.2 Uncertainty of a system SFDR

The next step is estimation of the uncertainty introduced by the system SFDR (digitizer and transducer). The effect on the U, I and P is estimated as a wost case combination of the spurs according:

$$u\_U\_sfdr = \frac{1}{\sqrt{3}} \cdot \left(\sqrt{U^2 + \sum_{s=1}^{S} 0.5 \cdot U\_spur(s)^2} - U \right), \tag{40}$$

$$u\_I\_sfdr = \frac{1}{\sqrt{3}} \cdot \left(\sqrt{I^2 + \sum_{s=1}^{S} 0.5 \cdot I\_spur(s)^2} - I \right), \tag{41}$$

$$u\_P\_sfdr = \frac{1}{\sqrt{3}} \cdot \sqrt{\sum_{s=1}^{S} 0.5 \cdot U\_spur(s) \cdot I\_spur(s)}, \tag{42}$$

where U_spur and I_spur are vectors of voltage and current spur amplitudes (2nd, 3rd, 4th harmonic, etc.), s is spur index and S is spurs count.

7.3.2.3 Uncertainty WRMS algorithm for a single tone signal

This step is estimation of the WRMS algorithm error for each identified spectral component $Ux(h)$, $Ix(h)$. Note this estimator does not cover mutual effects between spectral components. It is just a single tone estimator for each component. The estimation is performed by the function "wrms_unc_st()". This function uses precalculated LUT (see below) and empiric formulas to form an uncertainty interpolator dependent on following parameters: (i) Voltage amplitude, (ii) Current amplitude, (iii) Voltage noise, (iv) Current noise, (v) Voltage bit resolution, (vi) Current bit resolution, (vii) Periods count in the waveform, (viii) Samples per period. The usable range of each parameter for the "wrms_unc_st()" is shown in table 28. This interpolator returns a relative uncertainty estimate of frequency component voltage $u\_U\_st'(f)$, current $u\_I\_st'(f)$ and active power $u\_P\_st'(f)$. The component uncertainties are converted to absolute and combined to obtain RMS uncertainties:

$$u\_U\_st = \sqrt{\frac{\sum_{f=1}^{F} u\_U\_st'(f)^2 \cdot Ux(f)^2}{\sum_{f=1}^{F} Ux(f)^2}}, \tag{43}$$

$$u\_I\_st = \sqrt{\frac{\sum_{f=1}^{F} u\_I\_st'(f)^2 \cdot Ix(f)^2}{\sum_{f=1}^{F} Ix(f)^2}}, \tag{44}$$

$$u\_P\_st = \sqrt{\sum_{f=1}^{F} u\_P\_st'(f)^2}. \tag{45}$$

The LUT table itself for the interpolator of "wrms_unc_st()" was calculated as a worst case error of the WRMS algorithm from 50000 Monte Carlo iterations. The simulation was performed for each combination of parameters shown in table 27. I.e. 11x12x15x9x5 = 89100 combinations were calculated using a supercomputer (processing time roughly two days on 300 cores). Note the phase shift between voltage and current was randomised for each iteration as it had no effect on the relative active power uncertainty (when expressed in units W/VA). The bit resolution of one channel was held fixed at 32 bit as the error is defined by the worse of the voltage and current channel. Noise was also simulated for the worse of the channels only. Manual inspection of the obtained 5-dimensional space of uncertainties showed only the parameters "Periods count" and "Samples per period" are necessary in the LUT as they have hardly expressible shape given by the window function. The other parameters effects were approximated by empirical formulas. Therefore, the LUT size after compression is only 2.5 kBytes, which is perfectly acceptable.

Table 27: Simulation ranges and steps of the parameters for uncertainty estimator of "wrms_unc_st()" function.

Name	**Description**
Periods count	List: [3; 4; 5; 6; 7; 9; 11; 15; 20; 50; 100], 11 steps
Samples per period	Log. space: 7 to 100, 12 steps
U to I phase shift	random*
U to I amplitude ratio	Log. space: 0.01 to 1, 15 steps
U bit resolution	32 bits
I bit resolution	Log. space: 4 to 32 bits, 9 steps
max(U noise,I noise)	Log. space: 10^{-7} to 10^{-3}, 5 steps

Table 28: The usable range of input parameters of the single tone error estimator "wrms_unc_st()". 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**
Voltage amplitude	0 to ∞	const	const
Current amplitude	0 to ∞	const	const
Voltage noise	0 to ∞	const	const
Current noise	0 to ∞	const	const
Voltage bit resolution	4 to ∞	error	const
Current bit resolution	4 to ∞	error	const
Periods count	3 to ∞	error	const
Samples per period	7 to ∞	const	const

7.3.2.4 Uncertainty of WRMS mutual harmonic effects

This estimator calculates mutual effect of spur harmonic to analysed harmonic. The paper [1] shows example of these errors, however this effect had to be quantified extensively for means of the uncertainty estimator. Theoretically the calculation of mutual effect should be performed for each pair of components $Ux(h)$, $Ix(h)$, $phx(h)$, however that would be very slow. So and assumption was made the fundamental harmonic carries dominant portion of the active power, RMS voltage and current, so the mutual effects calculation is performed only between the fundamental component $Ux(1)$, $Ix(1)$, $phx(1)$ and the rest of components $Ux(h)$, $Ix(h)$, $phx(h)$, where $h \geq 2$. The estimation of the mutual effect itself is done by estimator function "wrms_unc_spur()". This function is an interpolator dependent on: (i) Reference harmonic voltage, (ii) Reference harmonic current, (iii) Relative spur frequency, (iv) Spur harmonic voltage, (v) Spur harmonic current, (vi) Periods of reference harmonic, (vii) Samples per period of reference harmonic. The estimator function is called once for each spur component to obtain uncertainty components $u\_U\_sp'(h-1)$, $u\_I\_sp'(h-1)$ and $u\_P\_sp'(h-1)$. These $(H-1)$ components are combined to a combined uncertainty due to the spur components:

$$u\_U\_sp(h) = \sqrt{\frac{\sum_{h=1}^{H-1} u\_U\_sp'(h)^2 \cdot Ux(h+1)^2}{\sum_{h=1}^{H} Ux(h)^2}}, \tag{46}$$

$$u\_I\_sp(h) = \sqrt{\frac{\sum_{h=1}^{H-1} u\_I\_sp'(h)^2 \cdot Ix(h+1)^2}{\sum_{h=1}^{H} Ix(h)^2}}, \tag{47}$$

$$u\_P\_sp(h) = \sqrt{\sum_{h=1}^{H-1} u\_P\_sp'(h)^2}. \tag{48}$$

The interpolator "wrms_unc_spurr()" itself is a combination of two precalculated LUT tables and empiric formulas. First LUT is used to estimate the uncertainty of active power. It was obtained as a worst case error of 1000 Monte Carlo iterations performed each parameter combination (11x10x5x9x15 = 74250 combinations) shown in table 29. The simulation was performed without noise or quantisation

errors, because these are already covered by the single tone WRMS estimator (section 7.3.2.3). The simulator performed following steps for each combination:

1. Synthesize waveforms with known apparent power S_ref with no spurs.
2. Calculate active power using the WRMS algorithm P_dut_0.
3. Synthesize waveforms with known active power with spur at voltage channel.
4. Calculate active power using the WRMS algorithm P_dut_tot.
5. Calculate relative error of the WRMS power: $\delta P(i) = |P\_dut\_tot - P\_dut\_0|/S\_ref$.
6. Repeat 1000 times from step 1.
7. Estimate worst case uncertainty: $\delta P = \max \delta P(i)$ for $i = 1..1000$.

The simulator for the first LUT generates spur only for voltage channel, because the voltage and current channels are interchangeable. Thus the same LUT is used twice for the estimation (once for voltage spur effect and once for current spur effect with swapped voltage and current parameters).

The second LUT is used to estimate the spur effect to the RMS voltage (or RMS current). It was simulated together with the first LUT. Only exception is the resulting LUT does not contain axis "U to I amplitude ratio", as it obviously has no effect to RMS level of single channel. So the table contains only 11x10x9x15 = 14850 combinations. The procedure performed for each combination is following:

1. Synthesize waveform with known RMS level A_ref with no spurs.
2. Calculate RMS level using the WRMS algorithm: A_dut.
3. Synthesize waveform with known RMS level with spur of RMS level S_ref.
4. Calculate RMS level using the WRMS algorithm: A_dut_tot.
5. Calculate relative error of the WRMS RMS amplitude: $\delta A(i) = (\sqrt{|A\_dut\_tot^2 - S\_ref^2|} - A\_ref)/A\_ref$.
6. Repeat 1000 times from step 1.
7. Estimate worst case uncertainty: $\delta A = \max \delta A(i)$ for $i = 1..1000$.

This LUT called twice to obtain spur effects for RMS voltage and RMS current.

Total size of both LUTs after compression is 265 kBytes which is still acceptable, so no further optimalisations were performed. Total range of input parameters to the estimator "wrms_unc_spur()" is shown in table 30.

Table 29: Simulation ranges and steps of the parameters for uncertainty LUT of "wrms_unc_spur()" function.

Name	**Description**
Periods count	List: [3; 4; 5; 6; 7; 9; 11; 15; 20; 50; 100], 11 steps
Samples per period	Log. space: 7 to 100, 10 steps
U to I phase shift	random*
U to I amplitude ratio	Log. space: 0.01 to 1, 5 steps
U or I spur amplitude	Log. space: 0.001 to 1, 9 steps
Relative spur frequency	Log. space: 0.001 to 0.9, 15 steps

Table 30: The usable range of input parameters of the single tone error estimator "wrms_unc_spur()". 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**
Ref. component voltage	0 to ∞	const	const
Ref. component current	0 to ∞	const	const
Relative spur frequency	0.001 to 0.9	const	const
Spur component voltage	0 to ∞	const	const
Spur component current	0 to ∞	const	const
Periods count	3 to 100	error	const
Samples per period	7 to 100	const	const

References

[1] K. B. Ellingsberg. Predictable maximum rms-error for windowed rms (rmws). In *2012 Conference on Precision electromagnetic Measurements*, pages 308–309, July 2012.

[2] 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.

[3] 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.

[4] R. Lapuh, B. Voljč, and M. Lindič. Measurement and estimation of arbitrary signal power using a window technique. In *2016 Conference on Precision Electromagnetic Measurements (CPEM 2016)*, pages 1–2, July 2016.

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

[6] Miroslav Valtr. ČMI HPC System Online. `https://translate.google.cz/translate?sl=cs&tl=en&js=y&prev=_t&hl=cs&ie=UTF-8&u=http%3A%2F%2Fprutok.cmi.cz%2Fsc%2Fdoku.php%3Fid%3Dsystem&edit-text=`, 2014.

8 TWM-Flicker - Flicker algorithm

The TWM wrapper TWM-Flicker is an algorithm for evaluation of the short term flicker parameters. It calculates instantaneous flicker sensation $Pinst$ and short-term flicker severity Pst Sampling rate has to be higher than 7 kHz. If sampling rate is higher than 23 kHz, signal will be down sampled by algorithm. More than 600 s of signal is required as the algorithm needs at least a minute to settle the filters. Typical sampling time value is above 660 s. The algorithm requires either Signal Processing Toolbox when run in MATLAB or a signal package when run in GNU Octave. Frequency of line (carrier frequency) f_line can be only 50 or 60 Hz.

The algorithm was implemented according IEC 61000-4-15 [3], [4], [2] and [5].

The algorithm wrapper is equipped by a simple uncertainty estimator based on the worst observed error of the algorithm on the tabulated Pst values for various sampling rates.

Note the algorithm output slightly differ for Matlab and GNU Octave implementation. The cause of this difference was not yet identified. Also the observed performance in the Matlab 2017b was about five times higher then in GNU Octave 4.2.2 on the same computer.

8.1 TWM wrapper parameters

The input quantities supported by the algorithm are shown in the table 31. Algorithm returns output quantities shown in the table 32. Calculation setup supported by the algorithm is shown in table 33.

Table 31: List of input quantities to the TWM-Flicker wrapper.

Name	Default	Unc.	Description
f_line	N/A	N/A	Nominal frequency of the network (50 HZ or 60 Hz).
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.
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 [] [] 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: $A' = A \cdot pi \cdot adc\_aper \cdot f\_est/\sin(pi \cdot adc\_aper \cdot f\_est)$ $phi' = phi + pi \cdot adc\_aper \cdot f\_est$
adc_Yin_Cp adc_Yin_Gp adc_Yin_f	1e-15 1e-15 []	Yes Yes No	Digitizer input admittance 1D table.
adc_sfdr adc_sfdr_f adc_sfdr_a	180 [] []	No No No	Digitizer SFDR 2D table.

Table 31: List of input quantities to the TWM-Flicker wrapper.

Name	Default	Unc.	Description
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_sfdr	180	No	Transducer SFDR 2D table.
tr_sfdr_f	[]	No	
tr_sfdr_a	[]	No	
tr_Zlo_Rp	1e3	Yes	RVD transducer low-side impedance 1D table. Note this is
tr_Zlo_Cp	1e-15	Yes	related to loading correction and it has effect only for RVD
tr_Zlo_f	[]	No	transducer and will work only if adc_Yin is defined as well.
tr_Zca_Rs	1e-9	Yes	Loading corrections: Transducer high side terminal series
tr_Zca_Ls	1e-12	Yes	impedance 1D table.
tr_Zca_f	[]	No	
tr_Zcal_Rs	1e-9	Yes	Loading corrections: Transducer low side terminal series
tr_Zcal_Ls	1e-12	Yes	impedance 1D table.
tr_Zcal_f	[]	No	
tr_Yca_Cp	1e-15	Yes	Loading corrections: Transducer output terminals shunting
tr_Yca_D	1e-12	Yes	impedance.
tr_Yca_f	[]	No	
tr_Zcam	1e-12	Yes	Loading corrections: Transducer output terminals mutual
tr_Zcam_f	[]	No	inductance 1D table.
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 32: List of output quantities of the TWM-Flicker wrapper.

Name	Uncertainty	Description
Pst	Yes	Short-term flicker severity.
Pinst	No	Instantaneous flicker sensation.

Table 33: List of "calcset" options supported by the TWM-Flicker wrapper.

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

8.2 Algorithm description

The structure of the TWM-Flicker algorithm wrapper is shown in fig. 21. The wrapper first applies correction to scaled the input signal y to actual measured level. The scaling is simplistic. The user defined frequency f_line with tolerance 2 Hz is assumed to be dominant component of the input signal y. Thus, the gain correction of digitizer, aperture error gain correction and transducer gain correction are

obtained for the *f_line* only. Resulting combined gain correction is applied to the time domain signal y. The wrapper also applies DC gain correction despite the main QWTB wrapper "flicker_sim" which does the flicker calculation is not using it.

After the signal is scaled, the wrapper calls the main QWTB algorithm "flicker_sim" to evaluate the flicker parameters.

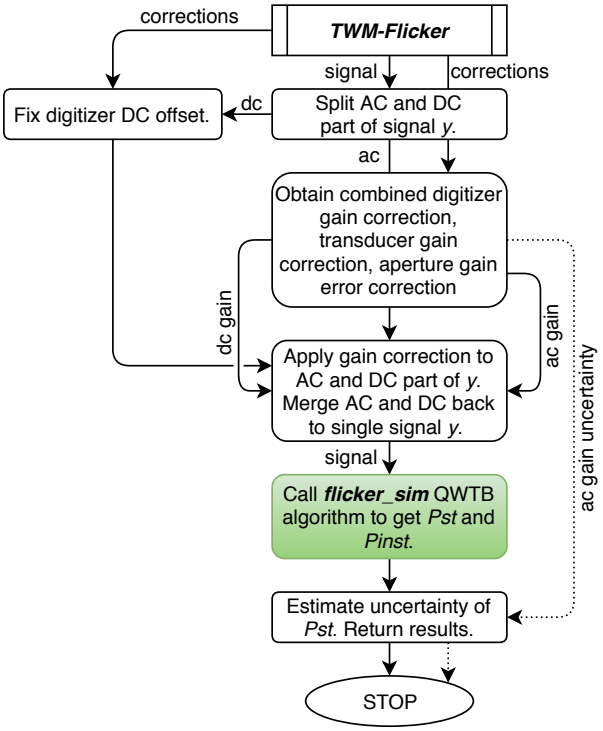

Figure 21: TWM-flicker algorithm wrapper diagram. The green blocks are calls to another QWTB wrappers.

8.2.1 QWTB algorithm wrapper "flicker_sim"

The core of the flicker algorithm is QWTB wrapper "flicker_sim". It calculates the flicker using a function "flicker_sim(u, fs, f_line, ...)". In general the algorithm calculates according to the block diagram shown in fig. 23.

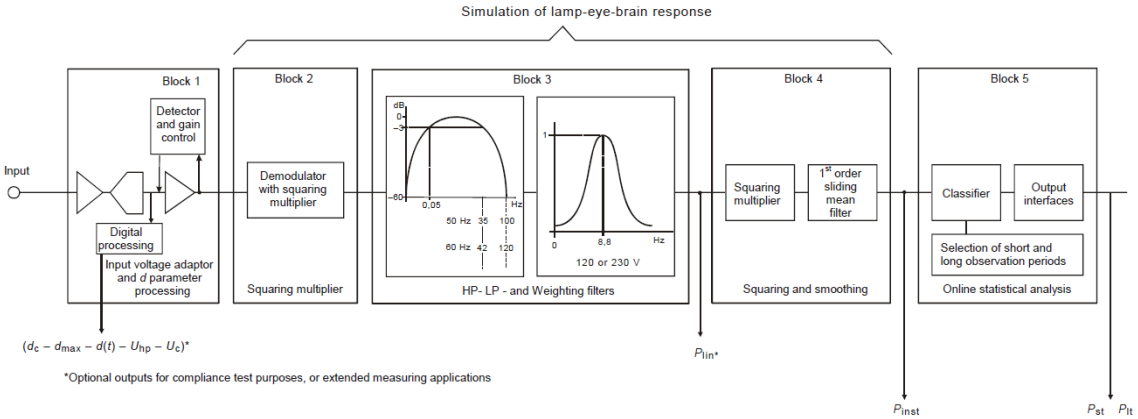

Figure 22: Flicker calculation block diagram according IEC 61000-4-15 [3].

The algorithm starts by checking of the input sampling rate. It will throw an error if the sampling rate is below 7 kHz. If the sampling rate is higher than 23 kHz, the algorithm will perform downsampling to a sampling rate near 17 kHz, which was empirically identified as optimal for the rest of the algorithm. Follows removal of the DC component and calculation of the RMS level of the whole signal u, which is used just for determination of the 120 V or 230 V systems.

Next step half-cycle RMS envelope calculation. The signal u is first passed via narrow passband filter (1st order Butterworth with passband 50 to 60 Hz). The filtered, theoretically noise-free signal is used for the zero-crossing detection. Next, RMS value of each half-cycle is calculated, so the u_half_rms envelope is calculated. This is input for the main flicker calculation as shown in fig. 23.

Following description of the flicker algorithm blocks is direct citation of report [1]:

Block 1 is the voltage adapter that scales the input mains frequency voltage to an internal reference level. Flicker measurements can be made independently of the actual input voltage level by this way.

Block 2 is the squaring multiplier that recovers the voltage fluctuation by squaring the input voltage signal. This block is simulating the behaviour of a lamp. Block 3 contains two sections. First section is composed of a cascade of two filters, a low-pass type and a high-pass type. Low-pass filter eliminates the double mains frequency ripple components in the signal. High-pass filter eliminates any DC voltage components in the signal. Second section is a weighting filter that simulates the frequency response of the human visual system to sinusoidal voltage fluctuations of a coiled filament gas-filled lamp (60 W/230 V or 60 W/120 V). Block 4 contains a squaring multiplier and a low-pass filter.

Combination of Block 2, Block 3 and Block 4 composes a non-linear system that simulates flicker signal applied to a lamp and human eye-brain response to this light. Output of the Block 4 is the instantaneous flicker severity P_inst.

Block 5 is the statistical analysis block that contains two sections. First section forms a cumulative probability function and second section forms a flicker level classifier. After the proper statistical evaluation of the P_inst values for 10 minutes observation, short-term flicker value P_st is generated by this block.

8.3 Uncertainty estimator

The uncertainty estimation is performed when the *calcset.unc* ='guf'. The estimation is performed at the TWM-Flicker algorithm wrapper level as the uncertainty comes partially from the gain uncertainty and timebase error uncertainty. However, it was found the effect of the typical gain and frequency uncertainties is so low, it is not even necessary to include their effect, because the error of the algorithm itself is orders of magnitude higher. So the uncertainty of this algorithm was estimated to fixed 0.9% of the Pst value for level of confidence 95%.

8.4 Validation

Validation of the algorithm was performed using a simulator function "verify_flicker_sim()" present on the "flicker_sim" QWTB wrapper folder. This function automatically performs series of the tests according different versions of IEC 61000-4-15 [3]. The test was run for various sampling rates to verify the algorithm works in full range of sampling rates. Example of the results is shown in the fig. 23.

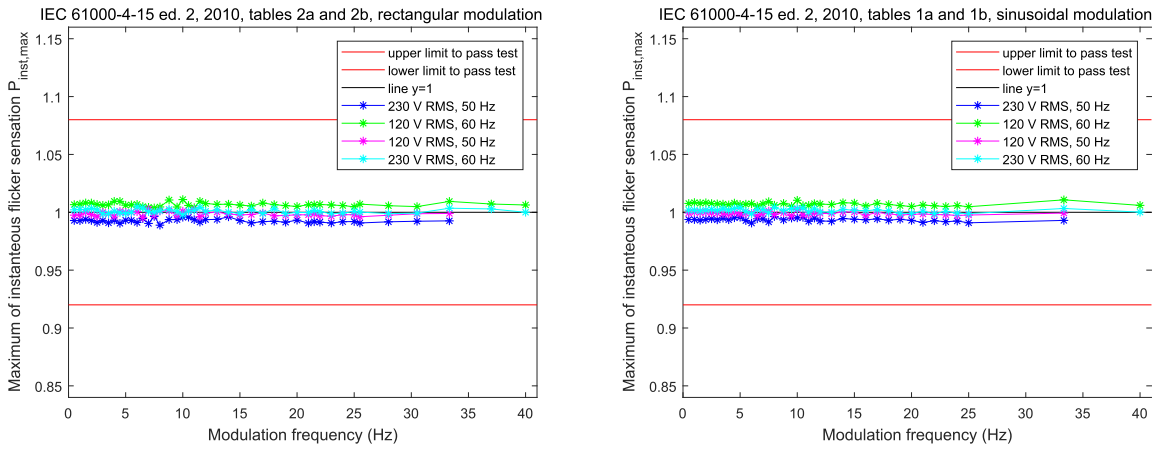

Figure 23: Example of flicker algorithm "flicker_sim" validation against IEC 61000-4-15 [3], edition 2, for sampling rate $fs = 50\,\text{kHz}$.

References

[1] Report A2.3.2: Developing a Matlab script file for Flicker calculation algorithm. Empir activity completion report, TÜBİTAK Ulusal Metroloji Enstitüsü, January 2018.

[2] Solcept AG. Solcept Open Source Flicker Measurement-Simulator. `https://www.solcept.ch/en/tools/flickersim/`.

[3] Electromagnetic compatibility (EMC), Testing and measurement techniques, Flickermeter. Standard, August 2010.

[4] Wilhelm Mombauer. *Messung von Spannungsschwankungen und Flickern mit dem IEC-Flickermeter*. VDE-Verlag.

[5] NPL. NPL Reference Flickermeter Design . `http://www.npl.co.uk/electromagnetics/electrical-measurement/products-and-services/npl-reference-flickermeter-design`, 2007.