The EMPIR initiative is co-funded by the European Union's Horizon 2020 research and innovation programme and the EMPIR Participating States

17RPT03 - DIG-AC

A DIGITAL TRACEABILITY CHAIN FOR AC VOLTAGE AND CURRENT

Deliverable 4

Report on the integrated software for data processing and uncertainty estimation of dynamic measurements, including related methods and algorithms

Leader
CMI

Contributors
IPQ, CEM, NPL

Due date
1. 2022

Delivered
31. 5. 2022

This project 17RPT03 DIG-AC has received funding from the EMPIR programme cofinanced by the Participating States and from the European Union's Horizon 2020 research and innovation programme.

Contents

Chapter 1

Introduction

This document focuses on the errors and uncertainties of quantity estimation algorithms and propagation of uncertainties through algorithms. It provides description of basic problems, shows several examples and discussion on properties of selected algorithms.

Basic theory on uncertainties is described in *Guide to the Expression of Uncertainty in Measurement* [1]. This approach is called *GUF*. Due to limitations, other approaches were studied. A Monte Carlo method (*MCM*) was summarized in [2], [3]. Methods presented in this document utilize both approaches.

Chapter 2 describes errors and uncertainties of any quantity estimation algorithm. First, general theory is presented, next, application to an actual algorithm is discussed. Chapter 3 discuss existing software QWTB and TWM, and also describes a newly developed software QWTBvar including its interface, usage and presents two examples. Chapter 7 presents the results from uncertainty estimation and validation of algorithm for estimation of Spurious Free Dynamic Range (SFDR). Chapter 8 presents the results from uncertainty estimation and validation of algorithm for estimation of Integral and Differential non-linearity (INL-DNL).

This document fulfills deliverable 4 (Activity 3.4.6) of the EMPIR project 17RPT03 - DIG-AC, *A digital traceability chain for ac voltage and current*. Chapter 3 describes the newly developed software with its inner structure, presented examples serves as a handbook of utilisation, and source code is presented in public repository [4].

Chapter 2

Errors and uncertainties of an quantity estimating algorithm

2.1 General description

Output quantities O_i are related to input quantities I_i by a general function:

$$(O_1, \ldots, O_n) = f(I_1, \ldots, I_m) \; . \tag{2.1}$$

An algorithm A estimates output quantities O'_i based on the input quantities I_i.

$$(O'_1, \ldots, O'_n) = A(I_1, \ldots, I_m) \; . \tag{2.2}$$

Algorithm adds unknown error ε_A. The algorithm error is a function of input quantities:

$$\varepsilon_A = f(I_1, \ldots, I_m) \; . \tag{2.3}$$

The algorithm error manifests as a bias of the output quantities and the estimated quantities differ from the true value of output quantities:

$$O'_i = O_i + \varepsilon_{A,i} \; . \tag{2.4}$$

As an example, one can take the Discrete Fourier Transformation (DFT) algorithm used for estimation of signal amplitude and phase from sampled record. A digitizer samples an AC waveform and record is obtained. DFT is applied to the signal and amplitude and phase of the main component is obtained. The input quantities are the record itself, the quantization properties of the digitizer, measurement coherency, noise, amplitudes, phases and frequencies of interference signals etc. The output quantities are the amplitude and phase of the main signal component. In the case of coherent measurement, record of infinite length and zero quantization error, without noise and interference, one can obtain true values of output quantities O_i. However, for the case

of non-coherent measurement, the DFT algorithm will result in O'_i estimates burdened by error ε_i.

The description can be extended to a case with uncertainties. Every input quantity has got an unknown error ε_{I_i}, that is represented by uncertainty of the input quantity $u(I_i)$. In calculations, the uncertainty is often represented by a probability distribution function (PDF) $g_{I_i}(\xi_i)$. Uncertainties of output quantities are related to input quantities and its uncertainties:

$$(u(O_1), \ldots, u(O_n)) = f(I_1, \ldots, I_m, u(I_1), \ldots, u(I_m)) \; . \tag{2.5}$$

The uncertainty of output quantities calculated by algorithm A is a function of input quantities, its uncertainties, and algorithm error:

$$(u(O'_1), \ldots, u(O'_n)) = A(I_1, \ldots, I_m, u(I_1), \ldots, u(I_m)) \; . \tag{2.6}$$

Because of the PDFs of input quantities and equation 2.3, the error of the algorithm will variate according to PDF $g_\varepsilon(\xi)$.

The error of the algorithm can be obtained exactly only if both input and output quantities and uncertainties are known. Using A, one can calculate O'_i from I_i and obtain ε. Another possibility is to know the inverse function f^{-1}. Using the inverse function, input quantities can be obtained using following equation:

$$(I_1, \ldots, I_m) = f^{-1}(O_1, \ldots, O_n) \; , \tag{2.7}$$

and again O'_i can be obtained using A and ε can be evaluated.

For the case of DFT, a knowledge of both signal and true value of amplitude would show out the error of the amplitude estimation.

2.2 Case of a real measurement

In the case of a real measurement, neither the true values of all input quantities X_i nor the true values of output quantities Y_i are known and the value of the error ε cannot be obtained.

The algorithm error can be estimated for another set of input and output quantities $\tilde{X}_i$, $\tilde{Y}_i$, that are near to the measured quantities. It is based on the assumption that the error of the algorithm is changing linearly and only a little with little change of quantities.

Thus, first output quantities Y_i are calculated using A and measured values X_i. Next, the values of the $\tilde{Y}_i$ are selected to be as near as possible to Y_i. Using A^{-1}, the input quantities $\tilde{X}_i$ are obtained, and using A the error of the algorithm $\tilde{\varepsilon}$ is calculated. Based on the assumption one can suppose that:

$$\varepsilon \approx \tilde{\varepsilon} \; . \tag{2.8}$$

The method is shown in figure 2.1. The case can be more complex due to required input quantities. It can be required to estimate multiple quantities not required by simple application of A.

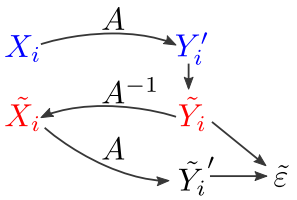


Figure 2.1: Method for estimation of algorithm error. Quantities in blue are obtained by measurement. Quantities in red are simulated.

In the example presented before, the measured signal and sampling frequency is used as inputs for DFT algorithm (A). The output quantities are amplitudes and phases of the signal components. To reconstruct the signal, also level of noise have to be estimated. These parameters can be used to construct a simulated sampled signal (A^{-1}) with known properties. Algorithm is applied to the simulated signal and output quantities obtained. The error $\tilde{\varepsilon}$ is calculated and using the approximation in equation 2.8 is used to obtain error of the algorithm for real measurement.

2.3 Pre-calculation of uncertainties

The described method can be used to estimate algorithm errors and algorithm uncertainties. The method is time consuming, especially if the Monte Carlo method has to be used. The issue can be solved by pre-calculating algorithm errors and by propagating uncertainties for a predefined values of input and output quantities and uncertainties.

For a selected number of points in the phase space of $\tilde{Y}_i$, the algorithm's error uncertainties can be estimated using the method described previously. This leads to a lookup table $Y_i' \rightarrow u(Y_i')$. For the case of actual measurement, the uncertainties $u(O_i)$ can be estimated using interpolation of the lookup table. The whole method is represented in figure 2.2.

2.4 Overall plan

1. Select ranges for all input quantities and its uncertainties.
2. Calculate output quantities using A.
3. Plot output quantities and uncertainties on values of input quantities and uncertainties.

4. Find out most influential input quantities and uncertainties.
5. Select ranges and spacing of most influential output quantities and uncertainties.
6. For all variations of output quantities in selected range, construct input quantities according to A^{-1}.
7. Calculate output quantities and uncertainties in the selected range for selected spacing.
8. Make lookup table for fast calculation of uncertainties of output quantities.

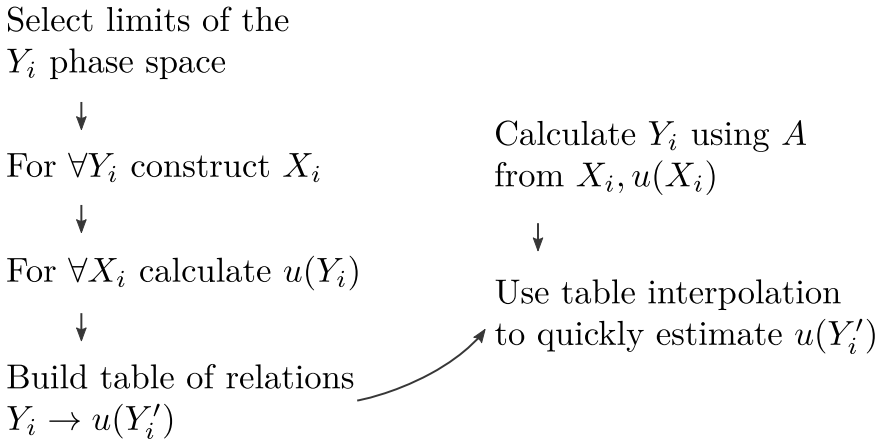


Figure 2.2: Method for pre-calculation of uncertainties

Chapter 3

Software for calculation and propagation of uncertainties

3.1 Q-Wave Toolbox

A common situation in the data processing of sampled signal is the estimation of multiple quantities using the same record. The user is interested in the amplitude and the phase of the main signal component, in a spectrum and stability of these quantities during multiple records. For the case of evaluating a properties of a digitizer, spurious free dynamic ratio (SFDR), total harmonic distortion (THD) and effective number of bits (ENOB) are important quantities. Algorithms exists for all of these quantities, but it is a complex task to learn how to use every single algorithm.

Q-Wave toolbox (QWTB) can help with this situation. It is a software toolbox written in M-code and is running in *Matlab* [5] or *GNU Octave* [6]. It aims for aggregation of high-quality algorithms required for data processing of sampled measurements. QWTB consist of data processing algorithms from different sources, unifying application interface and graphical user interface.

The toolbox gives the possibility to use different data processing algorithms with one set of data and removes the need to reformat data for every particular algorithm. Toolbox is extensible.

3.2 TracePQM wattmeter

The QWTB was designed to help using general quantity estimating algorithm. However, it was not tailored for actual metrological measurements. Therefore, during development of TWM (TracePQM Wattmeter) an extension of the QWTB interface was formulated.

TWM is a transparent, metrology grade measurement system for traceable measurement of Power and Power Quality (PQ) parameters. It is designed to allow recording

of voltage and current waveforms using various digitizers and processing the measured waveforms using any algorithm.

TWM defined name space for quantities needed for transducers, errors of connecting transducers to digitizers. During the TracePQM project [7], new versions of algorithms were developed capable of using the defined quantities. The core of TWM relies on QWTB.

3.3 QWTB variator

The estimation of algorithm errors was not solved successfully in QWTB nor in the TWM extension. Therefore, a Q-Wave toolbox variator *QWTBvar* was developed. It is a system that can:

- variate input quantities or its uncertainties,
- calculate errors of output quantities to the nominal values,
- plot dependence of output quantities on the varied input quantities or its uncertainties,
- create lookup table of uncertainties of output quantities,
- interpolate the lookup table for quick estimation of uncertainties.

The application interface is described in chapter 4.

Chapter 4

QWTBvar interface

QWTBvar is a *Matlab* [5]/*GNU Octave* [6] script with following use.

4.1 Variation of quantities

```
[jobfn] = qwtbvar(algid, datain, datainvar, calcset)
```

Variates inputs `datain` according to `datainvar` and applies them one by one into QWTB with settings `calcset`. Returns path to the calculation plan `jobfn`.

```
[jobfn] = qwtbvar(jobfn)
```

Continues interrupted calculation according to calculation plan `jobfn`.

```
[H] = qwtbvar(jobfn, varx, vary)
[H, x, y] = qwtbvar(jobfn, varx, vary)
```

Plots 2D figure `vary` vs `varx` with uncertainties for both axes, if available, using results of calculation described in `jobfn`.

4.2 Plotting of results

```
[H] = qwtbvar(jobfn, varx, vary, varz)
[H, x, y, z] = qwtbvar(jobfn, varx, vary, varz)
```

Plots `varz` versus `varx` and `vary` from results of calculation described in `jobfn`.

```
[x, y] = qwtbvar(jobfn, varx, vary)
[x, y, z] = qwtbvar(jobfn, varx, vary, varz)
```

Returns only data, the plot is not generated.

4.3 Generation of Look Up Table

```
lut = qwtbvar('lut', jobfn, ax_set_lut, rqset_lut)
```

Generates Look Up Table (LUT) based on the calculated results in `jobfn` using settings of LUT axes `ax_set_lut` and settings of result quantities `rq_lut`. The `lut` is a standalone structure with all required data.

4.4 Interpolating Look Up Table

```
unc = qwtbvar('interp', lutfn, axip);
```

Interpolates LUT from file `lutfn` at a point `axip`.

4.5 Input variables

- `algid` – id of the algorithm, as in QWTB.
- `datain` – input data, the same as described in QWTB.
- `datainvar` – input data that will be varied, see below.
- `calcset` – calculation settings, the same as described in QWTB, with additional structure `.var.*`, see below.
- `jobfn` – path and name of a file containing description of the calculation.
- `varx`,`vary`,`varz` – input or output quantities. Strings should contain name of input or output quantity and optionally the field, e.g.`x`,`y`,`x.v`,`y.v`,`y.u` etc.

4.6 Output variables

- `jobfn` – path and name of a file containing description of the calculation
- `H` – handle to the 2D/3D figure
- `x`, `y`, `z` – vectors of the requested values used for plot.

4.7 Structure Varied data

This structure should contain only quantities (Q) with fields (f) that will be varied. $Q.f$ in `datainvar` should have size of one dimension larger than $Q.f$ in `datain`, and this dimension in `datain` must be one.

Examples:

- Scalar:

```
datain.x.v = 6
datainvar.x.v = [6 7]
```

  Sizes of dimensions of `x.v` in `datain` are: `[1 1]`, and in `datainvar` are `[1 2]`.

- Vector:

```
datain.x.v = [6 6]
datainvar.x.v = [6 6; 7 7]
```

  Sizes of dimensions of `x.v` in `datain` are: `[1 2]`, and in `datainvar` are `[2 2]`.

- Matrix

```
datain.x.v = [6 6; 6 6]
datainvar.x.v = cat(3, [6 6; 6 6], [7 7; 7 7])
```

  Sizes of dimensions of `x.v` in `datain` are: `[2 2 1]`, and in `datainvar` are `[2 2 2]`. The following is also valid:

```
datain.x.v = [6 6; 6 6]
datainvar.x.v = cat(5, [6 6; 6 6], [7 7; 7 7])
```

  Sizes are `[2 2 1 1 1]` and `[2 2 1 1 2]`;

4.8 Structure Calculation settings

As described in QWTB, it can contain additional optional fields:

- `.var` – that is a structure with following optional fields (nominal value):
  - `.dir` – (`.`) directory for variation jobs and results.
  - `.fnprefix` – (`''`) filename prefix for variation jobs and results.
  - `.cleanfiles` (0) if 1, delete old jobs and results with colliding filenames during preparation of calculation (but not during continuation)
  - `.smalloutput` – (1) large data of quantities in datain and dataout are not saved. This affects fields `.c` (correlation matrix) and `.r` (randomized values).
  - `method` – (`singlecore`) one of: `singlecore`, `multicore`, `multistaion`.
  - `procno` – (1)
  - `chunks_per_proc` – (1) number of calculation jobs for one process

4.9 User function

Input variable `algid` does not have to be full QWTB algorithm but can be a user function. The function must have two inputs: `datain` and `calculation_settings`, and three outputs: `dataout`, `datain` and `calculation_settings`.

Example of user function working with quantity Q:

```
function [dataout, datain, cs] = userfunction(datain, cs)
    dataout.x.v = 2.*datain.x.v;
end
```

4.10 Documentation of inner structure

The QWTBvar is internally fully documented inside of the script itself. The overview of the inner structure is shown in figure 4.1.

4.11 Examples of QWTBvar use

Chapter 5 shows how to use the QWTBvar for comparison of two algorithms.

Chapter 6 shows the use of QWTBvar for variation of input quantities, generation of LUT and interpolating the output uncertainties.

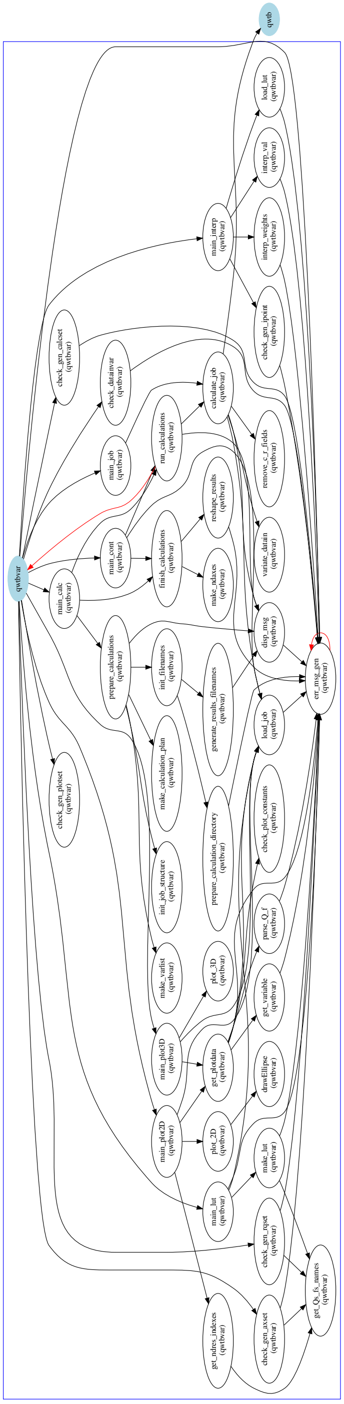


Figure 4.1: Inner structure of the QWTBvar. The blue ovals represents script files, white ovals represents sub functions, black arrows represents function calls, red arrow represents recursions. Blue rectangle encompass sub functions of the QWTBvar script.

Chapter 5

Comparison of two algorithms

5.1 Algorithms

Two algorithms have been selected:

- TWM-THDWFFT
- TWM-MFSF

The TWM-THDWFFT algorithm is designed for calculation of the harmonics and 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.

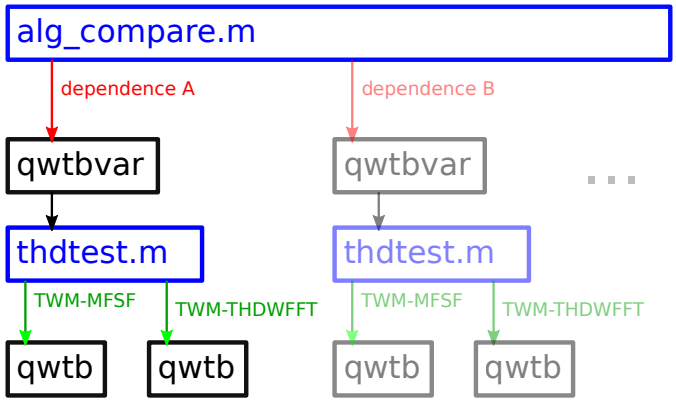
TWM-MFSF is an algorithm for estimating the frequency, amplitude, and phase of the fundamental and harmonic components in a waveform. Amplitudes and phases of harmonic components are adjusted to find minimal sum of squared differences between the sampled signal and the multi-harmonic model.

A detailed description of both algorithms can be found in TWM project documentation [8] and is copied for reference into this document, Annex F.1 and in F.2.

5.2 Method overview

First a simulated signal is constructed. Next both algorithms are used to calculate THD value using GUF (GUM uncertainty framework, [1]) and Monte Carlo [2] methods. Both results are plotted into figures.

The framework used to run the simulations is QWTB. Script `alg_compare.m` is used to set values and plot figures. For every dependence (e.g. THD on noise or THD on signal frequency) script calls function `qwtbvar` that is responsible for variation of inputs. `qwtbvar` calls the script `thdtest.m` that constructs a signal and calls `qwtb` to calculate results.



The scripts developed in this project are shown in blue and are listed in annexes A and B.

5.3 The testing signal

The following properties of the testing signal were used during comparison. Acquisition quantities:

- sampling frequency: 50 kHz;
- record length 100 kSa.
- resolution of the digitizer: 24 bit;

Signal quantities:

- frequency: 50.01 kHz,
- main signal component amplitude: 1 V,
- number of signal components: 5,
- amplitudes of harmonics 2 – 4: 0.01 V,
- signal components phases: 0 rad,
- offset: 0 V,
- standard deviation of noise: 10 µV,

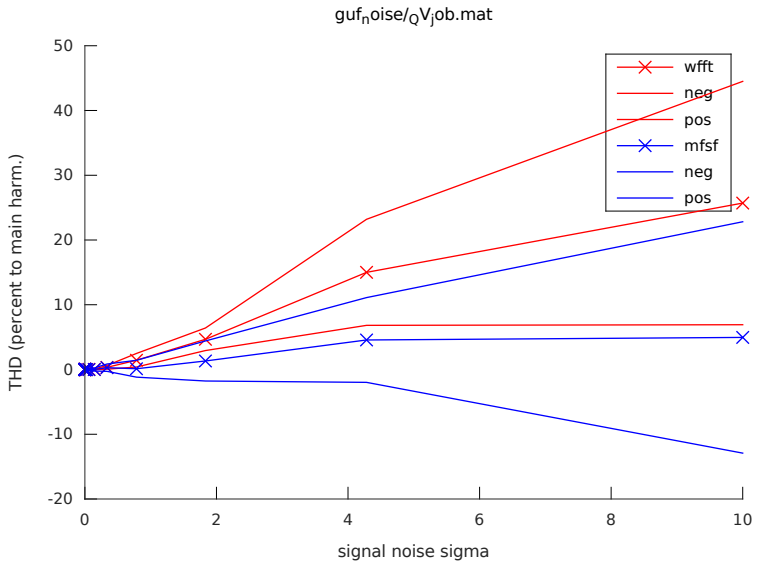
5.4 Comparison results

5.4.1 Influence of noise

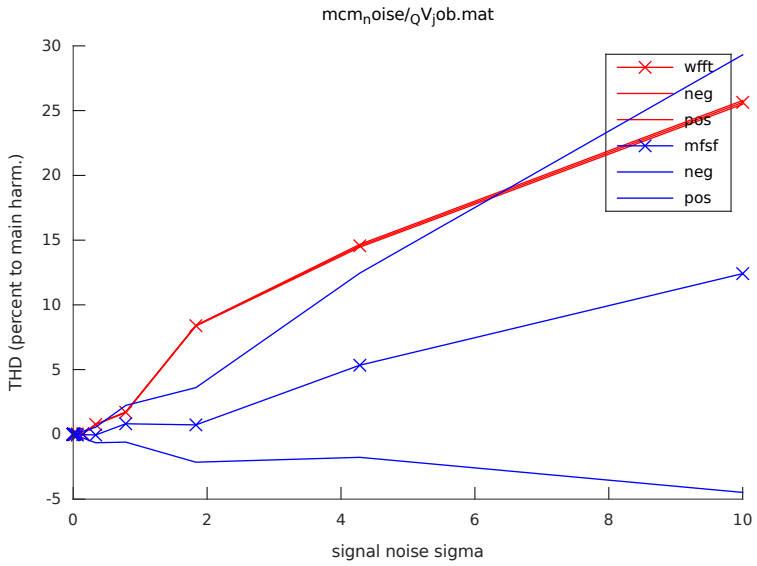
Following figures show out the dependence of the THD value on the sigma of the noise simulated in the signal. The first figure was generated for uncertainties calculated using GUF, the second figure with uncertainties calculated using MCM.

The value of THD calculated using THDFFT algorithm shows out a small offset, compared to the results of the MFSF algorithm. The uncertainties are mostly covering the error of the THD for both methods. However the WFFT algorithm does not implement MCM uncertainties correctly and only uncertainties calculated by GUF are relevant.

The uncertainties are affected by the noise and increased linearly with increasing noise, as can be expected.



guf_noise



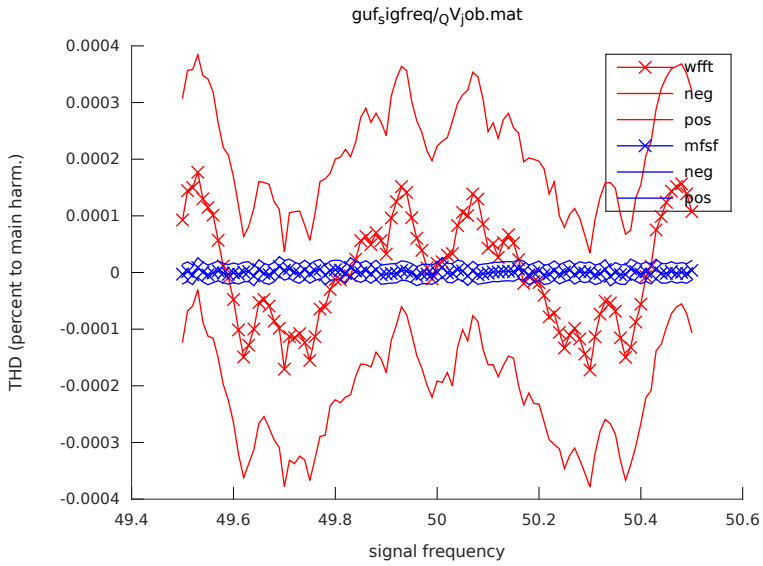
mcm_noise

5.4.2 Influence of signal frequency

Following figure shows out the dependence of THD value on the frequency of the main signal component with uncertainties calculated using GUF.

The value of THD calculated using THDFFT shows a variation on the signal frequency, that is to be expected due to principles of the implemented Discrete Fourier Transformation. The MFSF is not affected by signal frequency because of the implemented fitting method.

The uncertainties covers the THD errors for both methods.

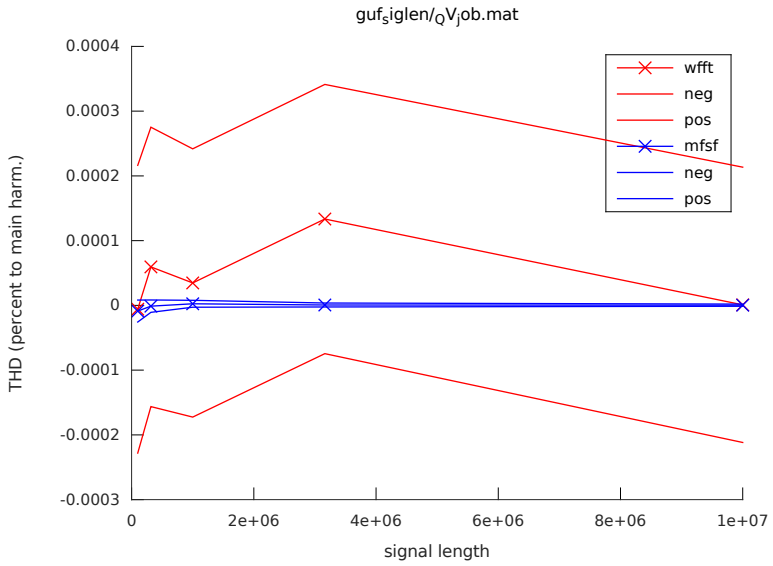


mcm_noise

5.4.3 Influence of signal length

Following figure shows out the dependence of THD value on the length of the record component with uncertainties calculated using GUF.

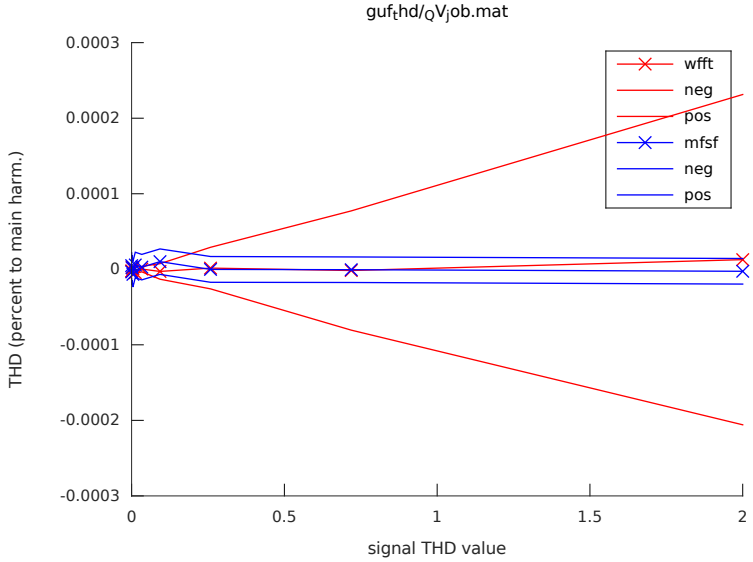
Again the value of THD error is smaller for MFSF method. The uncertainties covers the THD errors for both methods.



mcm_noise

5.4.4 Influence of THD

Following figure shows out the dependence of calculated THD value on the simulated THD value of the signal with uncertainties calculated using GUF. The errors are very small for both methods. The most interesting fact is zero or small dependence of MFSF uncertainty on the THD value.



mcm_noise

5.5 Comparison conclusion

The comparison showed out several things.

1. Both algorithms calculate GUF uncertainties correctly, i.e. the uncertainty is greater than the error of the algorithm, for at least 95 % of results.

2. The MFSF algorithm errors are much smaller than the WFFT algorithm ones.

3. For several cases the uncertainties of the MFSF algorithm are much smaller than the uncertainties of the WFFT algorithm.

Results validated the use of both algorithm, as described in item 1. As uncertainties are greater than the errors originated in the algorithm, the calculation can be considered as validated.

It seems that performance of MFSF algorithm is greater than WFFT one. However, the algorithms differ in one important point. The WFFT algorithm only requires a maximal harmonic to be evaluated. On the contrary, the MFSF algorithm requires the knowl-

edge of all harmonics present in the signal. In the case of incorrect input information the results will also be incorrect. This feature has yet to be properly evaluated.

Chapter 6

Example: Uncertainties of SFDR

The SFDR algorithm use Discrete Fourier Transformation with Blackmann window to calculate Spurious-Free Dynamic Range. The input to the algorithm is the sampled waveform and time vector, the output is the SFDR value, commonly expressed as the ratio of the main signal component to the highest spurious signal component in dBc units (decibel relative to carrier). The input uncertainties are the uncertainty of particular samples, and the uncertainty of time samples (or sampling frequency).

Estimation of the uncertainties by the SFDR algorithm takes a lot of time. The pre-calculation of the uncertainties and storing them into a Look Up Table (LUT) is a way to greatly speed up the process.

Script `make_lut.m` (see appendix C) shows an example of the process that is divided into following steps:

1. Preparation of waveform generator.
2. Selecting values of quantities.
3. Calculation of uncertainty propagation through algorithm.
4. Generation of LUT and adding into the algorithm.
5. Validation.

Uncertainty of SFDR is a value depending on the following quantities:

- N_p: number of main signal periods sampled in the record,
- ssr: ratio of sampling to signal frequency,
- A: Amplitude of the main signal,
- $jitter$: jitter of the samples,

- $noise$: standard deviation of the noise in the signal,
- $SFDR$: actual value of SFDR,
- smr: spurious to main signal frequency multiple.

Therefore the LUT table will be a n-dimensional matrix of results, where every quantity will form one dimension of the matrix.

6.1 Preparation of waveform generator

A script, that can automatically generate a waveform used for calculation of SFDR ha to be prepared. An example of the script `alg_generator.m` is in appendix D. Input quantities to the generator script, apart from quantities already defined as important for SFDR uncertainty (6), are:

- f: main signal component frequency,
- O: main signal offset,
- ph: main signal phase,
- dc: whole signal direct current level.

The script simply use the quantities, randomize values according to the uncertainties, and generates a waveform. The uncertainty of time of the samples is based on the jitter. The voltage uncertainty of the samples is based on the noise present in the signal (i.e. combined noise of both signal and digitizer).

6.2 Selecting values of quantities

The uncertainties in a LUT table can be defined only for a limited range of quantities. Values selected using typical measurement data are listed in table 6.1.

6.3 Calculation of uncertainty propagation through algorithm

The script `make_lut.m` (appendix C) is built in the following way. First, calculation settings are created in variable `CS_lut`. Nominal values of quantities for the generator are set in variable `DI_gen` and varied values in the specified ranges are set in variable `DI_gen_var`. Next, the variator is run using the line:

Table 6.1: Ranges of quantities used for LUT.

N_p	1	100	
ssr	20	100	
A	0.001	1000	V
$jitter$	10^{-13}	10^{-10}	s
$noise$	10^{-8}	10^{-5}	V
$SFDR$	1	$1e8$	V V^{-1}

```
jobfn = qwtbvar('calc', 'gen_and_calc', DI_gen, DI_gen_var,
    CS_lut);
```

where `gen_and_calc` is a function that asks QWTB to generate the waveform, and immediately asks QWTB to calculate SFDR (appendix E).

The QWTBVAR will variate `DI_gen` quantities, generate waveform, calculate SFDR and store results into files.

6.4 Generation of LUT and adding into the algorithm

The next step is to create a Look Up Table. The LUT will be a matrix with 6 dimensions, each dimension is one varied quantity, in the range specified before. First, the dimensions of the LUT has to be more specific. QWTBVAR has to know what to do if the user asks for a value outside the ranges of the LUT. Either, an error can be generated, or the output value can be kept at boundary values. In the example, the variable `ax_set_lut` provides detailed specifications of LUT dimensions. E.g. if the user asks for a value with jitter smaller than the range of calculations, the `ax_set_lut` specifies that the resulting value will be the same as user would ask for the lower limit of the jitter range. Variable `ax_set_lut` contains information about all varied values, i.e. all dimensions of the LUT.

Variable `rqset_lut` defines the scale for output quantity, that is uncertainty of SFDR. It is defined as a logarithmic scale, this improves the interpolation.

LUT is generated by the following command:

```
lut = qwtbvar('lut', jobfn, ax_set_lut, rqset_lut);
```

Next, the variable `lut` is saved into a file in the algorithm `alg_SFDR` directory.

The LUT has to be integrated into the algorithm. The algorithm has to estimate not only SFDR, but also all the quantities used in the LUT. This is done in the script `alg_wrapper` (appendix F). An estimation of the signal and spurious frequency is easy, it is already an output of the DFT used in the algorithm, thus, quantities N_p, ssr, A and

$SFDR$ are already estimated. The jitter is defined by the input quantity t. The signal and ADC noise can be also estimated using the DFT by calculating noise floor. By obtaining all quantities needed for LUT, the output uncertainty of SFDR can be interpolated using the following command:

```
unc = qwtbvar('interp', lutfn, axip);
```

The variable `lutfn` is a path to the file with stored LUT, and the variable `axip` contains values of all the quantities that are dimensions of the LUT, i.e. the varied quantities during calculations for the LUT.

Chapter 7

SFDR algorithm validation and uncertainty estimation

The SFDR algorithm was already presented in chapter 6. SFDR value is defined by the expression:

$$\text{SFDR} = 20\log(A_s/A_0)$$

where A_s is the amplitude of the spurious component and A_0 is the amplitude of the fundamental component of the signal.

An algorithm process validation is described in this chapter to verify and quantify any systematic error introduced by the SFDR algorithm, its dependence on input quantities values and the application of the qwtb.m function to estimate the uncertainty of the SFDR output value as a function of the uncertainty variation of the input quantity introducing disturbances such as noise.

The validation process is based on the application of the algorithm to a simulated sampled data generated from a sine wave signal.

The algorithm error is calculated as the difference between the SFDR value calculated by the algorithm and the theoretical value calculated from the simulated input signal applied. Error values presented in following sections are in relative units:

(SFDR algorithm output - SFDR calculated value) / SFDR calculated value

The SFDR algorithm was run using the qwtb.m and qwtbvar.m functions in QWTB framework.

First, a set of simulations were conducted to observe the dependence of the algorithm error as a function of input quantities values: frequency of the fundamental component, frequency of spurious components, spurious amplitude related to amplitude of fundamental and random noise in sampled input values.

Then, an amount of uncertainty was added to the sampled values and the effect in the algorithm output value and the related uncertainty value estimated by the qwtb.m

function (using Monte Carlo method) were observed. These results were compared with the SFDR algorithm error and its standard deviation obtained in previous tests.

Finally, the algorithm was applied to a simulated non-coherently sampled signal, and its error was calculated as function of the input quantities values.

The simulated signal used has the following parameters values:

- Signal quantities:
  - main signal component amplitude: 1 V,
  - amplitude offset: 0 V,
  - number of signal components: 2 (fundamental and one spurious component),
  - signal components phases: 0 rad,
  - frequency of fundamental, f_0, tested: 100 Hz to 1 kHz (steps of 100 Hz),
  - spurious component frequencies tested (multiple of f_0): 0.5, 1.1, 1.5, 2.5, 3.5 and 4.5,
  - ratio of the spurious component amplitude related to the amplitude of the fundamental: −40 dB, −80 dB, −120 dB and −140 dB.
- Acquisition quantities:
  - sampling frequency: 10 kHz,
  - record length: 10 kSa,
  - ADC bit resolution: 28 bit (input of `qwtb.m`)

The above sampling frequency value of 10 kHz assures that the highest frequency component tested, corresponding to a spurious component of $4.5 f_0$ and with $f_0 = 1$ kHz, still remains below the Nyquist frequency (5 kHz).

With the value selected for the record length of 10 kSa and with the 10 kHz for the sampling frequency, the minimum number of sampled periods presented in the generated testing signals is 100 (for the test signal with $f_0 = 100$ Hz).

7.1 Algorithm error dependence on input quantities

7.1.1 Algorithm error dependence on frequency, spurious component and SFDR value

SFDR algorithm was run (script `SFDR_test.m` in appendix G) with input sampled data defined above and for each one of the following signal parameter values combinations:

- Frequency values of the fundamental component (f_0): 100 to 1000 Hz, with 100 Hz step,
- Spurious frequency (multiple of f_s): 0.5, 1.1, 1.5, 2.5, 3.5 and 4.5,
- SFDR (relative to carrier): −40 dB, −80 dB, −120 dB and −140 dB.

The calculated relative error of the SFDR values obtained by the algorithm was between 0 and the maximum value found of 6×10^{-9} (test signal with f_0 = 1 kHz, f_s = 0.5xf_0 and SFDR = −140 dB). This means that, for a coherently sampled signal and with the acquisition quantities used (sample frequency, number of samples) and in the range of input quantities tested, it was possible to confirm that the SFDR algorithm does not input any relevant systematic error in the calculation of the SFDR value, being this limited to a maximum relative value of 6×10^{-9}.

7.1.2 Algorithm error dependence on noise value

Random noise was added into the simulated signals of 100 Hz and 1 kHz, with fixed values of SFDR = −80 dB (relative to carrier) and $f_s = 1.5 \cdot f_0$. The noise amplitude values added to each sample value of the test signal were 1×10^{-6} V, 1×10^{-5} V and 1×10^{-4} V.

The algorithm was run 1000 times for each noise value amplitude (script `SFDR_repeat_test.m` in appendix H).

The calculated mean value of the relative SFDR error and its standard deviation, maximum and minimum values are presented in the following table:

Relative Noise amplitude	f_0	SFDR relative error			
		Mean value	Standard deviation	Maximum	Minimum
1×10^{-6}	100 Hz	-3×10^{-7}	2×10^{-5}	7×10^{-5}	-7×10^{-5}
1×10^{-5}		-3×10^{-6}	2×10^{-4}	6×10^{-4}	-6×10^{-4}
1×10^{-4}		2×10^{-5}	2×10^{-3}	7×10^{-3}	-7×10^{-3}
1×10^{-6}	1 kHz	-4×10^{-7}	2×10^{-5}	7×10^{-5}	-6×10^{-5}
1×10^{-5}		7×10^{-6}	2×10^{-4}	6×10^{-4}	-6×10^{-4}
1×10^{-4}		4×10^{-5}	2×10^{-3}	7×10^{-3}	-6×10^{-3}

The presence of noise in signal originates the variation of the algorithm output value with larger scattering than the noise value, as it can be seen by the maximum and minimum values obtained from 1000 runs of the algorithm and from the standard deviation values which are one order of magnitude greater than the noise value.

The mean values of the SFDR relative error obtained are around 2 orders of magnitude lower than the corresponding standard deviation values.

The algorithm was also run with a rectangular window instead of the default Blackman window. The selection of the window was done in the alg_wrapper by changing the window variable:

```
datain.window.v = 'rectangular'
```

The results obtained from the standard deviation are between 21 % and 27 % lower which confirms that the use of windows in coherent sampling will increase the influence of noise in the standard deviation of the FFT results [9]. Figure 7.1 shows results for 1 kHz and 1×10^{-6} V of noise amplitude.

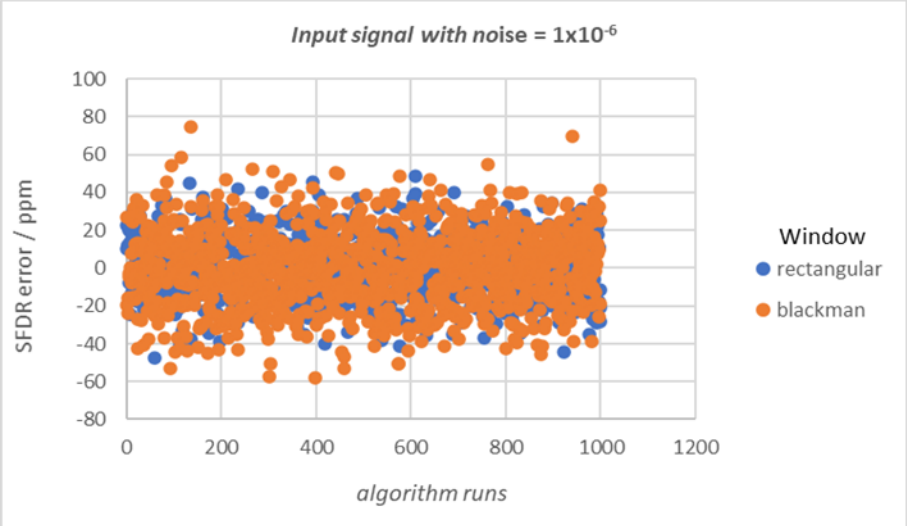


Figure 7.1: SFDR error of the algorithm output from a simulated signal of 1 kHz, SFDR −80 dB, spurious component at $1.5 \cdot f_0$ and sampled values with 1×10^{-6} V of random noise.

7.2 Algorithm error and uncertainty dependence on uncertainty in each sampled value

For each sampled value of the simulated signal (with SFDR −80 dB (relative to carrier) and $f_s = 1.5 \cdot f_0$) an uncertainty with values of 1×10^{-6} V, 1×10^{-5} V and 1×10^{-4} V was introduced .

For each one of these uncertainty values, the algorithm was run (script `SFDR_unc_test.m` in appendix I) 3 times and the results obtained are represented in figures 7.2 to 7.4.

Error bars represent the uncertainty calculated by the QWTB from the SFDR estimation. These uncertainty values are within the same order of magnitude of the standard

deviation values obtained in 7.1.2 and confirm the correct work of the uncertainty estimation by qwtb.m function.

Furthermore, uncertainty values cover the residual relative error of the SFDR value estimation.

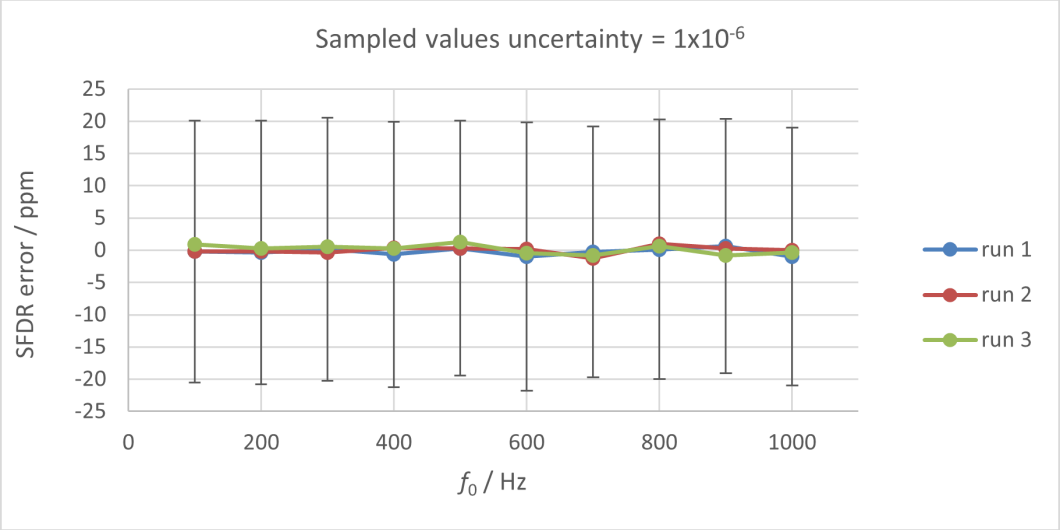


Figure 7.2: SFDR error of the algorithm output from a simulated signal (SFDR −80 dB and $f_s = 1.5 \cdot f_0$) with sampled values relative uncertainty of 1×10^{-6} V as a function of frequency values from 100 Hz to 1 kHz.

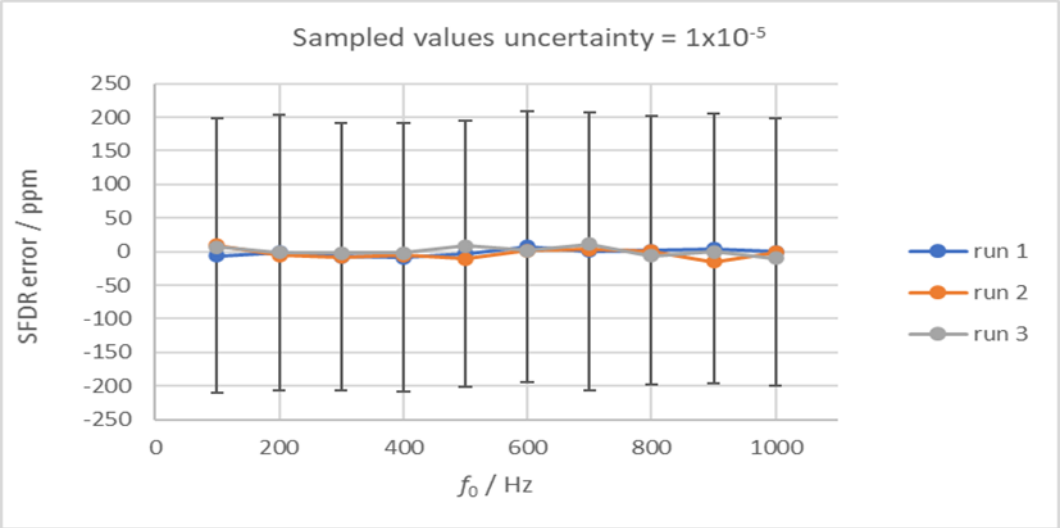


Figure 7.3: SFDR error of the algorithm output from a simulated signal (SFDR −80 dB and $f_s = 1.5 \cdot f_0$) with sampled values relative uncertainty of 1×10^{-5} V as a function of frequency values from 100 Hz to 1 kHz.

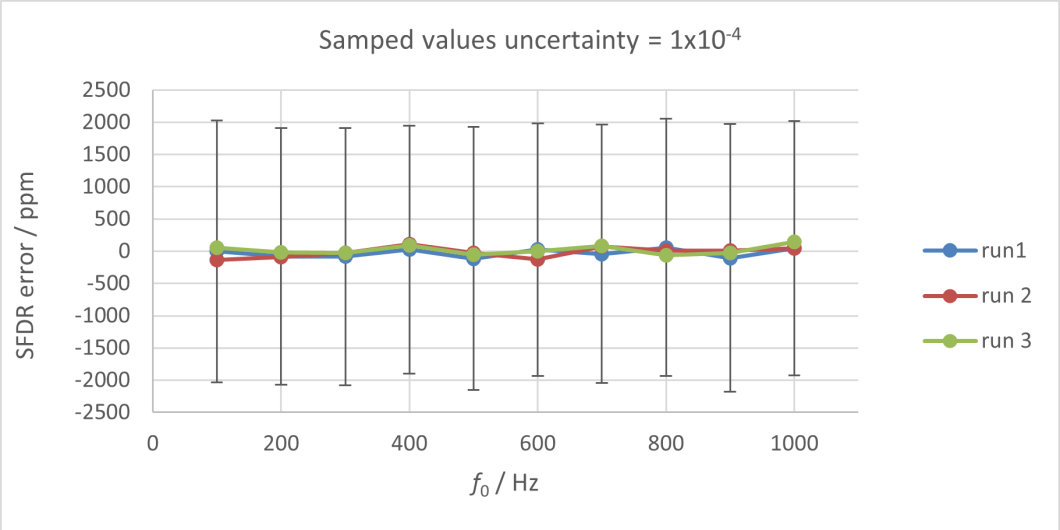


Figure 7.4: SFDR error of the algorithm output from a simulated signal (SFDR −80 dB and $f_s = 1.5 \cdot f_0$) with sampled values uncertainty of 1×10^{-4} V as a function of frequency values from 100 Hz to 1 kHz.

7.3 Algorithm error for non-coherent sampling

To see the performance of the algorithm with non-coherent sampling, a deviation of 10 µHz/Hz in the fundamental frequency to the simulated test signal was introduced (script SFDR_test.m in appendix G). Results obtained are represented in figures 7.5 to 7.7.

It was observed that the SFDR relative error depends on the input quantities of the signal: SFDR, fundamental frequency and non-harmonic component.

For the range of signal quantities tested, without considering the results of the spurious frequency $f_s = 1.1 \cdot f_0$, the largest error founded was 210 ppm from a signal with SFDR of −40 dB (relative to carrier) and with a spurious frequency equal to 4.5 multiples of the fundamental frequency.

For signals with a spurious frequency $f_s = 1.1 \cdot f_0$ and SFDR = −80 dB and SFDR = −90 dB, the absolute value of the SFDR error increases exponentially with the decrease of the fundamental frequency value. For higher values of the fundamental frequency, we will have a higher number of sampled periods of the signal and this seems to attenuate this effect observed for low frequencies.

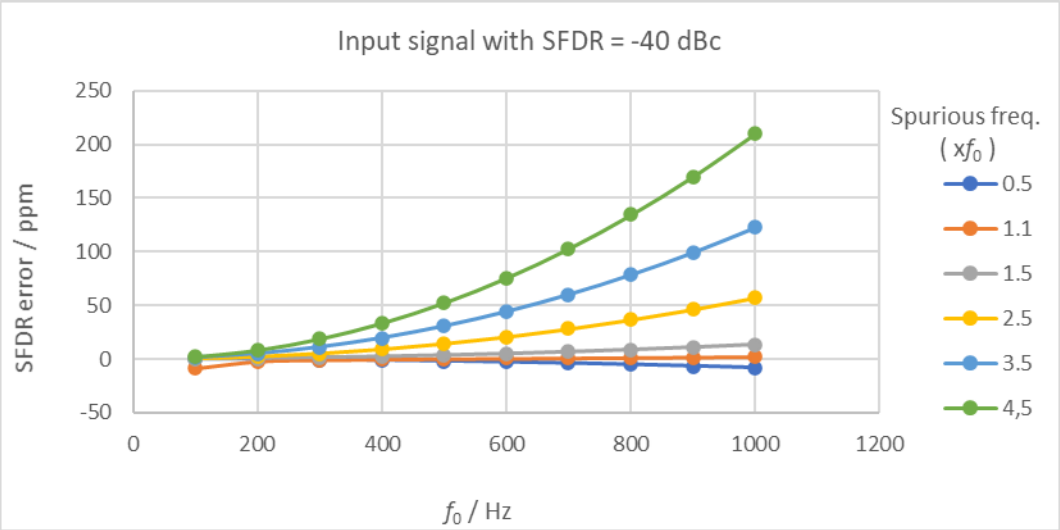


Figure 7.5: SFDR error of the algorithm output from simulated signal with SFDR −40 dB and non-coherently sampled values by the introduction of a relative deviation in fundamental frequency of 10 µHz/Hz.

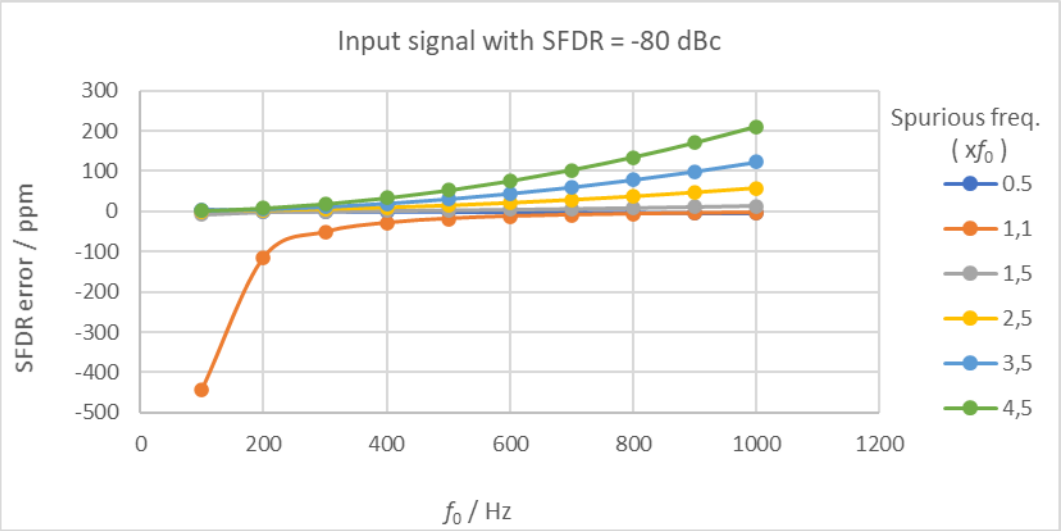


Figure 7.6: SFDR error of the algorithm output from a simulated signal with SFDR −80 dB and non-coherently sampled values by the introduction of a relative deviation in fundamental frequency of 10 µHz/Hz.

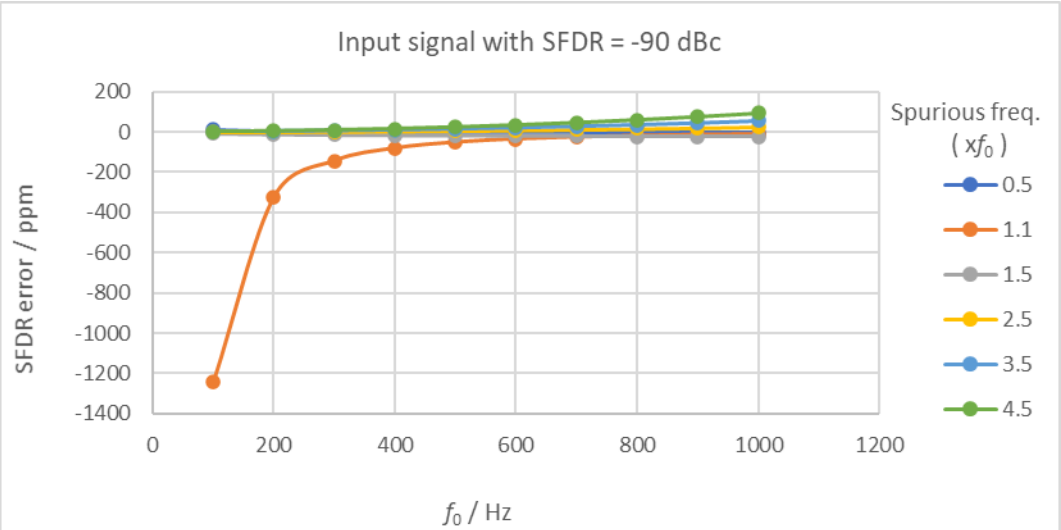


Figure 7.7: SFDR error of the algorithm output from a simulated signal with SFDR −90 dB dBc and non-coherently sampled values by the introduction of a relative deviation in fundamental frequency of 10 µHz/Hz.

A Blackman-Harris window with stronger leakage reduction than Blackman window [9] was experimented for signals with $f_s = 1.1 \cdot f_0$. The results are presented in figures 7.8

and 7.9. The results show that for low frequencies the algorithm error is strongly reduced. For higher frequencies (>500 Hz), the absolute error increases slightly.

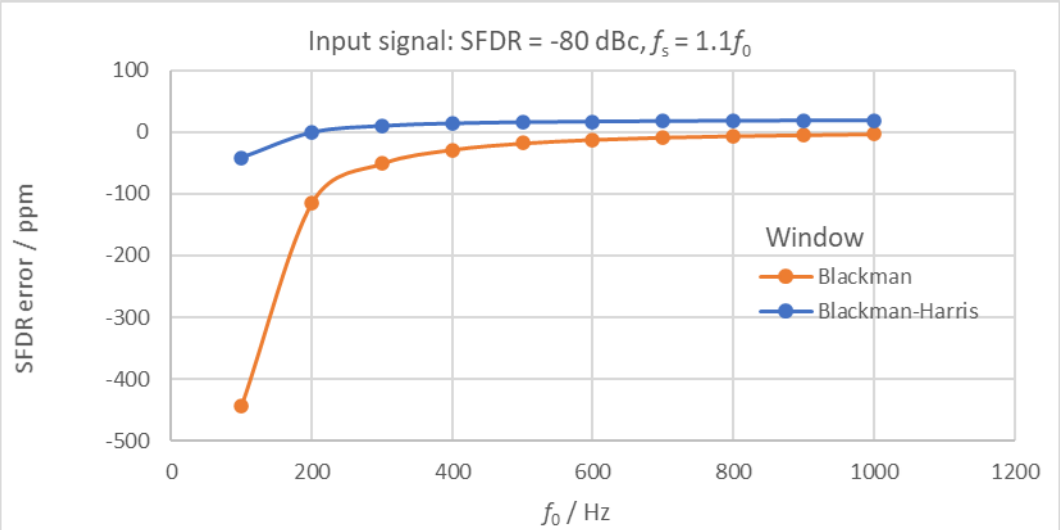


Figure 7.8: SFDR error of the algorithm output using different window from a non-coherently sampled signal of SFDR −80 dB and $f_s = 1.1 \cdot f_0$.

Some tests were also done for SFDR values of −100 dB and −120 dB. For SFDR = −100 dB and for frequencies above 300 Hz, the algorithm relative error values are in the interval from -1 % to -9 % . For SFDR −120 dB and for the entire range of frequency tested (100 Hz to 1 kHz) the algorithm relative errors obtained are in the interval from -7 % to -24 % . In both cases, these errors show dependence only on the value of the signal frequency and are independent of the tested non-harmonic component.

7.4 SFDR conclusion

The SFDR algorithm presents a residual systematic error (the maximum relative error found was 6×10^{-9}) for a coherent sampling of sine wave signal in the frequency range tested (100 Hz to 1 kHz), with non-harmonic components from 0.5 to 4.5 of the main frequency and to SFDR values from −40 to 140 dB.

The algorithm output shows dependence on the random noise value present in the sampled signal:

- with standard deviation values within one order of magnitude greater than the random noise.
- with the standard deviation of the result covered by two orders of magnitude the relative error of the SFDR estimation.

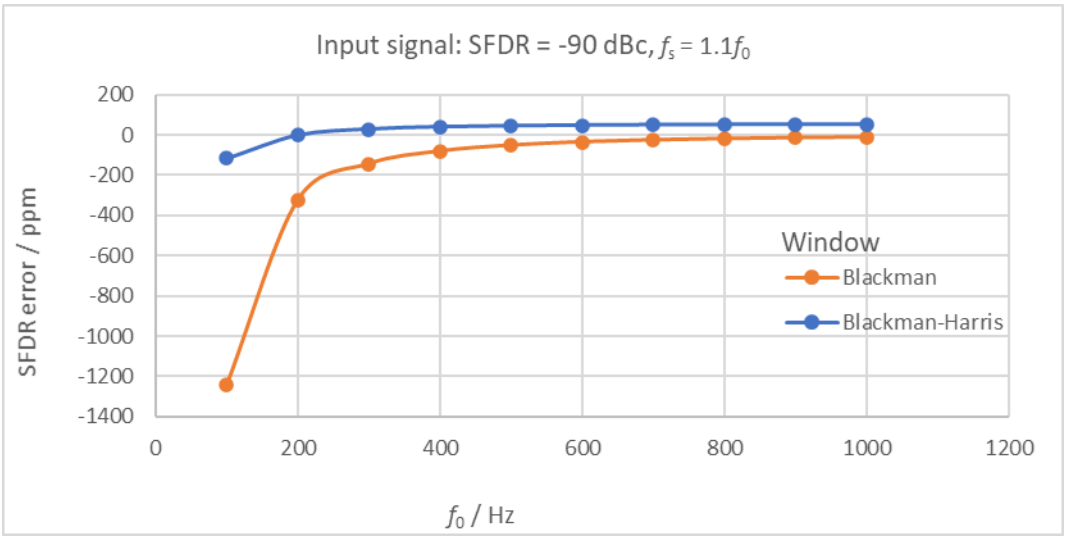


Figure 7.9: SFDR error of the algorithm output using different window from the non-coherently sampled signal of SFDR −90 dB and $f_s = 1.1 \cdot f_0$.

The uncertainty added to each sampled value is processed by the qwtb.m function and generates an uncertainty estimation which is in agreement with the observed algorithm dependence on random noise.

The application of SFDR algorithm to non-coherently sampled signal generate results with significant systematic error which depends on the signal parameter values: frequency, non-harmonic component present and SFDR value. Error values can reach hundreds of ppm, for non-harmonic component equal or greater than 0.5 times the fundamental frequency and can reach up to 0.12 % as was observed in the worst case, for a signal with a non-harmonic component, f_s, close to the fundamental, f_0: $f_s = 1.1 \cdot f_0$. The type of window used has a strong influence on the error obtained and can be effective to reduce significantly the algorithm error. For signals with SFDR values equal or below −100 dB, the algorithm is not working properly, generating output errors with values too high to be considered acceptable, reaching the relative value of -24%.

Chapter 8

INL-DNL algorithm validation and uncertainty estimation

Integral non-linearity (INL) and differential non-linearity (DNL) are used to measure the performance of analog-to-digital (ADC) converters. These measurement are performed after offset and gain errors have been compensated.

INL represents the deviation between the ideal input threshold value and the measured threshold level of a certain output code. The ideal transfer function of ADC is a straight line. The INL measurement depends on what line is chosen as ideal. One option is the line connecting the smallest and largest measured input/output value. A very good representation of INL using this option is shown in Figure 8.1. An alternative is to use a best fit line. This is the option chosen in QWTB algorithms. While the INL can be measured for every possible input/output code, often only the maximal error is provided when reporting the INL of a converter.

DNL is defined as the difference between an actual step width and the ideal value of one Least Significant Bit (LSB) as it is shown in Figure 8.2. The accuracy of a DAC is mainly determined by this specification. Ideally, any two adjacent digital codes correspond to output analog voltages that are exactly one LSB apart. Differential non-linearity may be expressed in fractional bits or as a percentage of full scale. A differential non-linearity greater than 1 LSB may lead to a non-monotonic transfer function in a DAC. It is also known as a missing code.

Figure 8.3 shows an example generated by QWTB code for a hypothetical 3 bits ADC converter, where transition 2 has been moved one LSB to the left, and transition 6 has been moved one LSB to the right. Note that INL and DNL values are going to be close to 1 (but not 1), since QWTB uses best fit line.

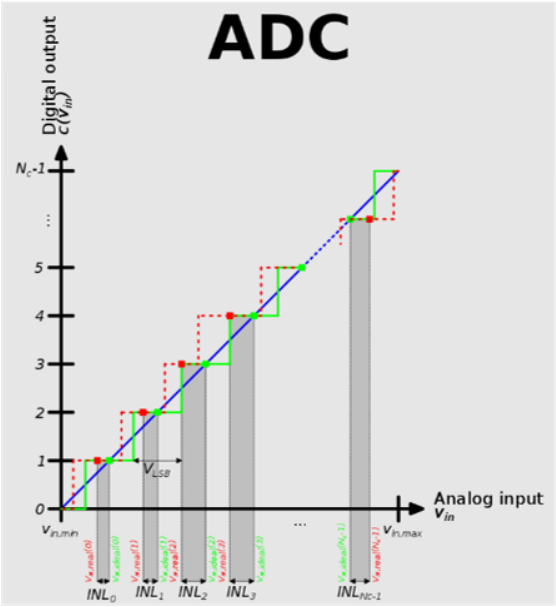


Figure 8.1: INL is represented at the bottom of the graph: the difference between mid points of the ideal transfer function (in green) and mid points of the actual transfer function (in red). Credit [10].

8.1 INL-DNL algorithm in QWTB

QWTB algorithm uses the sine-wave histogram method to located code transitions and then calculate INL and DNL values.

The first step is to input a sinewave with an overdrive, to ensure all codes are covered. ADC transform this signal to discrete values (Figure 8.4 left). The number of samples in each code are counted and a histogram is built (Figure 8.4 right). First and last bins of the histogram are discarded, and the other bins are compared to the ideal case. INL and DNL values are calculated from the histogram.

As only a finite number of records is possible, the histogram shape is always different to ideal histogram shape, even if there is no INL and DNL error. Therefore, the algorithm has an inherent error when INL and DNL are calculated.

As DNL is calculated from INL, in this study just INL has been researched.

The way to proceed was to input a known solution and compare the differences between algorithm solution and input solution when some variables are changed.

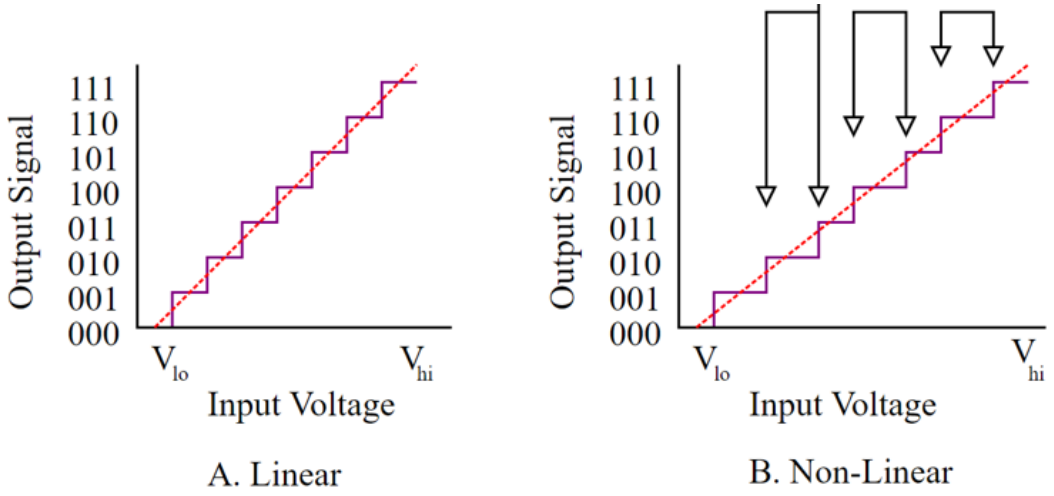


Figure 8.2: Linear response (left) and non linear response (right) where the DNL is calculated as the difference between the ideal code width and the actual code with (represented by the arrows). Credit [11]

8.2 Algorithm error and uncertainty

The first objective would be to calculate the algorithm error in order to correct it. However, this aim is only possible when influence variables have very little uncertainty (or extremely small, for example, for ADCs with more than 14 bits). This happens especially if there is noise, but it is also true for overdrive (see 8.3 for definitions). If this is the assumption, a case-by-case basis would be the right approach since errors are quantized, and they cannot be fitted to a smooth curve. This is similar, somehow, to quantization noise.

If uncertainty of input variables gets higher, INL uncertainty prevails over INL algorithm error, and a *one size fits all* solution can be provided.

In this way and regarding the aim of quick estimation of the uncertainty, the problem is reduced to two cases:

1. Input noise is small: algorithm errors are quantized, and just a maximum uncertainty can be provided since the mean of the error is not zero and varies abruptly

2. Input noise is big: uncertainty prevails over algorithm error and an uncertainty can be provided since the mean of the error is zero. To simplify the problem this uncertainty will be a maximum which is very close to the true uncertainty reached for most of the codes

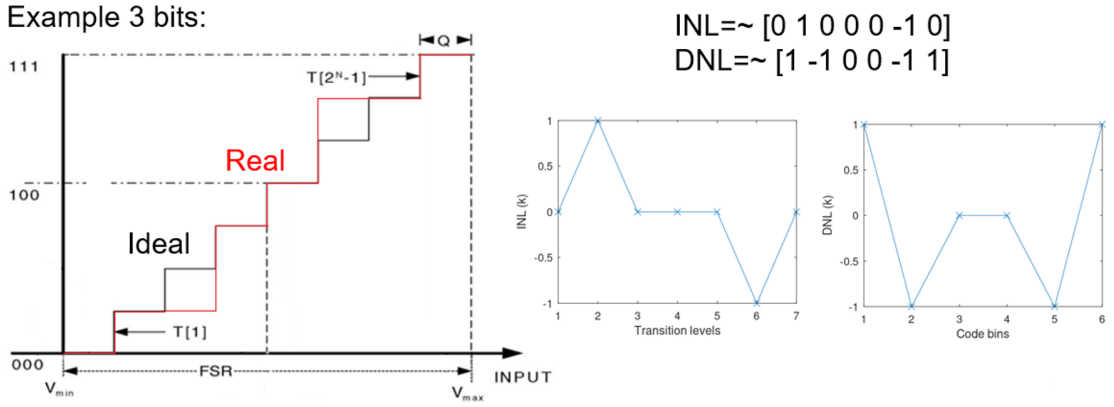


Figure 8.3: 3 bits INL and DNL example where transitions 2 has been moved one LSB to the left, and transition 6 has been moved one LSB to the right. INL and DNL values different from 0 are going to be close to 1 LSB (but not 1) since QWTB algorithm uses best fit line.

8.3 Algorithm uncertainty dependence on input quantities

Five variables that influences the error committed by the algorithm have been identified:

- N: Number of bits of the ADC resolution.
- f_s/f_i: Ratio between sampling frequency f_s and input signal frequency f_i. It is equivalent to n/m (ratio between number of samples and number of cycles of the record).
- m: Number of cycles (cycles must be an integer number).
- O_v: Overdrive. Difference between input signal amplitude and ADC semi range when input signal is centred in the range. It can be calculated as

$$O_v = A - R/2, \tag{8.1}$$

  where A is the amplitude of the input signal and R is the range.
- INL: INL value. The own value of the INL could influence the algorithm error
- σ: white noise of the input signal, as a percentage of the ADC input range

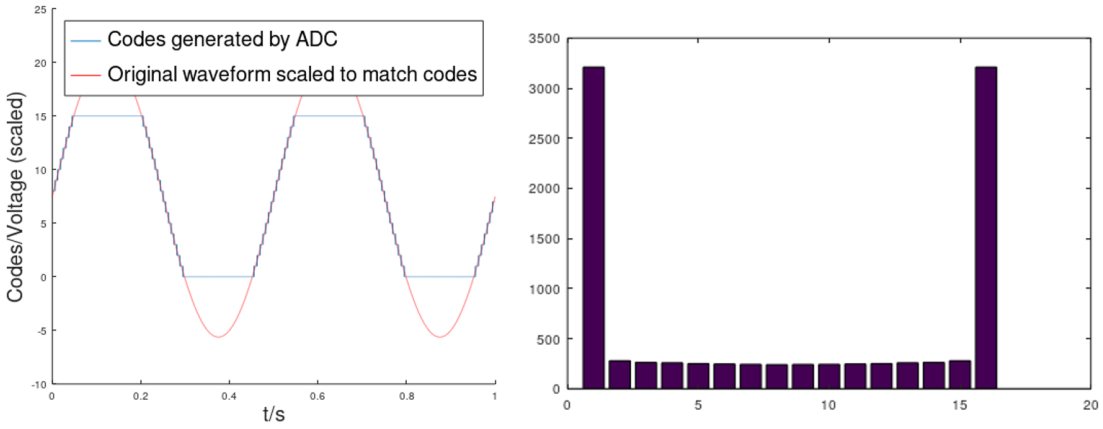


Figure 8.4: Sine-wave histogram method. The first step is to input a sine-wave saturated signal to the ADC. Then the number of samples in each code is counted and represented in an histrogram. Extreme bins are removed and histogram is compared to ideal case to estimate INL and DNL values for all the codes.

It is clear that the number of bits is a fixed value. Also, the number of samples is considered as a fixed value with no uncertainty. Experimentally it has been proven that the influence of the overdrive is small if this is known with enough accuracy, and that the value of the INL is not important for the calculation of the maximum uncertainty. Therefore, just the noise (uncertainty of the input signal) will be considered as the only uncertainty source in this study.

The QWTBvar function has not been directly used in the INL uncertainty estimation since just the variables number of bits and codes (from ADC) are the inputs of the algorithm. Note that codes have implicitly included the overdrive and number of samples. The validation of the algorithm and the uncertainty estimation have required the variation of other variables (singled and combined), as the INL and the noise of the input signal. The simplifications and conclusions from previous paragraphs were only possible with a thorough study of these variables.

The following variable ranges have been considered during calculations. It is believed that results can be extrapolated to wider ranges, although some systematic errors appear when Noise>1% and/or when overdrive is close to 0.

- N: 8 to 24 bits,
- m: 1 to 200 cycles,
- f_s/f_i: 5×10^4 to 5×10^7,
- O_v: $R/10$ to $R/2$,

- INL: −1 to 1,
- σ: 0 to 1 % of the ADC input range.

As an example, Figure 8.5 shows the calculated uncertainty for a 10 bits ADC, for different noises and f_s/f_i for one cycle ($m = 1$), and a fixed overdrive of 1 V ($R = 4$ V). The two cases explained in section 8.2 are identified. Although this is a particular instance, it is demonstrated experimentally that applies to all the variables considered in this study.

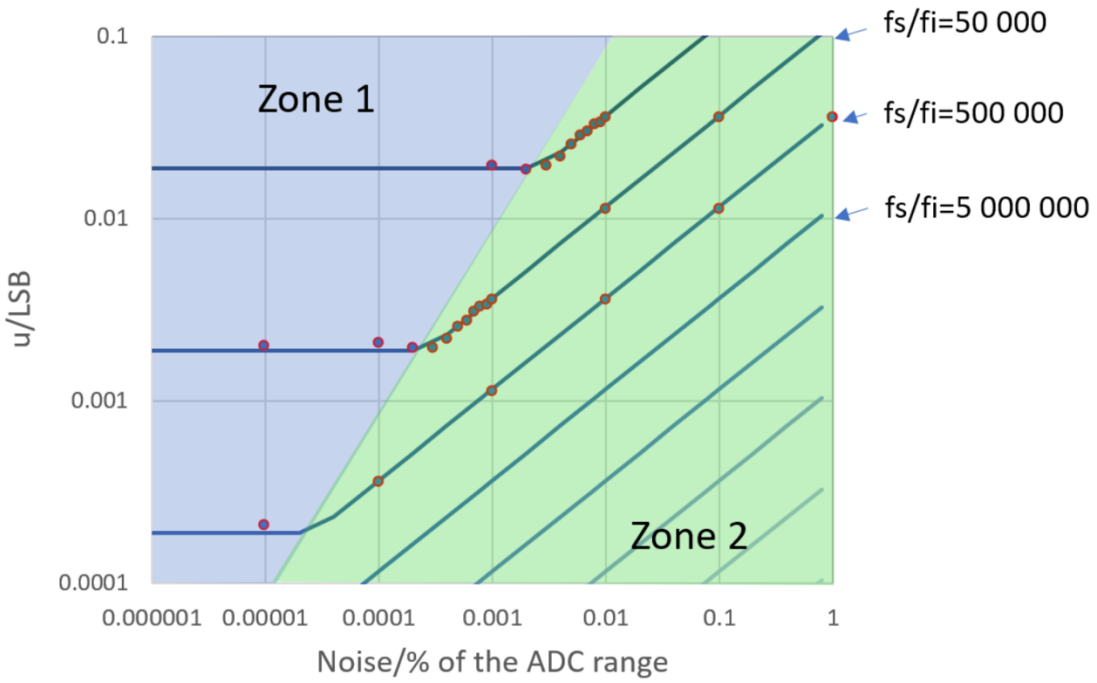


Figure 8.5: Uncertainty (y axis) vs. noise (x axis, as percentage of the range) for different sampling frequency and input signal frequency ratios. Points indicate calculated values, and lines fitted values. Two zones are identified depending on noise.

In Figure 8.5, points represent calculated values and lines represent the fitting of these points. For both zones, INL uncertainty fits quite well into the following equation::

$$u(INL) = C\sqrt{\frac{\sigma}{m}\frac{f_i}{f_s}(2^N - 1)} \tag{8.2}$$

Where in zone 1:

$$C = C_1 = \sqrt{\frac{m}{\sigma}\frac{f_i}{f_s}}\left(1.162 + \frac{1.155}{R/2}O_v\right) \tag{8.3}$$

And in zone 2:

$$C = C_2 = 0.1131 + 0.0822\frac{2O_v}{R} \tag{8.4}$$

Note that C_2 is just a constant with an overdrive correction. Note also that in zone 1 uncertainty is independent of noise and that it can be written also as a constant with an overdrive correction too.

With equations 8.2 and 8.4, the uncertainty of zone 2 can be calculated with high accuracy. Although they provide the maximum uncertainty in this zone it fits very well to the uncertainty of most of the codes.

For zone 1, equations 8.2 and 8.3, provides an upper limit of the uncertainty since it is impossible to provide a general case. This upper limit is currently very conservative, especially close to the boundaries of the two zones when m is high, but with some extra work could be improved to reflect uncertainty bounds in a better way.

The limit between zone 1 and 2 (noise) can be calculated approximately making equal $C_1 = C_2$ for $m = 1$.

8.3.1 Examples

Some examples are provided to show the usefulness of the formula provided and its accuracy. In this examples the expanded uncertainty ($k = 2$ for a 95 % of coverage factor) is calculated by the formula and by MCM simulations and then compared graphically. Table 1 shows the calculation parameters of the examples.

Table 8.1: Parameters used in the examples.

#	**Bits** N	**Noise** σ	**Freq. ratio** f_s/f_i	**Cycles** m	**Range**	**Overd.** O_v	**Amp.** A	**Zone**
1	12	0.25 %=10 mV	50 000	8	4 V	1 V	3 V	2
2	12	0.025 %=1 mV	50 000	8	4 V	1 V	3 V	2
3	12	0.0025 %=100 µV	50 000	8	4 V	1 V	3 V	1
4	12	0.00025 %=10 µV	50 000	8	4 V	1 V	3 V	1
5	12	0 %=0 µV	50 000	8	4 V	1 V	3 V	1

In zone 2, Figure 8.6 shows that the uncertainty calculated by8.2 and 8.4, matchs very well to uncertainty calculated by simulations. Central areas present slightly lower uncertainty and points at the end present decreasing uncertainty with 0 uncertainty in the ends (fixed points). For these areas the formula does not represent the uncertainty so well but globally the performance is excellent. Note that since the number of MCM simulations is just 1000, some noise appear for zone 2 examples. Also, constant C_2 was

calculated for 1000 simulations so it covers well the noise in the figures but it can be refined.

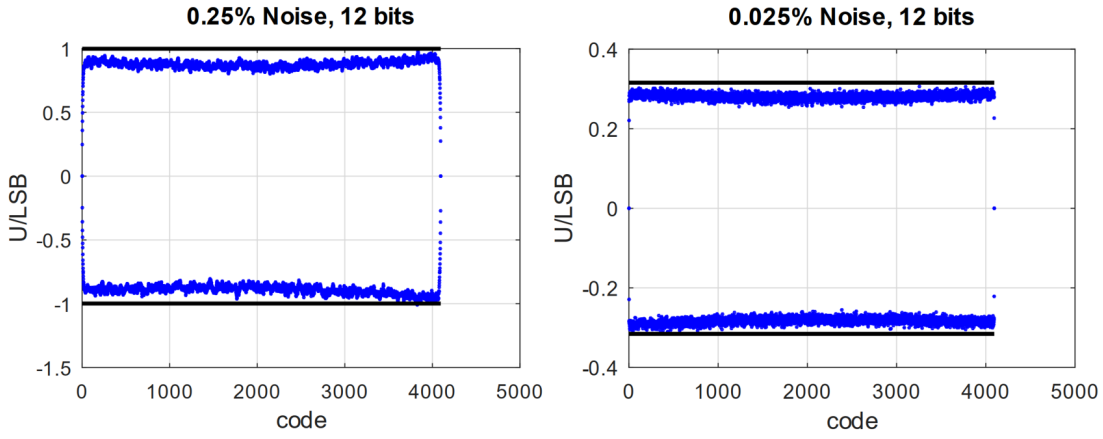


Figure 8.6: Uncertainty bounds calculated by the formula (black lines) and uncertainty bounds calculated by MCM method (blue dots) for the zone 2. Noise appeared due to limited number of MCM trials.

In zone 1 the variability represented is not a noise but the algorithm error that is quantized (patterns appear) (Figure 8.7). In this way, in the case where the noise is 0%, the graph just show the algorithm error with zero uncertainty. The uncertainty provided by8.2 and 8.3 keeps error plus uncertainty values within uncertainty bounds, since providing individual corrections is not possible. Thus, systematic effects are not corrected but substituted by an uncertainty [12] Uncertainty bounds are here quite conservative and further work is necessary to reduce them. Nevertheless, a maximum uncertainty bound, which works for all cases, is provided in a simple way showing the potential of the formula.

8.4 INL-DNL validation and calculation tips

The QWTB INL-DNL has been used extensively for different cases and parameters during this study and its output has been compared to an independant ideal output. The results has been always satisfactory so the algorithm is considered as validated.
In this study some tips were found and are included here:

- Since an integer number of cycles is necessary for the study, it is better to start the first point when the ADC is out of range (maximum or minimum amplitude).
- The minimum necessary overdrive is equal to the amplitude of input noise.

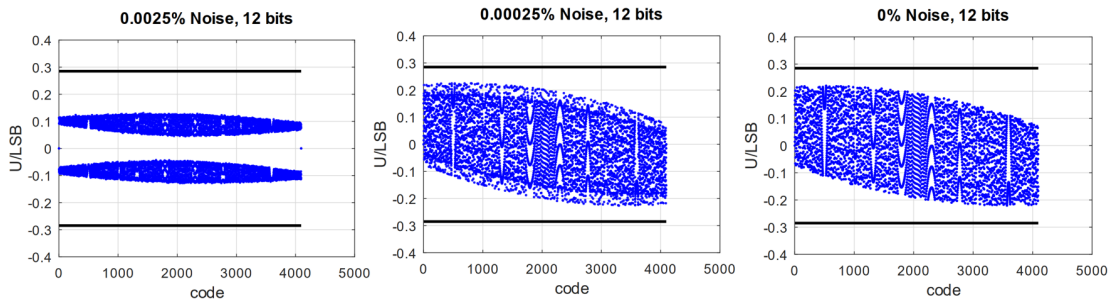


Figure 8.7: Uncertainty bounds calculated by the formula (black lines) and uncertainty bounds calculated by MCM method (blue dots) for the zone 1. Apparent noise is the algorithm error since it is quantized.

- The best way to take more samples from ADC is to input a sine wave with lower frequency, instead of increase the sampling rate.
- The number of cycles must be an integer.
- Offset and gain errors must be compensated beforehand.

On the other hand, since multiple MCM simulations with different variables has been performed in this study, calculation time has been a key parameter. At the beginning, the QWTB INL-DNL algorithm was optimized, and calculation time was remarkably reduced in a 90 %. Modified code has been included in Appendices J and K.

Chapter 9

Conclusion

The document described a general theory on errors and uncertainties of algorithms and method on estimating the errors. A new software QWTBvar is presented. This Document contains both the documentation to the software and example of its use. Up to it, validation of algorithms for estimation of THD, SFDR and INL-DNL have been provided. A method to speed up uncertainty estimation for SFDR algorithm have been shown. For INL-DNL, the uncertainty estimation was based on the analytical solution.

Chapter 10

Bibliography

[1] JCGM, *Evaluation of Measurement Data - Guide to the Expression of Uncertainty in Measurement*. Bureau International des Poids et Measures, 1995, ISBN: 92-67-10188-9. [Online]. Available: https://www.bipm.org/en/publications/guides/.

[2] ——, *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, 2008. [Online]. Available: https://www.bipm.org/en/publications/guides/.

[3] ——, *Evaluation of Measurement Data – Supplement 2 to the "Guide to the Expression of Uncertainty in Measurement" – Extension to Any Number of Output Quantities*. Bureau International des Poids et Measures, 2011, 80 pp. [Online]. Available: https://www.bipm.org/en/publications/guides/.

[4] M. Šíra, *QWTB - Software Toolbox for Sampling Measurements*, Czech Metrology Insitute, 2017. [Online]. Available: https://qwtb.github.io/qwtb/ (visited on 05/24/2019).

[5] *Matlab*, 2012. [Online]. Available: http://www.mathworks.com.

[6] *GNU Octave*, 2012. [Online]. Available: https://www.gnu.org/software/octave/.

[7] TracePQM consortium. (2019), [Online]. Available: http://tracepqm.cmi.cz.

[8] S. Mašláň, "Report A2.4.4: TWM algorithms description," Czech Metrology Institute, A2.4.4, p. 38. [Online]. Available: https://github.com/smaslan/TWM/blob/master/doc/A244%20Algorithms%20description.pdf (visited on 05/24/2019).

[9] R. Lapuh, *Sampling with 3458A*. Left Right d.o.o., Sep. 2018, ISBN: 978-961-94476-0-4.

[10] Fvultier. (2018), [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=74754102.

[11] Egmason. (2012), [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=18321482.

[12] K. Klauenberg, G. Wübbeler, and C. Elster, "About not correcting for systematic effects," *Measurement Science Review*, vol. 19, no. 5, pp. 204–208, 2019.

[13] S. Mašláň, *Activity A2.3.2 - Algorithms Exchange Format*. [Online]. Available: https://github.com/smaslan/TWM/tree/master/doc/A232%20Algorithm%20Exchange%20Format.docx.

[14] S. Mašláň and M. Šíra, "Automated non-coherent sampling thd meter with spectrum analyser," in *Proceedings CPEM*, 2014.

[15] 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)*, Aug. 2014, pp. 334–335. DOI: 10.1109/CPEM.2014.6898395.

[16] R. Lapuh, "Estimating the fundamental component of harmonically distorted signals from noncoherently sampled data," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 6, pp. 1419–1424, Jun. 2015, ISSN: 0018-9456. DOI: 10.1109/TIM.2015.2401211.

[17] G. 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," Max-Planck-Institut für Gravitationsphysik (Albert-Einstein-Institut) Teilinstitut Hannover, Tech. Rep., Feb. 2005. [Online]. Available: http://holometer.fnal.gov/GH_FFT.pdf.

[18] M. Valtr. (2014). "ČMI HPC System Online," [Online]. Available: http%5C%3A%5C%2F%5C%2Fprutok.cmi.cz%5C%2Fsc%5C%2Fdoku.php%5C%3Fid%5C%3Dsystem%5C&edit-text= (visited on 08/20/2018).

[19] J. Schoukens, R. Pintelon, and G. Vandersteen, "A sinewave fitting procedure for characterizing data acquisition channels in the presence of time base distortion and time jitter," *IEEE Transactions on Instrumentation and Measurement*, vol. 46, no. 4, pp. 1005–1010, Aug. 1997, ISSN: 0018-9456. DOI: 10.1109/19.650817.

[20] R. Pintelon and J. Schoukens, "An improved sine-wave fitting procedure for characterizing data acquisition channels," *IEEE Transactions on Instrumentation and Measurement*, vol. 45, no. 2, pp. 588–593, Apr. 1996, ISSN: 0018-9456. DOI: 10.1109/19.492793.

Appendices

Appendix A

alg_compare.m

```
% the FFT and MHSF algorithms will be tested for the quality of
    THD
% estimate for a selected signals and estimated uncertainties

function alg_compare()
    %%%%%%%%% CALCULATION SETTINGS %%%%%%%%%  %<<<1
    dev = 0; % if developement on simple computer, set to zero
   for final results
    % path to the qwtb:
    addpath('qwtb/')
    % calculation settings:
    % monte carlo:
    CS.verbose = 1;
    CS.unc = 'mcm';
    CS.loc = 0.6827;
    CS.cor.req = 0;
    CS.cor.gen = 0;
    CS.dof.req = 0;
    CS.dof.gen = 0;
    CS.mcm.repeats = 1e3;
    CS.mcm.verbose = 1;
    CS.mcm.method = 'multicore';
    % CS.mcm.method = 'singlecore';
    CS.mcm.procno = 6;
    CS.mcm.tmpdir = '.';
    CS.mcm.randomize = 1;
    CS.checkinputs = 1;
    % variation settings:
    CS.var.dir = ['THDcomp_' CS.unc];
```

```
CS.var.cleanfiles = 1;
%%%%%%%% CALCULATION SETTINGS %%%%%%%%

% general signal properties: %<<<1
% sampling frequency:
DI.fs.v = 1e5;
DI.fs.u = 1e-9; % is it used somewhere?
% record length:
DI.L.v = 1e5;
% signal frequency
DI.f.v = 50.01;
DI.f.u = 0.01; % this is uncertainty of estimate. is it
used in some alg?
% harmonics
DI.nharm.v =  [1   2   3   4   5]; % signal harmonic
multiple
DI.A.v =  [1 0.01 0.01 0.01 0.01]; % signal amplitudes
DI.ph.v = [0   0   0   0   0]; % signal phase
DI.O.v =  [0   0   0   0   0]; % signal offset
DI.noise.v = 1e-5;
% nonsense uncertainties because qwtb checks uncertainties
of all quantities:
DI.L.u = 0;
DI.nharm.u =  zeros(size(DI.nharm.v));
DI.A.u =  zeros(size(DI.A.v));
DI.ph.u =  zeros(size(DI.ph.v));
DI.O.u =  zeros(size(DI.O.v));
DI.noise.u =  zeros(size(DI.noise.v));

% calculation %<<<1

%%% Variation: noise %<<<2
% dependence of output THD on the input noise:
label = 'noise';
clear DIvar;
DIvar.noise.v = logspace(-6, 1, 20);
if dev DIvar.noise.v = logspace(-6, 1, 2); end
% DIvar.L.v = round(logspace(5, 6, 20));
jobfn = processing(label, DI, DIvar, CS, 'noise', 'signal
noise sigma', 'THD (percent to main harm.)');

%%% Variation: thd value %<<<2
```

```
    % dependence of output THD on the input THD value -
   gradually increasing harmonics:
    label = 'thd';
    clear DIvar;
    DIvar.A.v = logspace(-6, -2, 10)'.*[1 1 1 1];
    if dev DIvar.A.v = logspace(-6, -2, 2)'.*[1 1 1 1]; end
    DIvar.A.v = [ones(size(DIvar.A.v,1), 1) DIvar.A.v];
    jobfn = processing(label, DI, DIvar, CS, 'THDref', 'signal
   THD value', 'THD (percent to main harm.)');

    %%% Variation: signal length, large span %<<<2
    % dependence of output THD uncertainty on the signal length
   :
    label = 'siglen';
    clear DIvar;
    DIvar.L.v = logspace(5, 7, 5); % 10^8 is too much for
   notebook, 10^6 is too much for general monte carlo method (
   too large matrices)
    if dev DIvar.L.v = logspace(5, 6, 2); end
    jobfn = processing(label, DI, DIvar, CS, 'L', 'signal
   length', 'THD (percent to main harm.)');

    %%% Variation: signal length, small span - not interresting
   , boring, do not do! %<<<2
    % dependence of output THD uncertainty on the signal length
   :
    % label = 'siglen';
    % DIvar.L.v = 1e5+[-20:0.01:20]; % 10^8 is too much for
   notebook

    %% Variation: signal frequency %<<<2
    % dependence of output THD on the signal frequency:
    label = 'sigfreq';
    clear DIvar;
    DIvar.f.v = 50 + [-0.5:0.01:0.5];
    if dev DIvar.f.v = 50 + [-0.5:0.1:0.5]; end
    jobfn = processing(label, DI, DIvar, CS, 'f', 'signal
   frequency', 'THD (percent to main harm.)');
endfunction

function jobfn = processing(label, DI, DIvar, CS, xvar, xlbl,
   ylbl) %<<<1
    CS.var.dir = [CS.unc '_' label];
```

```
    jobfn = qwtbvar('thdtest', DI, DIvar, CS);

    % output plotting
    figure('visible','off');
    hold on
    [x, y] = qwtbvar(jobfn, xvar, 'thdffterr');
    plot(x.v, y.v, '-xr');
    plot(x.v, y.v - y.u, '-r');
    plot(x.v, y.v + y.u, '-r');
    [x, y] = qwtbvar(jobfn, xvar, 'thdmfsferr');
    plot(x.v, y.v, '-xb');
    plot(x.v, y.v - y.u, '-b');
    plot(x.v, y.v + y.u, '-b');
    xlabel(xlbl);
    ylabel(ylbl);
    legend('wfft', 'neg', 'pos', 'mfsf', 'neg', 'pos');
    title(jobfn);
    hold off
    pltfn = fullfile(CS.var.dir, label);
    printplt(pltfn); replot_to_pdf(pltfn);
   plot_change_gnuplot_terminal(CS.var.dir);

endfunction


% vim settings modeline: vim: foldmarker=%<<<,%>>> fdm=marker
   fen ft=octave textwidth=80 tabstop=4 shiftwidth=4
```

source_code_hard_links/alg_compare.m

Appendix B

thdtest.m

```
% test function used for variation of TWM-THDWFFT and TWM-MFSF
function [DO, DI, CS] = thdtest(DI, CS)
    % generate signal %<<<1
    % time series:
    DI.t.v = [0 : 1./DI.fs.v : DI.L.v./DI.fs.v];
    DI.t.u = ones(size(DI.t.v)).*1e-10;
    % sampled values:
    DI.y.v = DI.A.v'.*sin(2.*pi.*DI.f.v.*DI.nharm.v'.*DI.t.v +
   DI.ph.v') + DI.O.v';
    DI.y.v = sum(DI.y.v, 1);
    % add noise to the data:
    DI.y.v = DI.y.v + normrnd(0,DI.noise.v,size(DI.y.v));
    % uncertainties of every sample:
    DI.y.u = ones(size(DI.y.v)).*1e-5;
    % frequency components to fit for MFSF:
    DI.ExpComp.v = DI.nharm.v;
    DI.ExpComp.u = zeros(size(DI.ExpComp.v));

    % save memory:
    DI = rmfield(DI, 't');

    % calculate thd by algorithms %<<<1
    wfftDO = qwtb('TWM-THDWFFT', DI, CS);
    mfsfDO = qwtb('TWM-MFSF', DI, CS);

    % calculate signal reference THD:
    DO.THDref.v = sum(DI.A.v(2:end).^2)^0.5./DI.A.v(1) * 100;
    % output data %<<<1
    DO.thdffterr.v = wfftDO.thd.v - DO.THDref.v;
```

```
    DO.thdffterr.u = wfftDO.thd.u;
    DO.thdmfsferr.v = mfsfDO.thd.v - DO.THDref.v;
    DO.thdmfsferr.u = mfsfDO.thd.u;
end
```

source_code_hard_links/thdtest.m

Appendix C

make_lut.m

```
clear all; close all;
% function make_lut()
    % calculation settings %<<<1
    CS_lut.strict = 0;
    CS_lut.verbose =  0;
    CS_lut.checkinputs =  1;
    CS_lut.unc = 'mcm';
    CS_lut.loc =  0.68270;
    CS_lut.cor.req =  0;
    CS_lut.cor.gen =  0;
    CS_lut.dof.req =  0;
    CS_lut.dof.gen =  0;
    CS_lut.mcm.repeats =  100;
    CS_lut.mcm.verbose =  1;
    CS_lut.mcm.method = 'singlecore';
    CS_lut.mcm.procno = 0;
    CS_lut.mcm.tmpdir = '.';
    CS_lut.mcm.randomize =  1;

    CS_lut.var.dir = '_temp';
    CS_lut.var.fnprefix = 'var';
    CS_lut.var.cleanfiles = 1;
    CS_lut.var.smalloutput = 0;
    CS_lut.var.method = 'singlecore';
    CS_lut.var.procno = 1;
    CS_lut.var.chunks_per_proc = 1;

    % create qwtbvar %<<<1
```

```
% f - main signal frequency, list of spurious/harmonics
frequencies
% A - main signal amplitude, list of spurious/harmonics
amplitudes
% ph - main signal phase, list of spurious/harmonics phases
% O - main signal offset, list of spurious/harmonics
offsets
% dc - dc component
% Np - scalar real, number of main signal periods in the
record ( Ns/fs = Np/f(1) )
% ssr - ratio of sampling to signal frequency, ssr = fs/f
(1).
% SFDR - scalar, spurious free dynamic ratio
% jitter - standard deviation of jitter [s]
% noise - standard deviation of noise [V]%
% smr - spurious to main signal frequency multiple

% Nominal data in values:
DI_gen.f.v = 1000;
DI_gen.A.v = 1;
DI_gen.O.v = 0;
DI_gen.ph.v = 0;
DI_gen.dc.v = 0;
DI_gen.Np.v = 10;
DI_gen.ssr.v = 100;
DI_gen.SFDR.v = 1e-6;
DI_gen.jitter.v = 1e-12;
DI_gen.noise.v = 1e-6;
DI_gen.smr.v = 5;

% axis granularity
gr = 3;
% variated data in values:
DI_gen_var.Np.v = linspace(1, 100, gr);
DI_gen_var.ssr.v = linspace(20, 100, gr);
DI_gen_var.A.v = linspace(1e-3, 1000, gr);
DI_gen_var.jitter.v = linspace(1e-13, 1e-10, gr);
DI_gen_var.noise.v = linspace(1e-8, 1e-5, gr);
DI_gen_var.SFDR.v = linspace(1, 1e8, gr);

% run qwtbvar %<<<1
jobfn = qwtbvar('calc', 'gen_and_calc', DI_gen, DI_gen_var,
CS_lut);
```

```
% make lut %<<<1
 ax_set_lut.Np.v.scale = 'lin';
 ax_set_lut.Np.v.max_ovr = 2*max(DI_gen_var.Np.v);
 ax_set_lut.Np.v.min_ovr = min(DI_gen_var.Np.v);
 ax_set_lut.Np.v.max_lim = 'const';
 ax_set_lut.Np.v.min_lim = 'error';
 ax_set_lut.ssr.v.scale = 'lin';
 ax_set_lut.ssr.v.max_ovr = 2*max(DI_gen_var.ssr.v);
 ax_set_lut.ssr.v.min_ovr = min(DI_gen_var.ssr.v);
 ax_set_lut.ssr.v.max_lim = 'const';
 ax_set_lut.ssr.v.min_lim = 'error';
 ax_set_lut.A.v.scale = 'lin';
 ax_set_lut.A.v.max_ovr = 100*max(DI_gen_var.A.v);
 ax_set_lut.A.v.min_ovr = min(DI_gen_var.A.v);
 ax_set_lut.A.v.max_lim = 'const';
 ax_set_lut.A.v.min_lim = 'const';
 ax_set_lut.jitter.v.scale = 'lin';
 ax_set_lut.jitter.v.max_ovr = 100*max(DI_gen_var.jitter.v);
 ax_set_lut.jitter.v.min_ovr = min(DI_gen_var.jitter.v);
 ax_set_lut.jitter.v.max_lim = 'const';
 ax_set_lut.jitter.v.min_lim = 'const';
 ax_set_lut.noise.v.scale = 'lin';
 ax_set_lut.noise.v.max_ovr = max(DI_gen_var.noise.v);
 ax_set_lut.noise.v.min_ovr = min(DI_gen_var.noise.v);
 ax_set_lut.noise.v.max_lim = 'error';
 ax_set_lut.noise.v.min_lim = 'error';
 ax_set_lut.SFDR.v.scale = 'lin';
 ax_set_lut.SFDR.v.max_ovr = max(DI_gen_var.SFDR.v);
 ax_set_lut.SFDR.v.min_ovr = min(DI_gen_var.SFDR.v);
 ax_set_lut.SFDR.v.max_lim = 'error';
 ax_set_lut.SFDR.v.min_lim = 'const';

 rqset_lut.SFDR.u.scale = 'log';

 lut = qwtbvar('lut', jobfn, ax_set_lut, rqset_lut);
 lutfn = fullfile('alg_SFDR', 'lut.mat');
 save(lutfn, 'lut')

 % test interpolation:
 ax_interp.Np.v = 10;
 ax_interp.ssr.v = 50;
 ax_interp.A.v = 10;
```

```
    ax_interp.jitter.v = 1e-11;
    ax_interp.noise.v = 1e-7;
    ax_interp.SFDR.v = 1e-5;
    qwtbvar('interp', lutfn, ax_interp);

    %% test uncertainty estimation using lut:
    % CS_test.unc = 'guf';
    % test_waveform = qwtb('SFDR', 'gen', DI_gen);
    % DO = qwtb('SFDR', test_waveform, CS_test);
    % DO.SFDR.v
    % DO.SFDR.u
%
    %% test lut %<<<1
    %     % lutfn = qwtbvar('interp', lutfn, ax);
% end

% vim settings modeline: vim: foldmarker=%<<<,%>>> fdm=marker
   fen ft=octave textwidth=80 tabstop=4 shiftwidth=4
```

source_code_hard_links/make_lut.m

Appendix D

alg_generator.m

```
% Generator for SFDR algorithm
function [DO, DI] = alg_generator(DI)
% generates signal based on input quantities. if none given,
    basic signal is constructed.
%
% used quantities:
% f - main signal frequency, list of spurious/harmonics
    frequencies
% A - main signal amplitude, list of spurious/harmonics
    amplitudes
% ph - main signal phase, list of spurious/harmonics phases
% O - main signal offset, list of spurious/harmonics offsets
% dc - dc component
% Np - scalar real, number of main signal periods in the record
     ( Ns/fs = Np/f(1) )
% ssr - ratio of sampling to signal frequency, ssr = fs/f(1).
% SFDR - scalar, spurious free dynamic ratio
% jitter - standard deviation of jitter [s]
% noise - standard deviation of noise [V]
% smr - spurious to main signal frequency multiple
%
% calculated quantities:
% fs - scalar integer, sampling frequency
% Ns - scalar integer, record length (samples count)

    if ~exist('DI', 'var')
        DI = struct();
    end
```

```
% set initial values and randomize input quantities %<<<1
[DI, f]      = setQ(DI, 'f',      1e3,      0);
[DI, A]      = setQ(DI, 'A',        1,      0);
[DI, ph]     = setQ(DI, 'ph',       0,      0);
[DI, O]      = setQ(DI, 'O',        0,      0);
[DI, dc]     = setQ(DI, 'dc',       0,      0);
[DI, Np]     = setQ(DI, 'Np',      10,      0);
[DI, ssr]    = setQ(DI, 'ssr',    100,      0);
[DI, SFDR]   = setQ(DI, 'SFDR',   120,      0);
[DI, jitter] = setQ(DI, 'jitter',1e-12,     0);
[DI, noise]  = setQ(DI, 'noise',    0,      0);
[DI, smr]    = setQ(DI, 'smr',      5,      0);

% calculate and set other needed quantities %<<<1
% sampling frequency
fs = ssr.*f(1);
[DI, fs]     = setQ(DI, 'fs',      fs,      0);

% samples count:
Ns = fix(Np.*fs./f(1));
[DI, Ns]     = setQ(DI, 'Ns',      Ns,      0);

% add one single spurious based on SFDR and smr:
% A = [A 10^(-1.*SFDR/20)];
A = [A A./SFDR];
f = [f smr.*f];
O = [O 0];
ph = [ph 0];
DI.A.v = A;
DI.f.v = f;
DI.O.v = O;
DI.ph.v = ph;

% generate the signal %<<<1
% time series:
t = [0 : Ns-1]./fs;
% set t uncertainty as jitter:
ut = jitter.*ones(size(t));
% sampled values:
% (to save memory, use for cycle instead of matrix multipli
cation)
y = dc + zeros(size(t));
for j = 1:numel(A)
```

```
        y = y + A(j).*sin(2.*pi.*f(j).*t + ph(j)) + O(j);
    end
    % add noise to the data:
    uy = noise.*ones(size(y));

    % make output %<<<1
    DO.t.v = t;
    DO.t.u = ut;
    DO.y.v = y;
    DO.y.u = uy;

    DO.fs.v = fs;
    DO.fs.u = 0;
end

function [DI, val] = setQ(DI, Qn, v, u) %<<<1
% ensure the quantity is set in DI,
% set default values for .v and .u if missing,
% randomize quantity.
    % if quantitiy missing
    if ~isfield(DI, Qn)
        DI.(Qn).v = v;
        DI.(Qn).u = u;
    end
    % if uncertainty missing
    if ~isfield(DI.(Qn), 'u')
        DI.(Qn).u = u;
    end
    % randomize
    DI.(Qn).v = normrnd(DI.(Qn).v, DI.(Qn).u, size(DI.(Qn).v));
    val = DI.(Qn).v;
end % function setQ

% function A = harmonic_series(THD, N) %<<<1
% % Spectrum made using geometric series
% % N is number of harmonics
% % A_1 is given (value is 1)
% % A_2 is calculated from THD value in a such way that:
% % A_3..A_N is function of A_2: A_i = A_2/(i-1) for i=2..N
% % so:
% % harmonic id:      A_1     A_2     A_3     A_4     A_5
   A_N
```

```
%% harmonic number:  given   calc    A_2/2   A_2/3   A_2/4
   A_2/(N-1)
%
%     A(1) = 1;
%
%     if N > 1
%         S = sum(1./[1:N-1].^2);
%         A(2) = THD.*A(1)./sqrt(S);
%         A(2:N) = A(2)./[2-1:N-1];
%     else
%         A = 1;
%     end
%
%% selfcheck:
%% error = sum(A(2:end).^2)^0.5/A(1) - THD:
%
% end % function harmonic_series
```

source_code_hard_links/alg_generator.m

Appendix E

gen_and_calc.m

```
function [DO, DI, CS] = gen_and_calc(DI, CS)
    % generate waveform
    [waveform, DI] = qwtb('SFDR', 'gen', DI);
    % calculate waveform
    DO = qwtb('SFDR', waveform, CS);
end

% vim settings modeline: vim: foldmarker=%<<<,%>>> fdm=marker
   fen ft=octave textwidth=80 tabstop=4 shiftwidth=4
```

source_code_hard_links/gen_and_calc.m

Appendix F

alg_wrapper.m

```
function dataout = alg_wrapper(datain, calcset)
% Part of QWTB. Wrapper script for algorithm SFDR.
%
% See also qwtb

% Format input data ------------------------------ %<<<1
if isfield(datain, 'fs')
    fs = datain.fs.v;
elseif isfield(datain, 'Ts')
    fs = 1/datain.Ts.v;
    if calcset.verbose
        disp('QWTB: SFDR wrapper: sampling frequency was
   calculated from sampling time')
    end
else
    fs = 1/mean(diff(datain.t.v));
    if calcset.verbose
        disp('QWTB: SFDR wrapper: sampling frequency was
   calculated from time series')
    end
end

%  call SP-WFFT to calculate spectrum
datain.window.v = 'blackman';
cs.verbose = 0;
specDO = qwtb('SP-WFFT',datain, cs);
dataout.A = specDO.A;

% Call algorithm ------------------------------- %<<<1
```

```
% SFDR - the Spurious Free Dynamic Range in V/V. Convert to
    decibel relative to
%       carrier using euqation: SFDR in dBc is: 20*log10(SFDR)
    ;
% id_main - index of the peak of the main signal component
% id_spur - index of the peak of the highest spurious signal
    component (can be a harmonic).
[dataout.SFDR.v, id_main, id_spur] = spec_to_SFDR(dataout.A.v);
% calculate SFDR in decibel to carrier:
dataout.SFDRdBc.v = 20*log10(dataout.SFDR.v);

% Uncertainty estimation ------------------------------ %<<<1
if strcmp(calcset.unc, 'guf')
    lutfn = fullfile('alg_SFDR', 'lut.mat'); % XXX here add
    this-actual-wrapper-script-path and join with lut.mat
    % uncertainty estimate based on LUT: %<<<2
    % number of main signal periods is based on the signal
    frequency, number of
    % samples and sample frequency:
    axip.Np.v = (numel(datain.y.v)./fs) ./ (1/specDO.f.v(
    id_main));
    % ratio of sampling to signal frequency
    axip.ssr.v = fs./specDO.f.v(id_main);
    % amplitude of the main signal:
    axip.A.v = dataout.A.v(id_main);
    % time uncertainty is used for jitter:
    axip.jitter.v = std(datain.t.u);
    % sample uncertainty is used for noise:
    axip.noise.v = mean(datain.y.u);
    axip.SFDR.v = dataout.SFDR.v;

    lutfn = fullfile('alg_SFDR', 'lut.mat'); % XXX here add
    this-actual-wrapper-script-path and join with lut.mat
    % itnerpolate LUT to obtain uncertainty of SFDR:
    unc = qwtbvar('interp', lutfn, axip);
    dataout.SFDR.u = unc.SFDR.u;
end


end % function

% vim settings modeline: vim: foldmarker=%<<<,%>>> fdm=marker
    fen ft=octave textwidth=80 tabstop=4 shiftwidth=4
```

source_code_hard_links/alg_wrapper.m

F.1 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 the low-distortion signals, where the accuracy was not critical. The relative expanded uncertainty 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 near self-THD of the digitizer itself.

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 as it reduces the noise. The algorithm supports only single-ended transducer connection.

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 fig. F.1.

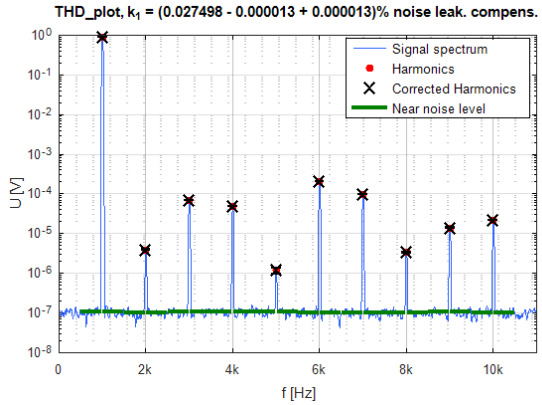


Figure F.1: Example of the TWM-THDWFFT algorithm output.

F.1.1 TWM wrapper parameters

The input quantities supported by the algorithm are shown in table F.1. Algorithm returns output quantities shown in table F.2. Calculation setup supported by the algorithm is shown in table F.3.

Table F.1: List of input quantities to the TWM-THDWFFT wrapper. Details on the correction quantities can be found in [13].

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!
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).

Table F.1: List of input quantities to the TWM-THDWFFT wrapper. Details on the correction quantities can be found in [13].

Name	**Default**	**Unc.**	**Description**
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	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_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 related to loading correction and it has effect only for RVD transducer and will work only if adc_Yin is defined as well.
tr_Zlo_Cp	1e-15	Yes	
tr_Zlo_f	[]	No	
tr_Zbuf_Rs	0	Yes	Loading corrections: Transducer output buffer output series impedance 1D table. Leave unassigned to disable buffer from the correction topology.
tr_Zbuf_Ls	0	Yes	
tr_Zbuf_f	[]	No	
tr_Zca_Rs	1e-9	Yes	Loading corrections: Transducer high side terminal series impedance 1D table.
tr_Zca_Ls	1e-12	Yes	
tr_Zca_f	[]	No	
tr_Zcal_Rs	1e-9	Yes	Loading corrections: Transducer low side terminal series impedance 1D table.
tr_Zcal_Ls	1e-12	Yes	
tr_Zcal_f	[]	No	
tr_Yca_Cp	1e-15	Yes	Loading corrections: Transducer output terminals shunting impedance.
tr_Yca_D	1e-12	Yes	
tr_Yca_f	[]	No	
tr_Zcam	1e-12	Yes	Loading corrections: Transducer output terminals mutual inductance 1D table.
tr_Zcam_f	[]	No	

Table F.1: List of input quantities to the TWM-THDWFFT wrapper. Details on the correction quantities can be found in [13].

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 F.2: List of output quantities of the TWM-THDWFFT wrapper. Note the uncertainty "No" means the algorithm may return some uncertainty but it should be ignored because it is either incomplete or not validated.

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

Table F.3: 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.

F.1.2 Algorithm description and uncertainty evaluation

The whole algorithm is extended and improved version of the THD analyser presented in [14]. The overview of the algorithm wrapper structure and internal functions is shown in fig. F.2. The wrapper start by a call to the top level function "thd_wfft()", which performs entire calculation and uncertainty estimation. Next, the wrapper may optionally plot graph showing the identified harmonics and near spectrum. Note the wrapper reduces the asymmetric uncertainty limits to symmetric as the TWM was not designed for such a case. This has no effect when the level of harmonics is at least twice the noise level. It will expand the uncertainty only for very small harmonic levels near noise level.

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

The function "thd_proc_waves()" first detects fundamental frequency of each record in y. It contains several modes of detection. The simplest is zero crossing, however it is very unreliable. Another options if FPNLSF [15], which may fail when initial estimate from zero cross detector is poor. Last and best option (default) is PSFE [16], 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 of side lobes was chosen for the goal - Flattop HFT248D from [17]. This window offer side lobes suppression by 248 dB and scalloping error only 0.0104 % for range ± 0.5 DFT bin.

The internal structure of the evaluation function "thd_eval_thd()" is shown in fig. F.3. 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

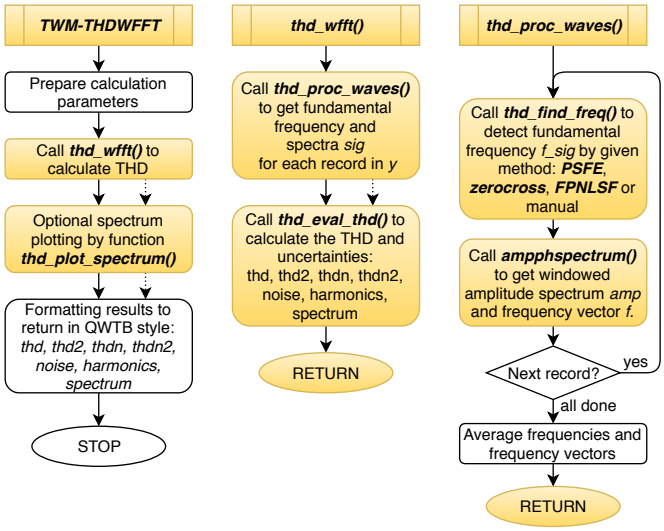


Figure F.2: Flow chart of the algorithm wrapper TWM-THDWFFT. Note the rounded gold blocks are calls to other local functions which are shown in another diagram or mentioned in the text.

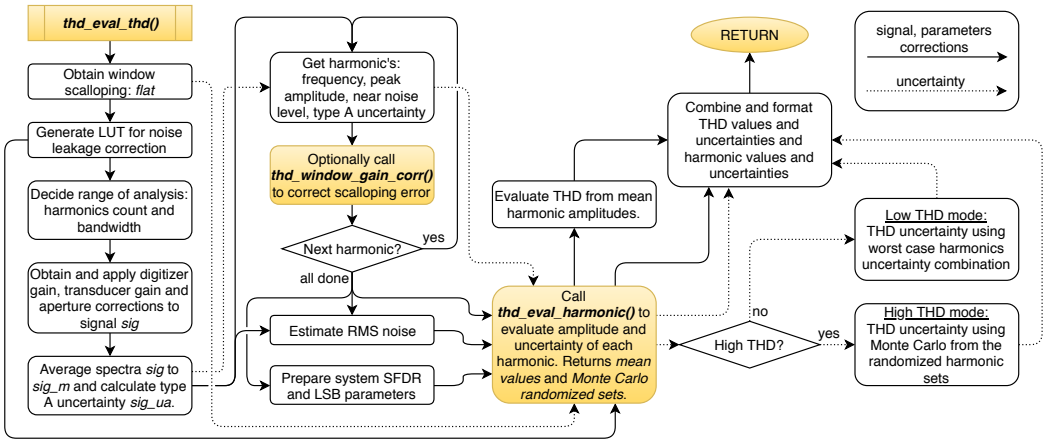


Figure F.3: Flow chart of the main algorithm function "thd_eval_thd()" for the TWM-THDWFFT algorithm.

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 frequency 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:

$$thd = \frac{\sqrt{U_2^2 + U_3^2 + \cdots + U_M}}{U_1}, \tag{F.1}$$

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:

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

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

$$thdn = \frac{\sqrt{U_2^2 + U_3^2 + \cdots + U_M + U_{\text{noise}}^2}}{U_1}, \tag{F.3}$$

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

$$thdn2 = \frac{\sqrt{U_2^2 + U_3^2 + \cdots + U_M^2 + U_{\text{noise}}^2}}{\sqrt{U_1^2 + U_2^2 + U_3^2 + \cdots + U_M^2 + U_{\text{noise}}^2}}. \tag{F.4}$$

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 cases occur:

1. $is\_high = true$: The distribution of the uncertainty of the harmonics is near Gaussian so the randomized amplitudes from step 8) are passed to the THD formulas above and the THD is evaluated using Monte Carlo and function "scovint()" (follows GUM guide [2]).

2. $is\_high = false$: The distribution of the uncertainty of the harmonics is very asymmetric, so the Monte Carlo would lead to large bias in the mean value of THD. Therefore the THD uncertainty is evaluated using the worst case combination of the harmonic uncertainties from step 8):

$$[thd_{\text{MAX}}, thd_{\text{MIN}}] = \left[\frac{\sqrt{\sum_{m=2}^{M} U_{m_{\text{MAX}}}^2}}{U_{1_{\text{MIN}}}}, \frac{\sqrt{\sum_{m=2}^{M} U_{m_{\text{MIN}}}^2}}{U_{1_{\text{MAX}}}} \right] \tag{F.5}$$

where:

$$U_{m_{\text{MAX}}} = U_m + U_+(U_m), \tag{F.6}$$
$$U_{m_{\text{MIN}}} = U_m - U_-(U_m). \tag{F.7}$$

The reported asymmetric uncertainties were calculated according to:

$$[U_+(thd), U_-(thd)] = [thd_{\text{MAX}} - thd, thd - k_{\text{MIN}}]\,. \tag{F.8}$$

The evaluation of the uncertainty of each harmonic is performed by the function "thd_eval_harmonic()" shown in fig. F.4. 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 wrapper). The performance is acceptable as long as no more than 50 harmonics are analysed. The resulting randomised set of harmonic amplitudes

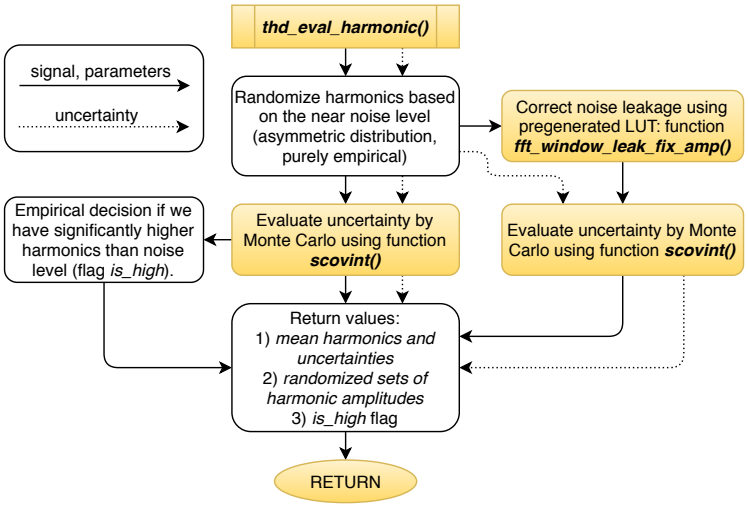


Figure F.4: Flow chart of the function "thd_eval_harmonic()" of the TWM-THDWFFT algorithm.

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]).

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:

$$U_{\text{x}} = \frac{1}{I}\sum_{i=1}^{I}\left|U_m + U_{\text{noise}}\sum_{k=1}^{K} W_k \cdot \text{e}^{-j2\pi \text{R}(i,k)}\right| \tag{F.9}$$

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. F.5.

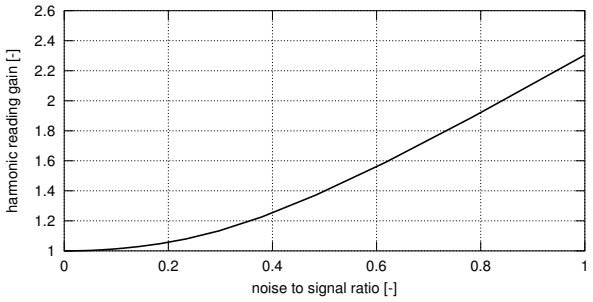


Figure F.5: Error of the harmonics amplitude measurement using FFT with window Flattop HFT248D. Note the "noise" means amplitude of the noise in surrounding DFT bins, not RMS noise.

.

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. F.5). 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. F.6.

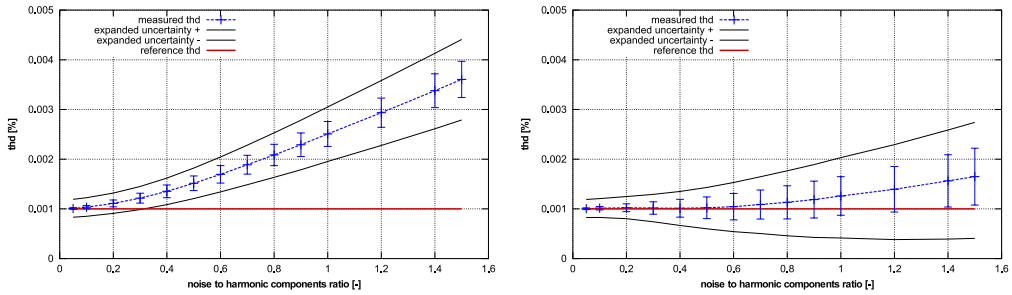


Figure F.6: Deviation of THDWFFT algorithm from simulated THD level 10 ppm for various noise to higher harmonic ratios. The simulated waveform has 10 harmonic components with amplitudes $U_\mathrm{m} = \{0.9, 3\cdot10^{-6}, 3\cdot10^{-6}, \dots\}$ V. Left graph shows results without noise spectral leakage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

F.1.3 Validation

The algorithm TWM-THDWFFT has many input quantities (45) and some of them are matrices. That is too many possible degrees of freedom. Thus, varying the quantities in some systematic way would be very complicated if the validation should cover full range of used signals and corrections. Therefore, an alternative approach was used.

QWTB test function "alg_test.m" was created, which performs the validation using randomly generated test setups. It randomizes the signal parameters, correction quantities and uncertainties and algorithm configurations in ranges expected to occur during the real measurements. The test is run many times to cover full operating range of the algorithm. Following operations are performed for each random test setup:

1. Generate signal y with known harmonic content $A_{\text{ref}}(h)$ and thus known THD thd_{ref}.

2. Distort the signal y by inverse corrections, i.e. simulate the transducers, and digitizer (e.g. gain errors, quantisation, SFDR ...).

3. Run the algorithm TWM-THDWFFT with enabled uncertainty evaluation to obtain the harmonic levels $A_{\text{x}}(h)$, distortion thd_{x} and their uncertainties $u(A_{\text{x}}(h))$ and $u(thd_{\text{x}})$.

4. Compare the reference and calculated harmonics and distortion and check if the deviations are lower than assigned uncertainties:

$$pass\_A(i,h) = |A_{\text{ref}}(h) - A_{\text{x}}(h)| < u(A_{\text{x}}(h)), \tag{F.10}$$
$$pass\_thd(i) = |thd_{\text{ref}} - thd_{\text{x}}| < u(thd_{\text{x}}), \tag{F.11}$$

   where i is test run index.

5. Repeat N times from step 1, with the same test setup parameters, but with corrections randomised by their uncertainties, and with randomised noise, SFDR and jitter.

6. Check that at least 95 % of $pass\_A(i,h)$ and $pass\_thd(i)$ results passed (for 95 % level of confidence).

The test runs count per test setup was set to $N = 300$, which is far from optimal infinite set, but due to the computational requirements it could not have been much higher. Note the low count of test induces uncertainty to the obtained pass rates.

The algorithm in the uncertainty estimation mode was tested in 4 different configurations with 10000 test setups per each. I.e. the algorithm was ran 12 million times in total (4x10000x300). The processing itself was performed on a supercomputer [18] so it took about 3 days at 400 parallel octave instances.

The randomization ranges of the signal are shown in table F.4. The randomization ranges of the corrections are shown in table F.5.

The test results were split into several groups given by the randomiser setup: (i) Scalloping correction enabled/disabled; (ii) Randomisation of corrections by uncertainty enabled/disabled. When the randomisation of corrections is disabled, the test runs cover only the algorithm itself and the contributions of the correction uncertainties are ignored.

The summary of the validation test results is shown in table F.6. The success rate was 100 % for all cases.

Table F.4: Validation range of the signal for TWM-THDWFFT algorithm.

Parameter	**Range**
Sampling rate	30 to 70 kHz (no need to randomize in wider range, as all other parameters are generated relative to this rate).
Sampling time	0.3 to 5 seconds.
Fundamental frequency	Random, so there are at least 30 DFT bins between harmonics and the highest harmonic is no higher then $0.4 \cdot fs$.
Analysed harmonics count	5 to 10.
Fundamental amplitude	0.1 to 0.9 of fullscale digitizer input.
Harmonic amplitudes	Each harmonic is randomised from 1 µV to A_max of fundamental, where the A_max is randomised from 0.0001 to 0.1 of fundamental.
Phase angles	Random for all harmonics.
Averaging cycles	10.
SFDR	-140 to -80 dBc, all spurs have the same level.
Digitizer RMS noise	1 to 50 µV.
Sampling jitter	1 to 100 ns.

Table F.5: Validation range of the correction for the TWM-THDWFFT algorithm.

Parameter	**Range**
Transducer type	Random 'shunt' or 'rvd'.
Nominal input range	0.1 to 100 V (0.1 to 100 A)
Aperture	1 ns to 100 µs
Digitizer gain	Randomly generated frequency transfer simulating NI 5922 FIR-like gain ripple (possibly the worst imaginable shape) and some ac-dc dependence. The transfer matrix has up to 50 frequency spots. Nominal gain value is random from 0.95 to 1.05 with uncertainty 2 µV/V. Maximum ac-dc value at $fs/2$ is up to ± 1 % with uncertainty 50 µV/V. Gain ripple amplitude is random from 0.005 to 0.03 dB with up to 5 periods between 0 and $fs/2$.
Digitizer SFDR	Value based on table F.4.
Transducer SFDR	Value based on table F.4. Note the "SFDR" from table F.4 is randomly split between digitizer and transducer SFDR correction.
Digitizer bit resolution	16 to 28 bits.
Digitizer nominal range	1 V
Transducer gain	Randomly generated frequency transfer. The transfer matrix has up to 50 frequency spots. Nominal gain value is random (see above) with relative uncertainty 2 µV/V. Maximum ac-dc value at $fs/2$ is up to ± 2 % with uncertainty 50 µV/V. Gain ripple amplitude is 0.005 dB with 4 to 10 periods between 0 and $fs/2$.

Table F.6: Validation results of the algorithm TWM-THDWFFT. The "passed test" shows percentage of passed tests under conditions defined in tables F.4 and F.5.

Scallop. fix.	**Rand. corr.**	**Passed test [%]**		
		thd	$h(1)$	$h(2..n)$
no	no	100.00	100.00	100.00
	yes	100.00	100.00	100.00
yes	no	100.00	100.00	100.00
	yes	100.00	100.00	100.00

F.2 TWM-MFSF - Multi-Frequency Sine Fit

TWM-MFSF is an algorithm for estimating the frequency, amplitude, and phase of the fundamental and harmonic components in a waveform. Amplitudes and phases of harmonic components are adjusted to find minimal sum of squared differences between sampled signal and multi-harmonic model. When all sampled signal harmonics are included in the model, the algorithm is efficient and produces no bias. It can even handle aliased harmonics, if they are not aliased back exactly at frequencies where other harmonics are already present. Further, it can also handle non harmonic components, when their frequency ratio to the fundamental frequency is exactly known a-priori. It is based on the [19] and [9].

The TWM wrapper TWM-MFSF is equipped with a Monte Carlo uncertainty calculator and also a fast uncertainty estimator limited for certain types of signal and algorithm setup.

F.2.1 TWM wrapper parameters

The input quantities supported by the algorithm are shown in table F.7. Algorithm returns output quantities shown in table F.8. Calculation setup supported by the algorithm is shown in table F.9.

Table F.7: List of input quantities to the TWM-MFSF wrapper.

Name	Default	Unc.	Description
fest	0	N/A	Initial estimate of fundamental frequency [Hz]. Options:
ExpComp	N/A	N/A	List of relative frequencies of the harmonic components to fit (e.g. [1, 2, 4, 3.3] means to fit fundamental, 2nd and 4th harmonic and interharmonic $3.3 \cdot f0$).
H	3	N/A	Alternative to $ExpComp$. Defines number of harmonics to fit, i.e. 3 means to fit fundamental, 2nd and 3rd harmonic.
CFT	3.5e-11	N/A	Cost Function Threshold for the MFSF minimising algorithm. Note the uncertainty estimator was calculated for the default value only!.
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	N/A	No	Input sample data vector.

Table F.7: List of input quantities to the TWM-MFSF wrapper.

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.
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.
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).

Table F.7: List of input quantities to the TWM-MFSF wrapper.

Name	Default	Unc.	Description
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 related to loading correction and it has effect only for RVD transducer and will work only if adc_Yin is defined as well.
tr_Zlo_Cp	1e-15	Yes	
tr_Zlo_f	[]	No	
tr_Zbuf_Rs	0	Yes	Loading corrections: Transducer output buffer output series impedance 1D table. Leave unassigned to disable buffer from the correction topology.
tr_Zbuf_Ls	0	Yes	
tr_Zbuf_f	[]	No	
tr_Zca_Rs	1e-9	Yes	Loading corrections: Transducer high side terminal series impedance 1D table.
tr_Zca_Ls	1e-12	Yes	
tr_Zca_f	[]	No	
tr_Zcal_Rs	1e-9	Yes	Loading corrections: Transducer low side terminal series impedance 1D table.
tr_Zcal_Ls	1e-12	Yes	
tr_Zcal_f	[]	No	
tr_Yca_Cp	1e-15	Yes	Loading corrections: Transducer output terminals shunting impedance.
tr_Yca_D	1e-12	Yes	
tr_Yca_f	[]	No	
tr_Zcam	1e-12	Yes	Loading corrections: Transducer output terminals mutual inductance 1D table.
tr_Zcam_f	[]	No	
Zcb_Rs	1e-9	Yes	Loading corrections: Cable series impedance 1D table.
Zcb_Ls	1e-12	Yes	
Zcb_f	[]	No	
Ycb_Rs	1e-15	Yes	Loading corrections: Cable series impedance 1D table.
Ycb_Ls	1e-12	Yes	
Ycb_f	[]	No	

Table F.8: List of output quantities of the TWM-MFSF wrapper. The quantities marked * may have partial or none assigned uncertainty depending on the selected uncertainty calculation mode. They will be available only for Monte Carlo uncertainty method.

Name	**Uncertainty**	**Description**
f	Yes	Vector of frequencies of all fitted components [Hz].
A	Yes	Vector of amplitudes of all fitted components.
ph	Yes*	Vector of phases of all fitted components [rad].
thd	Yes	Total harmonic distortion of the fitted components [%]. Note it is a fundamental referenced value.

Table F.9: List of “calcset” options supported by the TWM-MFSF 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 job sharing 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 server processes. Example: “calcset.mcm.user_fun = @coklbind2”. Leave empty to not execute any function.
calcset.loc	Level of confidence [-].
calcset.verbose	Verbose level.
calcset.dbg_plots	Non-zero value shows debugging plots of the MFSF uncertainty calculator.

F.2.2 Algorithm description

Internal structure of the TWM-MFSF wrapper is shown in the fig. F.7. The wrapper supports only single-ended input, so the signal conditioning is simple. The wrapper starts by a call of the QWTB algorithm "MFSF" to calculate the estimates of the harmonics. This call is performed with uncertainty option disabled, because at this point the required parameters for its calculation are not know.

Follows correction of the timebase frequency error. Next, the DC offset of the digitizer is corrected. In the next step, the wrapper compensates the aperture error, digitizer gain and phase errors and transducer gain and phase errors. At the same time the uncertainties of the corrections are calculated.

Next, the uncertainty calculator/estimator takes place. First, the required parameters for the calculation are prepared: jitter, system SFDR and digitizer resolution. Then, the wrapper calls the QWTB "MFSF" algorithm for the second time, but this time with enabled uncertainty calculation. Returned uncertainties are scaled by the correction factors so they match the scaled estimates. Next, the algorithm uncertainties are combined with the correction uncertainties and the required quantities are expressed and returned.

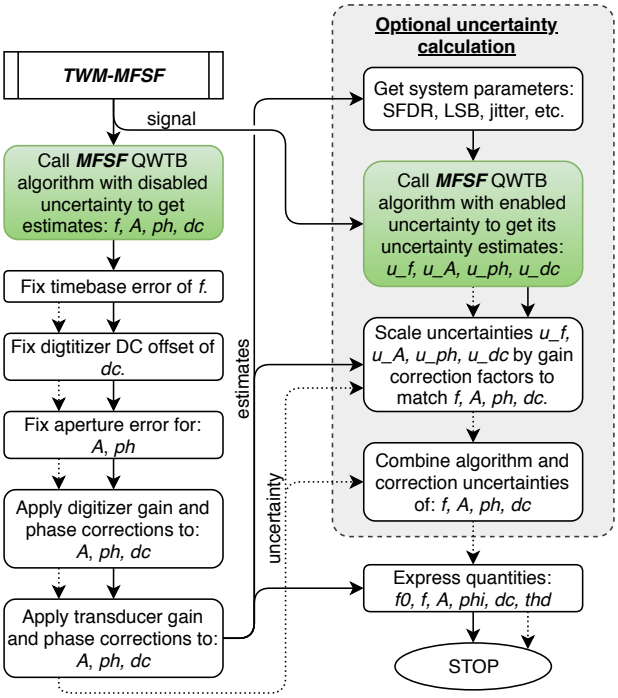


Figure F.7: Structure of TWM-MFSF algorithm wrapper. Note the green blocks are calls to another QWTB wrappers.

F.2.2.1 QWTB algorithm MFSF

The structure of the QWTB wrapper "MFSF", which contains the fitting function "MFSF()" itself is shown in fig. F.8. The wrapper starts with optional override of the internal initial estimator of fundamental component frequency by function "ipdft_spect()". Follows the call of the "MFSF()" function itself. The function returns fitted harmonic coefficients f, A, ph and offset O. It also calculated Total Harmonic Distortion (THD) following the "fundamental referenced" definition:

$$THD = \sqrt{\frac{\sum_{h=2}^{H} A(h)^2}{A(1)}}, \tag{F.12}$$

where h is harmonic index and H is harmonics count.

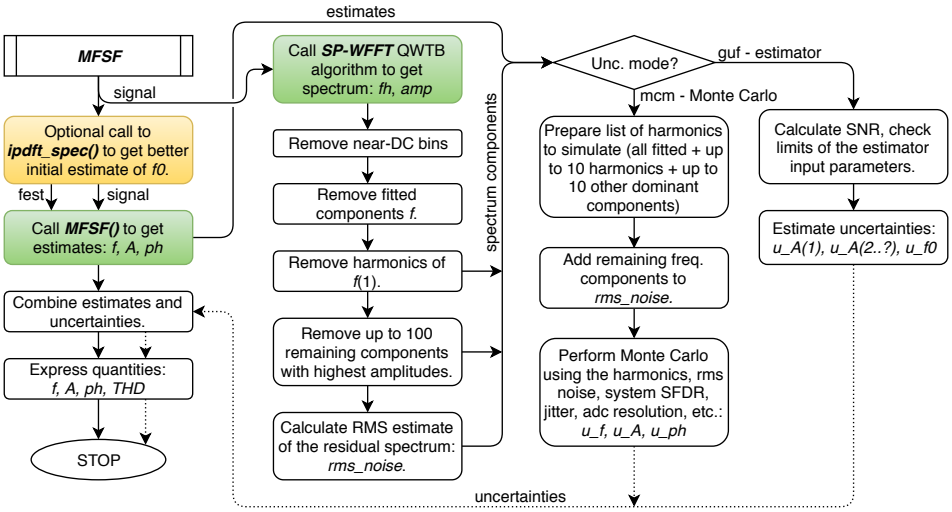


Figure F.8: Structure of MFSF algorithm wrapper. Note the green blocks are calls to another QWTB wrappers, the gold cells are calls to another functions described in the text.

The Multi-Frequency Sine-Fit algorithm itself (function "MFSF()") is used to estimate the harmonic components that are present in non-coherently sampled periodic signal. The main input parameter is the sampled record $y(n \cdot T_S)$ having the length N, the sampling period T_S and the index signal harmonics to be estimated $k = [1, h]$. Optionally, the method for initial guess estimation and the cost function threshold can be defined (the default value for the threshold is $3.5 \cdot 10^{-11}$). The outputs of the algorithm are: (i) frequency of the fundamental signal f1, (ii) amplitudes $A1$ to Ah and (iii) the phases ϕ_1 to ϕ_h of the analysed fundamental signal and harmonics, (iv) offset of the sampled signal A_0, (v) total harmonic distortion THD, (vi) total number of iterations and (vii) variance amplitude estimate.

The frequency of the fundamental signal f_1, and complex amplitudes $A_{\text{comp},k}$ are estimated first by nonlinear-least-square algorithm which iteratively minimize the K_{NLS} function (equation F.13) using Gauss-Newton procedure [20]. The first approximate frequency of the record y is estimated using either peak amplitude DFT bin frequency or interpolated DFT frequency estimate.

$$K_{\text{NLS}}(A_{comp,0}, A_{comp,1}, ..., A_{comp,h}, f_1) = \sum_{n=1}^{N} \left(y(n \cdot T_{\text{S}}) - \sum_{k=-h}^{h} A_k \cdot \exp^{j \cdot k \cdot n \cdot 2 \cdot \pi \cdot f_1 \cdot T_{\text{S}}} \right)^2 \quad \text{(F.13)}$$

$$A_{-k} = A_k^*. \quad \text{(F.14)}$$

After the complex harmonic amplitudes $A_{\text{comp},k}$ are defined the amplitudes A_k and the phases ϕ_k of the fundamental signal and harmonic components as well as the offset A_0 and the THD of the record are calculated using following equations:

$$A_k = \sqrt{A_{\text{comp,real},k}^2 + A_{\text{comp,imag},k}^2}, k \in [1, h], \quad \text{(F.15)}$$

$$\phi_k = \arctan \frac{A_{\text{comp,imag},k}}{A_{\text{comp,real},k}}, k \in [1, h], \quad \text{(F.16)}$$

$$A_0 = A_{\text{comp},0}, \quad \text{(F.17)}$$

$$THD = \frac{\sum_{k=2}^{h} A_k^2}{A_1^2}. \quad \text{(F.18)}$$

F.2.2.2 Uncertainty calculation

The TWM-MFSF supports two modes of uncertainty calculation. First option is the Monte Carlo mode, which is slower, but more accurate and it can handle any number of fitted components. Second option is fast estimator, which is less accurate, but considerably faster.

Note the uncertainty calculation is split between the "TWM-MFSF" wrapper and "MFSF" wrapper as shown in fig. F.7. The uncertainty of the algorithm is calculated in the "MFSF" wrapper, whereas the uncertainty of the corrections is included in the TWM wrapper "TWM-MFSF".

First part of the uncertainty calculation is in the "MFSF" wrapper and it is common for both modes of calculation . The spectrum analysis of the input signal is performed by the "SP-WFFT" algorithm with the windowing function "Flatttop HFT116D" [17], which has low scalloping and good spectral resolution. The spectrum is heuristically analysed:

1. The fitted components are removed from the spectrum. These are not relevant for the uncertainty evaluation, as they are already known from the "MFSF()" function itself, but they must be removed from the spectrum before searching the additional frequency components.

2. All harmonics of the fundamental frequency "f0" exceeding the threshold relative to the fundamental component are identified and removed in a full bandwidth.

3. Up to 100 residual components (harmonic or inter-harmonic) exceeding the threshold relative to the fundamental component are identified and removed in a full bandwidth.

4. The residual signal is taken as RMS noise.

Following steps differ for the Monte Carlo mode and estimator.

F.2.2.2.1 Monte Carlo

The Monte Carlo would be extremely slow if all harmonics and inter-harmonics are taken into account, because in fact it takes longer to synthesize the waveform with all the frequency components than to apply MFSF algorithm. So, before Monte Carlo itself, a selection of the dominant components is performed. All fitted components are simulated, up to 10 harmonics of "f0" are simulated and 10 of the remaining harmonic and inter-harmonics with highest amplitudes are simulated. The rest of the components identified from the spectrum is added to the RMS noise and simulated together as a noise.

The Monte Carlo (MC) simulation itself is performed by the function "proc_MFSF()", which is called once for each MC iteration cycle. The function does following steps:

1. Randomize fundamental frequency $f0$ in a small range ± 0.001 Hz/Hz to prevent accidental lock in some local minimum of uncertainty.

2. Generate time vector with the jitter effect.

3. Generate list of fitted harmonics and randomise their amplitudes by ± 1 % to reflect fitted amplitude uncertainty. Generate random phase angles of the harmonics, because it is not easy to state what was accuracy of the fit. This should produce the worst case errors.

4. Randomise the fitted harmonics by system SFDR.

5. Generate additional harmonics of the $f0$, based on the identified list from the spectrum. Randomize their amplitudes by ± 1 % and generate random phase.

6. Generate inter-harmonics based on the spectral analysis. Randomise frequency by ± 1 DFT bin to reflect resolution of FFT spectrum, amplitude by 1 % and generate random phase.

7. Synthesize waveform with all the harmonics and inter-harmonics.

8. Add RMS noise.

9. Add random offset with very pessimistic uncertainty, because MFSF may not estimate the DC correctly, when not all harmonics are in the fitted list.

10. Perform quantisation of the waveform.

11. Call "MFSF()" to get estimates of f, A, ph and O.

12. Compare the estimates to the actually generated parameters.

The results from the iterations are processed according to the GUM Annex 1 [2] using function "scovint()" to get uncertainties of the estimated components.

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 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" [18] which requires to call a special script to assign server processes to particular CPU cores.

See table F.9 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, 3 fitted harmonics and 10 additional spur harmonics is typically below 20 seconds. However, the situation may change drastically when more harmonics is fitted or high count of spur harmonic components is presents in the signal.

F.2.2.2.2 Fast estimator

The MFSF algorithm estimates several output parameters therefore the uncertainty was analysed for the frequency and the amplitude of the fundamental signal f_1 and A_1, and for the amplitudes of the other harmonic components A_2 to A_h. The phases, the offset A_0 and the THD are additional informative parameters calculated by the MFSF algorithm, therefore the uncertainty analysis for those parameters was not performed.

Three uncertainty contributions were considered in this study (see Table F.10): resolution, jitter and noise. Additionally, several other parameters related to the sampled signal or sampling (i.e. condition) are expected to affect the uncertainty therefore enormous number of Monte Carlo simulation would be needed for accurate uncertainty analysis.

Table F.10: A list of parameters that were varied during the Monte-Carlo simulations.

Uncertainty contribution	**Variation range**	**Reference value**
RMS jitter	1 ns - 10 ns	**1 ns** (0 ns)
resolution	10 pV - 100 mV	**10 µV** (0 V)
noise, SNR[*1]	10^2 - 10^6	**1000** (infinite)
Condition parameters		
amplitude of the fundamental signal, $A1$	0.1 V – 1000 V	1 V
frequency of the fundamental signal, $f1$	10 Hz – 200 Hz	100 Hz
SFDR[*2]	0 – 0.5	0.1
sampling frequency, f_s	5 kHz – 200 kHz	10 kHz
number of samples, N	500 Sa – 100 kSa	10 kSa

[*1] SNR in this study is defined as an amplitude of the fundamental signal vs. the RMS noise ratio.

[*2] SFDR is spurious-free dynamic range which is defined as the harmonic amplitude to fundamental signal amplitude ratio.

Herein, different and slightly simplified approach was used. We run 18 different Monte Carlo simulation sets. For each set only one uncertainty contribution was considered using the bold reference value given in Table F.10. The other two uncertainty contributions were neglected by using the reference values given in the brackets. Additionally, only one condition parameter has been varied at the time using the variation range as defined in Table F.10 while we used the reference values for the other condition parameters. We also verified the linearity of uncertainty contribution by varying its value over a certain variation range while neglecting the other uncertainty contributions (by using the reference values given in brackets) and keeping all condition parameters at reference values. For each combination of uncertainty contribution, condition and variation range we performed 25000 simulation where one additional harmonic component has been randomly chosen between 2nd and 10th components. Additionally, the initial

phases of the fundamental signal and harmonic component have been randomly varied between $+\pi$ and $-\pi$. For each simulation a Gaussian distribution has been obtained. The uncertainty contribution (Gaussian distribution, $k = 1$) for each estimated parameter (i.e. f_1, A_1, A_k) due to the resolution, noise and jitter are defined by equations F.22 to F.30. The uncertainty contributions for each estimated parameter are finally combined, and recalculated for Gaussian distribution, $k = 2$:

$$u_{f_1} = 2 \cdot \sqrt{u^2_{f_1,\text{res}} + u^2_{f_1,\text{noise}} + u^2_{f_1,\text{jitter}}}, \tag{F.19}$$

$$u_{A_1} = 2 \cdot \sqrt{u^2_{A_1,\text{res}} + u^2_{A_1,\text{noise}} + u^2_{A_1,\text{jitter}}}, \tag{F.20}$$

$$u_{A_h} = 2 \cdot \sqrt{u^2_{A_h,\text{res}} + u^2_{A_h,\text{noise}} + u^2_{A_h,\text{jitter}}}. \tag{F.21}$$

$$u_{f,\text{res}} = 0.52\,\text{mHz} \cdot \left(\frac{f_\text{S}}{10\,\text{kHz}}\right)^{1.6} \cdot \left(\frac{N}{10\,\text{kSa}}\right)^{-2} \cdot \left(\frac{res}{A_1}\right), \tag{F.22}$$

$$u_{A_1,\text{res}} = 0.5 \cdot \left(\frac{f_1}{f_\text{S}}\right)^{0.5} \cdot res, \tag{F.23}$$

$$u_{A_h,\text{res}} = 1.3 \cdot \left(\frac{f_1}{f_\text{S}}\right)^{0.5} \cdot res, \tag{F.24}$$

$$u_{f,\text{noise}} = 5.5\,\mu\text{Hz} \cdot \left(\frac{f_\text{S}}{10\,\text{kHz}}\right) \cdot \left(\frac{N}{10\,\text{kSa}}\right)^{-1.5} \cdot \left(\frac{SNR}{1000}\right)^{-1}, \tag{F.25}$$

$$u_{A_1,\text{noise}} = 10\,\mu\text{Hz} \cdot \left(\frac{N}{10\,\text{kSa}}\right)^{-0.5} \cdot \left(\frac{A_1}{1\,\text{V}}\right)^{1} \cdot \left(\frac{SNR}{1000}\right)^{-1}, \tag{F.26}$$

$$u_{A_h,\text{noise}} = 25\,\mu\text{Hz} \cdot \left(\frac{N}{10\,\text{kSa}}\right)^{-0.5} \cdot \left(\frac{A_1}{1\,\text{V}}\right)^{1} \cdot \left(\frac{SNR}{1000}\right)^{-1}, \tag{F.27}$$

$$u_{f,\text{jitter}} = 1\,\mu\text{Hz} \cdot \left(\frac{f_\text{S}}{10\,\text{kHz}}\right)^{1.2} \cdot \left(\frac{N}{10\,\text{kSa}}\right)^{-1.7} \cdot \left(\frac{f_1}{100\,\text{Hz}}\right)^{0.55} \cdot \left(\frac{jitter}{1\,\text{ns}}\right)^{1.2} \tag{F.28}$$

$$u_{A_1,\text{jitter}} = 2.1\,\mu\text{V} \cdot \left(\frac{A_1}{1\,\text{V}}\right)^{1} \cdot \left(\frac{f_1}{100\,\text{Hz}}\right)^{1} \cdot \left(\frac{jitter}{1\,\text{ns}}\right)^{1}, \tag{F.29}$$

$$u_{A_h,\text{jitter}} = 5\,\mu\text{Hz} \cdot \left(\frac{N}{10\,\text{kHz}}\right)^{-0.5} \cdot \left(\frac{A_1}{1\,\text{V}}\right)^{1} \cdot \left(\frac{f_1}{100\,\text{Hz}}\right)^{1} \cdot \left(\frac{jitter}{1\,\text{ns}}\right)^{1}. \tag{F.30}$$

F.2.3 Validation

The algorithm TWM-MFSF has many input quantities and some of them are matrices. That is too many possible degrees of freedom. Thus, varying the quantities in some

systematic way would be very complicated if the validation should cover full range of used signals and corrections. Therefore, an alternative approach was used.

QWTB test function "alg_test.m" was created, which performs the validation using randomly generated test setups. It randomizes the signal parameters, correction quantities and uncertainties and algorithm configurations in ranges expected to occur during the real measurements. The test is run many times to cover full operating range of the algorithm. Following operations are performed:

1. Generate signal with known frequency, amplitude, phase of the fundamental fundamental and harmonics component and with a know DC offset.

2. Distort the signal by inverse corrections, i.e. simulate the transducers, and digitizer (e.g. gain errors, quantisation, SFDR ...).

3. Run the algorithm TWM-MFSF with enabled uncertainty evaluation to obtain the estimated values and corresponding uncertainties of the frequency (fundamental signal), amplitude (fundamental signal and harmonics), phase (fundamental signal and harmonics), DC and THD estimation.

4. Compare the reference and calculated values and check if the deviations are lower than assigned uncertainties.

5. Repeat N times from step 1, with different setup parameters, different corrections randomised by their uncertainties, and with randomised noise, SFDR and jitter.

6. Check that at least 95 % of results passed (for 95 % level of confidence).

Following validation applies only to the fast uncertainty estimator. The Monte-Carlo uncertainty calculator was not validated.

The total number of Monte-Carlo simulations was 200000. The parameters of the input signal, the digitizer and transducer settings were randomly varied. The sampling frequency was between 5 kHz and 200 kHz and the number of samples between 500 Sa and 100 kSa. The frequency of fundamental signal was between 10 Hz and 200 Hz. The frequency of the harmonics and interharmonics were always above frequency of the fundamental signal but below the Nyquist frequency. The number of harmonics that were added to the fundamental signal and that needs to be estimated by the algorithm was 3. The number of interharmonics was 1. The amplitude of the fundamental signal was between 0.1 V and 1000 V and the amplitude of the harmonics and interharmonic between 0.00001 and 0.05 and between 0.00001 and 0.02 of the amplitude of the fundamental signal, respectively (the amplitudes have been varied individually for each harmonics and interharmonic). The DC offset was between -10 and +10 of the amplitude of the fundamental signal. The phases of the fundamental signal as well as of the harmonics and interharmonic were individually and randomly varied between +3.14 rad and -3.14 rad.

The ADC noise was between 1e-11 and 1e-3 of the amplitude of the fundamental signal while the jitter was between 1e-9 s and 1e-7 s. Additionally, the spur has been added to the signal (spurious free dynamic range was 100e-6, number of spurs 10). ADC aperture was between 1e-5 s and 4e-5 s, ADC gain between 1 and 1.5, ADC phase between +1.57 rad and -1.57 rad, frequency correction of the digitizer timebase between -5e-3 and 5e-3, ADC offset between 0.005 V and 0.005 V and number of bits between 22 and 24. Relative time-stamp of the first sample was varied between -10 s and 10 s. The transducer gain was between 0.5 and 20 and the transducer phase was between +1.57 rad and -1.57 rad. The resistive voltage divider low-side impedance value (i.e. resistance and capacitance) were between 100 Ω and 500 Ω and 0.1 pF and 10 pF, respectively (only resistive voltage divider was used in the simulations). The randomisation of corrections was also enabled which means that not only the uncertainty of the algorithm but also the contributions of the correction uncertainties were included in the Monte-Carlo simulations.

The success rate of the TWM-MFSF algorithm for the fundamental frequency estimation was 99.91 %, 99.63 % for the amplitude of the fundamental signal, 99.40 % for the amplitude of the harmonics, 99.77 % for the phase of the fundamental signal, 77.62 % for the phase of the harmonics, 68.24 % for the DC and 59.59 % for the THD.

Note the preliminary tests for the Monte Carlo method show much higher success rates at least for the harmonics, however the processing time is much higher.

Appendix G

SFDR_test.m

```
% SFDR algoritm test for a selected sinewave and with variation
    of the input
% value for the signal frequency


clear all, close all

% Generate sine wave parameters
DI.Anom.v = 1;  % signal amplitude
DI.f.v = 1e2;   % signal frequency, lower value
DI.ph.v = 0;    % signal phase
DI.O.v = 0;     % signal offset

%ADC parameters
DI.bitres.v = 28;   % bit resolution
DI.FSR.v = 2;       % full scale range

% Sampling parameters
DI.fs.v = 1e4;  % sampling frequency: 10 kHz

% Time series:
DI.t.v = [0 : 1/DI.fs.v : 1-1/DI.fs.v];

% Selection of Harmonic (spurious) and distortion level to be
   added to the signal
harm = 1.1;
distortion_dB = -40;
```

```
% Run SFRD algorithm on input data |DI|. Frequency values: 100
    Hz to 1 kHz, step = 100 Hz
i=1
for f = 100:100:1000
    DI.f.v = f;  % signal frequency, for coherent sampling or
    % DI.f.v = f*(1+10e-6); % signal frequency, for non-coherent
     sampling, with
    % deviation in frequency

    % Sampled values
    DI.y.v = ones(size(DI.t.v)).*0;
    DI.y.v = DI.Anom.v.*sin(2.*pi.*DI.f.v.*DI.t.v + DI.ph.v) +
    DI.O.v;

    % Add distortion with level and harmonic selected above
    DI.y.v = DI.y.v + DI.Anom.v*10^(distortion_dB/20)*sin(2*pi*
    harm*DI.f.v*DI.t.v + DI.ph.v) + DI.O.v;

    % Run SFDR algorithm and copy result to SFDRdBc.y vector
    DO = qwtb('SFDR', DI);
    SFDRdBc.y.v(i) = DO.SFDRdBc.v;
    i=i+1;
end
```

source_code_hard_links/SFDR_script/sfdr_example2_vmc.m

Appendix H

SFDR_repeat_test.m

```
% SFDR algoritm test for a selected sinewave with random noise
    added to each sample and
% with 1000 repetitions

clear all, close all

% Generate sine wave parameters
DI.Anom.v = 1;  % signal amplitude
DI.f.v = 1000;   % signal frequency
DI.ph.v = 0;    % signal phase
DI.O.v = 0;     % signal offset

%ADC parameters
DI.bitres.v = 28;   %bit resolution
DI.FSR.v = 2;   % full scale range

% Sampling
DI.fs.v = 1e4; % sampling frequency: 10 kHz
%DI.fs.u = 0;

% Time series:
DI.t.v = [0 : 1/DI.fs.v : 1-1/DI.fs.v];

% Selection of Harmonic (spurious) and distortion level to be
    added to the signal
harm = 1.5;
distortion_dB = -80;

% Selection of noise level to be added to the signal:
```

```
Noise = 1e-6;


% Run SFRD algorithm on input data |DI|: repeat 1000 times
i=1
for i = 1:1:1000
    % Sampled values
    DI.y.v = ones(size(DI.t.v)).*0;
    DI.y.v = DI.Anom.v.*sin(2.*pi.*DI.f.v.*DI.t.v + DI.ph.v) +
   DI.O.v;

    % Add distortion with level and harmonic selected above
    DI.y.v = DI.y.v + DI.Anom.v*10^(distortion_dB/20)*sin(2*pi*
   harm*DI.f.v*DI.t.v + DI.ph.v) + DI.O.v;

    % Add random noise to every sample
    DI.y.v = DI.y.v + normrnd(0,Noise,size(DI.y.v));

    % Run SFDR algorithm and copy result to SFDRdBc.y vector
    DO = qwtb('SFDR', DI);
    SFDRdBc.y.v(i) = DO.SFDRdBc.v;
    i=i+1;
end
```

source_code_hard_links/SFDR_script/sfdr_example3_vmc.m

Appendix I

SFDR_unc_test.m

```
% SFDR algoritm test for a selected sinewave with uncertainty
    added to each
% sample and with variation of signal frequency

clear all, close all

% calculation settings: monte carlo
CS.unc = 'mcm';
CS.mcm.repeats = 1e3;
CS.mcm.method = 'multicore';
CS.mcm.procno = 6;

% variation settings:
CS.var.dir = 'SFDR';
CS.var.dir = 'freq_estimate';
CS.var.cleanfiles = 1;

% Generated sine wave:
DI.Anom.v = 1;  % signal amplitude
DI.f.v = 1e2;   % signal frequency, lower value
DI.ph.v = 0;    % signal phase
DI.O.v = 0;     % signal offset

%ADC parameters
DI.bitres.v = 28;  %bit resolution
DI.FSR.v = 2;      % full scale range

% Sampling parameters
DI.fs.v = 1e4; % sampling frequency 10 kHz
```

```
% Time series
DI.t.v = [0 : 1/DI.fs.v : 1-1/DI.fs.v];

% Sampled values
DI.y.v = DI.Anom.v.*sin(2.*pi.*DI.f.v.*DI.t.v + DI.ph.v) + DI.O
    .v;

% Add distortion at harmonic
harm = 0.5;
distortion_dB = -90;
DI.y.v = DI.y.v + DI.Anom.v*10^(distortion_dB/20)*sin(2*pi*harm
    *DI.f.v*DI.t.v + DI.ph.v) + DI.O.v;

% Add uncertainty to every sample:
DI.y.u = ones(size(DI.y.v))*1e-4;

%Add uncertainties to each input variable: needed to run the
    qwtb fucntion called by qwtbvar
DI.Anom.u  = DI.Anom.v*0;
DI.f.u  = DI.f.v*0;
DI.ph.u = DI.ph.v*0;
DI.O.u  = DI.O.v*0;
DI.t.u = ones(size(DI.t.v)).*0;
DI.fs.u = 0;
DI.bitres.u = 0;
DI.FSR.u = 0;

% Run the SFRD algorithm on input data |DI|,with calculattion
    settings |CS| and
% with variation in f

DIvar.f.v = linspace(100, 1000, 10);

jobfn = qwtbvar('SFDR', DI, DIvar, CS);
[x, y] = qwtbvar(jobfn, 'f', 'SFDRdBc');

% Output plotting
[H, x, y] = qwtbvar(jobfn, 'f', 'SFDRdBc');
hold on
plot(xlim, -1.*[distortion_dB distortion_dB], '-')
legend('calculated SFDR','real value')
```

source_code_hard_links/SFDR_script/sfdr_example1_vmc.m

Appendix J

PosIntHist.m

```
function y = PosIntHist(x,NoBits)
% @fn PosIntHist
% @brief Creates histogram form positive integer values
% @param x The vector or matrix that conatins the positive
   integers
% @return y The histogram calculated
% @author Tamás Virosztek, Budapest University of Technology
   and Economics,
%         Department of Measurement and Infromation Systems,
%         Virosztek.Tamas@mit.bme.hu
%         modified by David Peral dpera@cem.es

[s1,s2] = size(x);
if ~isempty(find(x ~= round(x),1)) || (min(x) < 1);
    error('Input values are not positive integers');
end

y = zeros(max(max(x)),1);

y=histc(x,[1:2^NoBits]); %this command improve speed

end
```

source_code_hard_links/INL_scripts/PosIntHist.m

Appendix K

ProcessHistogramTest.m

```
function INL = ProcessHistogramTest(dsc,display_settings,
    varargin)
% @fn ProcessHistogramtest
% @brief Processes measurement descriptor using histogram test
    with
%        sinusoidal excitation signal
% @param dsc The measurement descriptor to process
% @param display_settings A struct sets the options for each
    window to
%                             appear or not
%            warning_dialog : Warning_dialog box
%            results_win: Results window
%            summary_win: summary window
% @param varargin Additional paramemetrs to be passed:
%            varargin{1} = estimate_ratio: the ratio of
    histogram bins not used to
%               estimate the amplitude and the DC component
%            varargin{2} = edge_cut: the INL values near the
    peak values
%               of the sine wave may be inaccurate according to
     the noise of
%               the measurement. These INL values will not be
    estimated.
%               edge_cutoff determines the ratio of INL values
    not to be
%               estimated
%
% @return none
```

```
% @author Tamás Virosztek, Budapest University of Technology
    and Economics,
%         Department of Measurement and Information Systems,
%         Virosztek.Tamas@mit.bme.hu
%         modified by David Peral dpera@cem.es, graphic output
    omitted

% ProcessHistogramTest (dsc,display_settings,estimate_ratio,
    edge_cut);

if (nargin == 3) %estimate_ratio is passed
    ESTIMATE_RATIO = varargin{1};
    EDGE_CUT = 0.00;
elseif (nargin == 4) %edge_cut is passed
    ESTIMATE_RATIO = varargin{1};
    EDGE_CUT = varargin{2};
else %no addtional parameters passed, or incorrect call of
    ProcessHistogramtest()
    ESTIMATE_RATIO = 0.00;
    EDGE_CUT = 0.00;
end
screensize = get(0,'ScreenSize');

%Pre-processing time domain data:
%ADC codes must be positive integers to perform PosIntHist
if (min(dsc.data) < 0)
   dsc.data = dsc.data + (0 - min(dsc.data)) ;
   warndlg('ADC codes are not nonnegative integers. Added %d to
    each code to process histogram test')%, (0 - min(dsc.data))
   );
end

if ((min(dsc.data) < 0) || (max(dsc.data) > 2^dsc.NoB - 1))
    warndlg('Mismatch between number of bits provided and ADC
   codes in the measurement record');
end
%Adding code offset +1 to process PosintHist in histogram test
dsc.data = dsc.data + 1;

h = PosIntHist(dsc.data,dsc.Nob);
nh = h/sum(h); %Normalized histogram;
nch = zeros(2^dsc.NoB,1);
acc = 0;
```

```
for k = 1:length(nh)
    acc = acc + nh(k);
    nch(k) = acc;
end
%End of the normalized cumulative histogram shall be filled
   with ones:
nch(length(nh)+1:end) = ones(length(nch) - length(nh),1);

ideal_trans_levels = linspace(0,1,(2^dsc.NoB-1)).';
%Finding transition levels assumed to be correct:
[val,ind_low] = min(abs(nch - ESTIMATE_RATIO));
[val,ind_high] = min(abs(nch - (1 - ESTIMATE_RATIO)));
if (ind_low < 1)
    ind_low = 1;
end
if (ind_high > 2^dsc.NoB - 1)
    ind_high = 2^dsc.NoB -1;
end
%Transition levels between T[m] and T[l] are used to estimate A
    and Mu
%Finding the optimal solution of these equations in least
   squares sense:
D = zeros(ind_high-ind_low+1,2);
D(:,1) = ones(ind_high-ind_low+1,1);
for k = ind_low:ind_high
    D(k-ind_low+1,2) = -cos(pi*nch(k));
end
p = inv(D.'*D)*D.'*ideal_trans_levels(ind_low:ind_high); %p = [
   mu;A] = inv(D.'*D)*D.'*T(ind_low:ind_high);

trans_levels = zeros(2^dsc.NoB-1,1);
small = 0.1/sum(h); %Effect of "0.1 sample" in the normalized
   histogram
for k = 1:length(trans_levels)
    if (abs((nch(k)) - 0) < small) % there are no code bins
   tested below this transition level
        trans_levels(k) = NaN;
    elseif (abs(nch(k) - 1) < small) %thereare no code bins
   tested above this transition level
        trans_levels(k) = NaN;
    else trans_levels(k) = p(1) - p(2)*cos(pi*nch(k));
    end
end
```

```
q = 1/(2^dsc.NoB - 2);
INL = (trans_levels - ideal_trans_levels)/q;

%Discarding uncertain values of INL near the edge of the sine
    wave
for k = 1:length(INL)
    if ((nch(k) < EDGE_CUT) || (nch(k)) > (1 - EDGE_CUT))
        INL(k) = NaN;
    end
end

%Calibrating INL values to the lowest and highest transition
    level estimated.
lowest_estimated = find(~isnan(INL),1,'first');
highest_estimated = find(~isnan(INL),1,'last');
INL_calib = [zeros(lowest_estimated-1,1);linspace(INL(
    lowest_estimated),INL(highest_estimated),highest_estimated-
    lowest_estimated+1).';zeros(length(INL)-highest_estimated,1)
    ];
INL = INL - INL_calib;
DNL = diff(INL);

%WARNING DIALOGS (if necessary)
if (display_settings.warning_dialog)
    if ((lowest_estimated ~= 1) || (highest_estimated ~= 2^dsc.
    NoB - 1))
        warndlg({'The device is not overdriven enough';...
            sprintf('Transition levels under %d and over %d
    cannot be estimated',lowest_estimated,highest_estimated);...
            'At least 120% full scale overdive is recommended';
    ...
            'Non estimated INL values vill be assumed to be 0';
    ...
            'Results of histogram test may be less accurate
    than desired'},...
            'Histogram test warning');
    end
    if (avg_sample_per_code_bin < 10)
        warndlg({sprintf('Average sample per code bin is few
    (%1.2f)',avg_sample_per_code_bin);...
            'Results of histogram test may be less accurate
    than desired'},...
```

```
            'Histogram test warning');
    end
    if (fractional_ratio > 1e-2)
        warndlg({'Ratio of samples in fractional period is too
   high';...
            sprintf('%1.3e',fractional_ratio);
            'Results of histogram test may be less accurate
   than desired'},...
            'Histogram test warning')
    end
end

%%%%%%%%Adding evaluation result to results cell array:
try
    testresults = evalin('base','adctest_process_results');
    res_len = size(testresults,1);
    %Search for existing results block
    existings_index = 0;
    for k = 1:res_len
        if strcmpi(dsc.model,testresults{k,1}.DUT.model) ...
                && strcmpi(dsc.serial,testresults{k,1}.DUT.
   serial)...
                && (dsc.channel == testresults{k,1}.DUT.channel
   )...
                && (dsc.NoB == testresults{k,1}.DUT.NoB)
            existings_index = k;
        end
    end
    if (existings_index ~= 0) %existing result struct
        %Adding new results:
        testresults{existings_index,1}.INL.max = max(INL);
        testresults{existings_index,1}.INL.min = min(INL);
        testresults{existings_index,1}.DNL.max = max(DNL);
        testresults{existings_index,1}.DNL.min = min(DNL);
    else %new result struct shall be added
        testresults{res_len + 1,1}.DUT.model = dsc.model;
        testresults{res_len + 1,1}.DUT.serial = dsc.serial;
        testresults{res_len + 1,1}.DUT.channel = dsc.channel;
        testresults{res_len + 1,1}.DUT.NoB = dsc.NoB;
        %Adding new results:
        testresults{res_len + 1,1}.INL.max = max(INL);
        testresults{res_len + 1,1}.INL.min = min(INL);
        testresults{res_len + 1,1}.DNL.max = max(DNL);
```

```
        testresults{res_len + 1,1}.DNL.min = min(DNL);
    end
    %updating adctest_process_results
    assignin ('base','adctest_process_results',testresults);

catch
    %If testresults global variable does not exist:
    testresults = cell(1,1); %creating new cell array for
   testresults
    testresults{1,1}.DUT.model = dsc.model;
    testresults{1,1}.DUT.serial = dsc.serial;
    testresults{1,1}.DUT.channel = dsc.channel;
    testresults{1,1}.DUT.NoB = dsc.NoB;
    %Adding new results:
    testresults{1,1}.INL.max = max(INL);
    testresults{1,1}.INL.min = min(INL);
    testresults{1,1}.DNL.max = max(DNL);
    testresults{1,1}.DNL.min = min(DNL);
    assignin ('base','adctest_process_results',testresults);
end
%%%%%%End of adding evaluatin results to cell array%%%%%%%%%

%Callbacks: (for histogram_summary_window)
    function OK_callback(source,eventdata)
        close(histogram_summary_window);
    end

end
```

source_code_hard_links/INL_scripts/ProcessHistogramTest.m