# A2.3.2 Development of algorithms

Following document summarizes interface between the algorithm and the QWTB toolbox [3] to which it will be integrated. Updated versions of the document will be present at the in the TWM GitHub repository [1]. Author strongly suggests to check the updates as the parameters may change once the design of the wideband setup is determined.

Version V0.7.0, 15.4.2019, Stanislav Mašláň.

## Addition of new algorithm

TWM processing module is designed to use QWTB toolbox algorithms. However, to make TWM work properly, one must follow the naming rules for input and output quantities in the newly developed algorithm, so all the data received from TWM processing module are recognized.

Each QWTB algorithm compatible with TWM processing module must contain at least following files in following file structure:

* **.\octprog\QWTB** QWTB toolbox root folder in the TWM ‘octprog’ folder.
  + **alg\_TWM**-my\_algorithm\_name Algorithm ‘**TWM**-my\_algorithm\_name ’root folder .
    - **alg\_info.m** Algorithm info definition file (see QWTB doc [3]).
    - **alg\_wrapper.m** Algorithm wrapper (call to low level functions).
    - \*.\* Auxiliary function of the wrapper.

The **red** parts are mandatory names. The **bold** prefix in the algorithm name ‘**TWM-**’ was chosen to distinguish the original QWTB algorithms that usually do not apply any corrections from the TWM algorithm extensions that do calculate corrections and their uncertainties. So it is recommended to follow this rule.

TWM always expect at least ‘alg\_info.m’ file, which contains name, short description, references and definitions of input and output quantities. The desired names of quantities are described below.

TWM then only requires main algorithm wrapper ‘alg\_wrapper.m’, which may either call some low level functions, call another QWTB algorithms or contain the algorithm code directly. It must process all the received quantities with the sample data and corrections (see below) and return quantities to be saved and displayed by TWM. It also contains estimator or Monte Carlo calculator of uncertainty.

In order to make the newly added algorithm visible in TWM, one must first add its name to the filter of supported algorithms which is located in the file:

.\octprog\qwtb\_list.info

This file is described in [6]. It must at least contain name of the algorithm in the list of supported algorithms:

*// filter of the algorithms*

type:: qwtb list

*// === list of the supported algorithms ===*

*// note: enter algorithm ID's, e.g.: PSFE, SFDR, ...*

**#startmatrix**:: list of supported algorithms

…

TWM-my\_algorithm\_name

…

**#endmatrix**:: list of supported algorithms

…

The newly integrated algorithm should automatically appear in the list of available algorithms in the QWTB processing panel of TWM.

When implementing the algorithms, user should keep in mind the algorithm should be compatible with both GNU Octave and Matlab environments, so it is recommended to not use the extended Octave syntax and when possible also not the commercial packages where no equivalent for the other environment is available. Where needed, use may use if/else statement to distinguish Matlab and Octave code to reflect the differences or eventually throw and error with information the algorithm cannot run in given environment. Example of environment dependent code:

%% Check if running in Matlab or GNU Octave:

IS\_OCTAVE = (exist('OCTAVE\_VERSION') ~= 0);

%% Check available packages

if IS\_OCTAVE

if not(exist('butter'))

pkg load signal

if not(exist('butter'))

error('This algorithm requires package `signal` from Octave Forge');

end

end

else

if not(exist('butter'))

error('This algorithm requires `Signal Processing Toolbox`');

end

end

## Input quantities

The format of the input quantities is given by the QWTB design. QWTB toolbox passes parameters to the algorithm’s wrapper function as a structure containing substructures, one for each quantity. Each quantity structure may contain several items. E.g. the values ‘**v**’, associated uncertainty ‘**u**’ if exists, etc. For more details see documentation of the QWTB [3]. Example of the input to the algorithms wrapper:

DI.Ts.v – value of sampling period

DI.Ts.u – uncertainty of sampling period

DI.y.v – input waveform data

DI.y.u – uncertainty of the input waveform data

…

Note the uncertainty **‘u’** may not be present if the quantity does not need it. E.g. window type for FFT is a string that obviously do not have uncertainty.

Rules for naming the input quantities:

1. Each algorithm will receive the predefined mandatory parameters listed in the table below.
2. Each algorithm may receive custom correction quantities from the TWM corrections loader under names defined by the user.
3. Each algorithm may have any number of custom parameter-quantities that are entered by the user on runtime, such as window type, etc.

Note the names of the custom quantities and parameters must not collide with the predefined names of the mandatory parameters!

Each algorithm will automatically receive following quantities from the TWM system:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Note** | **type** | **Description** |
| support\_diff | 3 | Integer scalar | This is special parameter that has no importance for the algorithm, but its presence in ‘alg\_info.m’ tells TWM processing module that this algorithm can accept differential input data from the transducers. |
| support\_multi\_inputs | 4 | Integer scalar | This is special parameter that has no importance for the algorithm, but its presence in ‘alg\_info.m’ tells TWM processing module that the algorithm is capable of processing more than one waveform per input channel, which is intended for processing of several repeated measurements at once. |
| Ts | 3 | Real scalar | Sampling period in [Seconds]. |
| y or  u and i | 2, 3, 4 | Real column vector(s) | Sample data. For single input channel algorithm, such as THD, only one vector ‘**y**’ will be passed.  For multichannel algorithms, such as power, two vectors are passed, the voltage and current. Both ‘**u**’ and ‘**i**’ vectors have the same size.  The samples are in [Volts] as returned by the digitizer (no transducer scaling).  Note the ‘**y**’, ‘**u**’, ‘**i**’ may have multiple columns, one per record if the algorithm supports ‘**support\_multi\_inputs**’! |
| time\_stamp | 6 | Real scalar | Relative time-stamp of the first sample of ‘**y**’ or ‘**u**’. The time-stamp is relative time to some reference event of the TWM system. In case of 5922 digitizer it is a reset of the cards. This has relevance for instance for time multiplexed measurements. To get time-shift of other channels of the system, use ‘**time\_shift\***’ values. Note this has also relevance for single input algorithms. E.g. phase measurement algorithm may use this to cancel each digitizere channel’s time shift so the estimated phase of each channel can be compared. |
| time\_shift |  | Real scalar | Timeshift between ‘**u**’ and ‘**i**’ channel in [Seconds] (*t\_i* – *t\_u*). Applies only for multichannel algorithms. |
| time\_shift\_lo | 2 | Real scalar | Timeshift between high-side and low-side channel of the differential channels in [Seconds] (*t\_hi* – *t\_lo*). |
| adc\_jitter | 1, 2 | Real scalar | Sampling jitter value [Seconds]. |
| adc\_offset | 1, 2 | Real scalar | Offset voltage of ADC. |
| adc\_aper | 3 | Real scalar | Aperture value [s] of the ADC at current settings. Note this value may not be available for some ADCs. |
| adc\_aper\_corr | 1, 2, 3, 5 | Real scalar | Non-zero value in this parameter indicates the algorithm should automatically apply gain/phase correction to compensate aperture effect. Note it will work only if ‘**adc\_aper**’ aperture time is present. |
| adc\_gain | 1, 2 | 2D real matrix | 2D matrix of the absolute gain coefficients of the digitizer in [Vout/Vin]. I.e. value 1.001 means the sample data will be multiplied by 1.001 to get corrected value. Dependent on the frequency ‘**adc\_gain\_f**’ and amplitude ‘**adc\_gain\_a**’. |
| adc\_gain\_f | 1, 2, 3 | Real column vector | Independent variable of the ‘**adc\_gain**’ containing nominal frequency in [Hertz], one item per row of ‘**adc\_gain**’. |
| adc\_gain\_a | 1, 2, 3 | Real row vector | Independent variable of the ‘**adc\_gain**’ containing nominal amplitude in [Volts], one item per column of ‘**adc\_gain**’. |
| adc\_phi | 1, 2 | 2D real matrix | 2D matrix of the absolute phase correction coefficients of the digitizer channel in [rad]. Value +12e-6 rad means the phase of harmonic component must be increased by 12e-6 rad. Note this is not interchannel phase correction! Dependent on the frequency ‘**adc\_phi\_f**’ and amplitude ‘**adc\_phi\_a**’. |
| adc\_phi\_f | 1, 2, 3 | Real column vector | Independent variable of the ‘**adc\_phi**’ containing nominal frequency in [Hertz], one item per row of ‘**adc\_phi**’. |
| adc\_phi\_a | 1, 2, 3 | Real row vector | Independent variable of the ‘**adc\_phi**’ containing nominal amplitude in [Volts], one item per column of ‘**adc\_phi**’. |
| adc\_freq |  | Real scalar | Frequency correction of the digitizer timebase. Note it is expected to have identical correction for all channels. |
| tr\_gain | 1 | 2D real matrix | 2D matrix of the absolute gain coefficients of the transducer in [Vin/Vout] for dividers or [Ain/Vout] for shunt. Dependent on the frequency ‘**tr\_gain\_f**’ and amplitude ‘**tr\_gain\_a**’. |
| tr\_gain\_f | 1, 3 | Real column vector | Independent variable of the ‘**tr\_gain**’ containing nominal frequency in [Hertz], one item per row of ‘**tr\_gain**’. |
| tr\_gain\_a | 1, 3 | Real row vector | Independent variable of the ‘**tr\_gain**’ containing nominal rms value in [Volts] or [Ampers], one item per column of ‘**tr\_gain**’. |
| tr\_phi | 1 | 2D real matrix | 2D matrix of the absolute phase correction coefficients of the transducer in [rad]. Dependent on the frequency ‘**tr\_phi\_f**’ and amplitude ‘**tr\_phi\_a**’. |
| tr\_phi\_f | 1, 3 | Real column vector | Independent variable of the ‘**tr\_phi**’ containing nominal frequency in [Hertz], one item per row of ‘**tr\_phi**’. |
| tr\_phi\_a | 1, 3 | Real row vector | Independent variable of the ‘**tr\_phi**’ containing nominal rms value in [Volts] or [Ampers], one item per column of ‘**tr\_phi**’. |
| crosstalk\_re  crosstalk\_im | ??? | Real column vectors | Complex crosstalk coefficients expressing complex transfer from ‘u’ channel to ‘i’ channel defined as: crosstalk = i/u. Crosstalk in the opposite direction is assumed to be identical. The value is dependent on the frequency ‘**crosstalk\_f**’.  **TODO: how to pass corrections for the differential mode??? Up to 4x3xN matrix? Or up to four 3xN matrices?** |
| crosstalk\_f | ???, 3 | Real column vector | Independent variable of the ‘**crosstalk\***’ containing nominal frequency in [Hertz], one item per row of ‘**crosstalk**\*’. |
| adc\_sfdr | 1, 2 | 2D real matrix | Spurious Free Dynamic Range coefficients of the digitizer channel [dBc]. The values are ratios of the fundamental amplitude to the highest spurious component, i.e. 100 dBc means highest spur is fundamental\_amplitude\*1e-5. The value is dependent on the fundamental frequency ‘**adc\_sfdr\_f**’ and amplitude ‘**adc\_sfdr\_a**’. |
| adc\_sfdr\_f | 1, 2, 3 | Real column vector | Independent variable of the ‘**adc\_sfdr**’ containing frequency of the fundamental harmonic in [Hertz], one item per row of ‘**adc\_sfdr**’. |
| adc\_sfdr\_a | 1, 2, 3 | Real row vector | Independent variable of the ‘**adc\_sfdr**’ containing amplitude of the fundamental harmonic in [Volts], one item per column of ‘**adc\_sfdr**’. |
| tr\_sfdr  tr\_sfdr\_f  tr\_sfdr\_a | 1, 3 |  | Spurious Free Dynamic Range coefficients of the transducer. Meaning is the same as for digitizer. |
| adc\_Yin\_Cp  adc\_Yin\_Gp  adc\_Yin\_f | 1, 2 | Real column vectors | Measured input capacitance ‘**adc\_Yin\_Cp**’ and conductance ‘**adc\_Yin\_Gp**’ of the digitizer channel. One row per frequency in ‘**adc\_Yin\_f**’. Note the ‘**adc\_Yin\_f**’ may be empty matrix. In such case the capacitance and resistance are not dependent on frequency. |
| tr\_type | 3 | Char string | String identifier of the connected transducer:  empty – no tran. correction will be applied  ‘shunt’ – resistive current shunt  ‘rvd’ – resistive voltage divider |
| tr\_Zlo\_Rp  tr\_Zlo\_Cp  tr\_Zlo\_f | 1 | Real column vectors | RVD low-side impedance value in Cp-Rp format, one column per frequency in ‘**tr\_Zlo\_f**’. Note the ‘**tr\_Zlo\_f**’ may be empty matrix. In such case the impedance is not dependent on frequency. Note this parameter have importance only for RVD and is part of the loading correction. |
| tr\_Zca\_Rs  tr\_Zca\_Ls  tr\_Zca\_f | 1 | Real column vectors | Effective series impedance of the transducer’s output terminals in Ls-Rs format, one row per frequency in ‘**tr\_Zca\_f**’. Note the ‘**tr\_Zca\_f**’ may be empty matrix. In such case the impedance is not dependent on frequency. |
| tr\_Yca\_Cp  tr\_Yca\_D  tr\_Yca\_f | 1 | Real column vectors | Effective shunting admittance of the transducer’s output terminals in Cp-D format, one row per frequency in ‘**tr\_Yca\_f**’. Note the ‘**tr\_Yca\_f**’ may be empty matrix. In such case the impedance is not dependent on frequency. |
| tr\_Zcal\_Rs  tr\_Zcal\_Ls  tr\_Zcal\_f | 1 | Real column vectors | Effective series impedance of the transducer’s low-side terminal in Ls-Rs format, one row per frequency in ‘**tr\_Zcal\_f**’. Note the ‘**tr\_Zcal\_f**’ may be empty matrix. In such case the impedance is not dependent on frequency. |
| tr\_Zcam  tr\_Zcam\_f | 1 | Real column vectors | Effective mutual inductance of the transducer’s output terminals, one row per frequency in ‘**tr\_Zcam\_f**’. Note the ‘**tr\_Zcam\_f**’ may be empty matrix. In such case the impedance is not dependent on frequency. |
| Zcb\_Rs  Zcb\_Ls  Zcb\_f | 1 | Real column vectors | Effective series impedance of the cable(s) between transducer and digitizer in Ls-Rs format, one row per frequency in ‘**Zcb\_f**’. Note the ‘**Zcb\_f**’ may be empty matrix. In such case the impedance is not dependent on frequency. |
| Ycb\_Cp  Ycb\_D  Ycb\_f | 1 | Real column vectors | Effective shunting admittance of the transducer’s output terminals in Cp-D format, one row per frequency in ‘**Ycb\_f**’. Note the ‘**Ycb\_f**’ may be empty matrix. In such case the impedance is not dependent on frequency. |
| Zbuf\_Rs  Zbuf\_Ls  Zbuf\_f | 1 | Real column vectors | Output impedance of buffer inserted between transducer and cable. These quantities are not passed when buffer is not used. |
| adc\_bits | 1, 2, 3 | Integer scalar | Bit resolution of the ADC of the digitizer. |
| adc\_nrng | 1, 2, 3 | Real scalar | Range of the digitizer channel in [Volts]. |
| lsb | 1, 2, 3 | Real scalar | Value of the least significant bit of the ADC [Volts]. |

**Note 1):** The parameters are defined for each channel of the measurement system. For algorithms with single input **‘y’** the names of the parameters are as defined in the table above. For multichannel algorithms which have two inputs **‘u’** and **‘i’** the parameters will be combined with prefixes defining the channel ‘**u\_**’ and ‘**i\_**’. See following example for parameter naming rules:

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **U channel parameter name** | **I channel parameter name** |
| adc\_gain | u\_adc\_gain | i\_adc\_gain |
| adc\_gain\_f | u\_adc\_gain\_f | i\_adc\_gain\_f |
| adc\_nrng | u\_adc\_nrng | i\_adc\_nrng |
| … | … | … |

**Note 2):** The TWM supports differential connection of the transducers, i.e. each transducer have two digitizer channels assigned: (i) high-side, (ii) low-side as shown in [5]. If the algorithm has input quantity ‘**support\_diff**’ and user sets the TWM to the differential mode, the TWM will pass additional quantities for the low-side of the transducer (ADC channel data and its corrections). If user of TWM sets it to single-ended mode, the TWM will pass only the single ended quantities. The naming convention:

|  |  |  |
| --- | --- | --- |
| **Single ended parameter name** | **High-side name** | **Low-side name** |
| y | y | y\_lo |
| adc\_gain | adc\_gain | lo\_adc\_gain |
| u | u | u\_lo |
| i | i | i\_lo |
| u\_adc\_gain | u\_adc\_gain | u\_lo\_adc\_gain |
| u\_adc\_gain\_f | u\_adc\_gain\_f | u\_lo\_adc\_gain\_f |
| u\_adc\_nrng | u\_adc\_nrng | u\_lo\_adc\_nrng |
| … | … | … |

Note the transducers have no additional low-side quantities! The impedance model of the transducer is made in the single ended mode and the connection cable ‘**Zcb**’ and ‘**Ycb**’ is expected to be identical for both low and high side (for simplification).

**Note 3):** These parameters have no assigned uncertainty, just value **‘v’**.

**Note 4):** The main waveform data quantities ‘**y**’, ‘**u**’ and ‘**i**’ can be either single waveforms (single record) or can be multi-record if the ‘**support\_multi\_inputs**’ is present. In case the TWM will pass the multiple records at once, it will set ‘**support\_multi\_inputs = N**’ and the ‘**y**’, ‘**u**’ and ‘**i**’ will contain **N** columns, one for each record.

**Note 5):** Integrating ADCs have large gain/phase errors as the aperture time approaches period of the sampled signal. One way to compensate it is to create ADC gain/phase calibration tables that includes this effect. However, it is more convenient to correct it by formula as it can be easily calculated and correct the residual errors using ADC gain/phase tables. The formulas algorithm should apply are: *gain\_correction* = (pi\**f*\**ta*)/sin(pi\**f*\**ta*), where *f* is analyzed frequency component and *ta* is aperture time. Phase correction is calculated as: *phase\_correction* = +pi\**f*\**ta*.

**Note 6):** The ‘**time\_stamp**’ parameter is relevant for both single and dual input algorithms. The TWM algorithms executed function ‘**qwtb\_exec\_algorithm()**’ does following. It takes interchannel timeshift correction for each of the digitizer channels: *tc = [0, t2-t1, t3-t1 , …]* and timestamps returned by the digitizer (these are in 99.999% identic for all channels but not necessarily): *ts = [ts1, ts2, ts3, …]* and it will sum the two vectors: *tt = ts + tc*. Now assume we calculate mutual phase angles of three line phases e.g. by PSFE algorithm. The phases are connected to the channels in order: [L1, L2, L3]. So the TWM will repeat the PSFE algorithm for each of the channels (phases) but for L1 it will pass in ‘time\_stamp = tt(1)’, for L2 ‘time\_stamp = tt(2)’ and for L3 ‘time\_stamp = tt(3)’. Therefore the three results should be synchronized (time shifts corrected). The same applies for dual input algorithms or differential algorithms, except apart from ‘**time\_stamp**’ the TWM also calculates the ‘**time\_shift**‘ and ‘**\*time\_shift\_lo**’ values from the vector ‘tt’. Therefore each algorithm that calculates phase or some time event should perform correction to the ‘**time\_stamp**’ (optionally). For phase the correction is defined by: *phi\_corr = -2\*pi\*f\_of\_component\*time\_shift*.

Note if any of the default corrections is not available (not loaded to the TWM system), it will be still passed into the algorithm but with nominal value, such as 1.0 for gains, 0.0 for phase, etc. However for convenience of the user who may want to call the algorithm manually it is better to make the algorithm in such a way it does not require any correction data at the input and it will therefore use nominal values.

Note that independent variables (**amplitude** and **frequency**) of the 1D or 2D dependencies in the input quantities table may differ for **each channel** and even for **gain** and **phase** of the same correction! The ranges and steps of the independent variables depend on the user correction data files. Each algorithm must check the range of each of the correction individually and somehow respond if the correction range does not cover the required range (throw and error, warning, etc.).

Note the 1D and 2D corrections which are dependent on the **frequency** or **amplitude** quantity may have one or both of the dependencies undefined! I.e. the corresponding dimension of the correction data **‘v’** and uncertainty **‘u’** matrices will have size of 1. In such case the algorithm shall assume the correction is not dependent on that quantity and apply the correction and its uncertainty in the whole range of **frequency**, **amplitude** or both. E.g.:

adc\_sfdr.v = [93]; means algorithm shall assume 93 dBc SFDR for all frequencies and amplitudes. adc\_sfdr.v = [93; 90]; adc\_sfdr\_f.v = [1e3; 1e4]; means to assume 93 dBc for frequency 1e3 Hz and 90 dBc for frequency 1e4.

Note the SFDR data are not meant as corrections. These are only for estimation of the uncertainty.

Note the **‘lsb’** parameter may not be present depending on the selected digitizer. If it is not available, the algorithm should use combination of the **‘adc\_bits’** and **‘adc\_nrng’** for estimation of the **‘lsb’**.

## Input quantities preparation/conversion

The inherent feature of the QWTB toolbox is it automatically converts vectors to horizontal (row vectors). Under normal conditions it is useful function because algorithm will receive the vector data always in the same orientation. However, in case of the 2D correction data it will cause a trouble as the correction data may have one dimension undefined (unity size). Therefore, the data become 1D vector and it may be incorrectly oriented. In order to fix it, a function ‘**qwtb\_restore\_twm\_input\_dims()**’ was made (available in ‘**TWM\octprog\utils**’). The function shall be called as a first thing in the algorithm’s wrapper ‘**alg\_wrapper.m**’. It will restore original orientations of all predefined correction to the ones defined in the list above. It can be also called to fix orientation of individual corrections, see its help. After the orientation fix it will check availability of the default quantities (see table above) and it will generate default values for each of them. Therefore the algorithm wrapper won’t fail even if none of the correction is passed in. On top of that is will also analyze the input quantities and sets a flags that are useful for further processing of the algorithm.

Another function ‘**qwtb\_restore\_correction\_tables()**’ was prepared. This will take the QWTB quantities and it will create TWM style correction tables for the 1D and 2D dependencies. Its use it not necessary, but it makes the algorithm must simpler to use as some comfortable functions for interpolation can be used.

Example of the alg. startup:

**function** dataout = alg\_wrapper(datain, calcset)

% Part of QWTB. Wrapper script for algorithm TWM-FPNLSF.

% Restore orientations of the input vectors to originals (before passing via QWTB)

% This is critical for the correction data!

**[datain,cfg] = qwtb\_restore\_twm\_input\_dims(datain,1);**

**if** cfg.y\_is\_diff

% Input data 'y' is differential: if it is not allowed, put error message here

%error('Differential input data ''y'' not allowed!');

**end**

**if** cfg.is\_multi

% Input data 'y' contains more than one record: if it is not allowed, put error message here

error('Multiple input records in ''y'' not allowed!');

**end**

% Rebuild TWM style correction tables:

% This is not necessary but the TWM style tables are

% more comfortable to use then raw correction matrices

**tab = qwtb\_restore\_correction\_tables(datain,cfg);**

% --------------------------------------------------------------------

% Start of the algorithm

% --------------------------------------------------------------------

% Processing stuff

…

% --- returning results ---

dataout.f.v =

dataout.f.u =

…

% --------------------------------------------------------------------

% End of the algorithm.

% --------------------------------------------------------------------

**end**

## Output quantities

Algorithm may return any quantities: scalars, vectors or matrices. Naming of the output quantities is irrelevant. It will be translated by the QWTB toolbox wrapper function and stored into the file.

If the algorithm calculates frequency **spectrum** in some intermediate step of the calculation, it is preferred to return it as an output quantity together with its frequency scale so it can be displayed in the TWM software.

## Resources

1. TWM tool, url: <https://github.com/smaslan/TWM>
2. INFO-STRINGS, url: <https://github.com/KaeroDot/info-strings>
3. QWTB toolbox, url: <https://qwtb.github.io/qwtb/>
4. GOLPI interface, url: <https://github.com/KaeroDot/GOLPI>
5. A231 Correction Files Reference Manual, url:

<https://github.com/smaslan/TWM/tree/master/doc/A231 Correction Files Reference Manual.docx>

1. A231 Data exchange format and file formats, url: <https://github.com/smaslan/TWM/blob/master/doc/A231%20Data%20exchange%20format%20and%20file%20formats.docx>