QWTB documentation

Implemented algorithms

QWTB version 0.1

`https://qwtb.github.io/qwtb/`

Contents

1

Introduction

This document gives overview of the algorithms implemented in toolbox QWTB.

Toolbox was realized within the EMRP-Project SIB59 Q-Wave. The EMRP is jointly funded by the EMRP par- ticipating countries within EURAMET and the European Union.

2

ADEV – Allan Deviation

Description

.id — ADEV

.name — Allan Deviation

.desc — Compute the Allan deviation for a set of time-domain frequency data.

.citation — D.W. Allan, "The Statistics of Atomic Frequency Standards", Proc. IEEE, Vol. 54, No. 2, pp. 221-230, Feb. 1966. Implementation by M. A. Hopcroft, mhopeng@gmail.com, Matlab Central, online: `http://www.mathworks.com/matlabcentral/fileexchange/13246-allan` Test data by W. J. Riley, "The Calculation of Time Domain Frequency Stability", online: `http://www.wriley.com/paper1ht.htm`

.remarks — If tau is empty array, tau values are automatically generated. Tau values must be divisible by 1/fs. Invalid values are ignored. For tau values really used in the calculation see the output. Using sigma as uncertainty is probably not correct.

.license — BSD License

.requires

- y — sampled values
- fs — sampling frequency
- tau — observation time

.returns

- adev — Allan deviation
- tau — observation time of result values

.providesGUF — yes

.providesMCM — no

Example

Allan Deviation

Example for algorithm ADEV.

ADEV is an algorithm to compute the Allan deviation for a set of time-domain frequency data.

See also W. J. Riley, "The Calculation of Time Domain Frequency Stability". Implementation: M. A. Hopcroft, `mhopeng@gmail.com`, Matlab Central.'

Contents

Generate sample data

```
DI = [];
%!demo
%ysin=2.*sin(2.*pi.*1/300.*[1:1:1e3]);
%! [x y s errors]=adev(1, ysin, 1, 'best averaging time
    is 300 s, i.e. cca one sine period');
% A random numbers with normal probability distribution
    function will be geneated into data input |DI.y.v|.
DI.y.v = 1.5 + 3.*randn(1, 1e3);
% Next a drift is added:
DI.y.v = DI.y.v + [1:1:1e3]./100;
% Lets suppose a sampling frequency is 1 Hz:
DI.fs.v = 1;
% Let the algorithm generate all possible tau values
   automatically:
DI.tau.v = [];
```

Call algorithm

Use QWTB to apply algorithm ADEV to data DI.

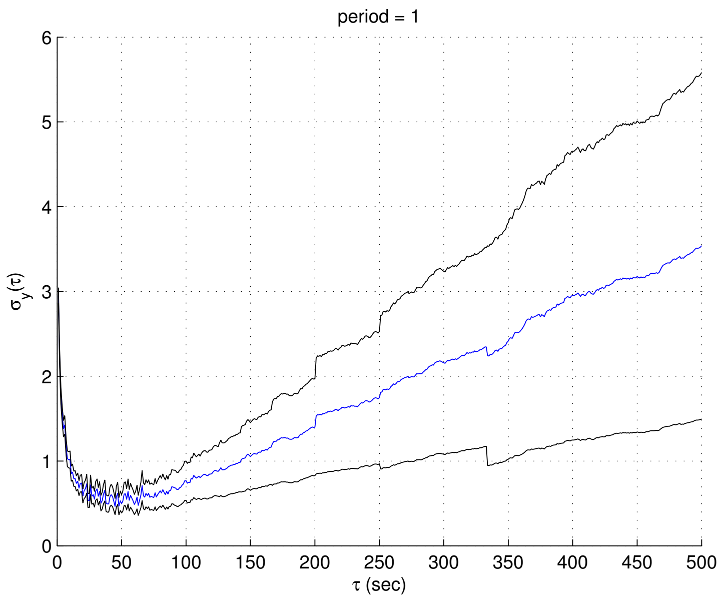
```
DO = qwtb('ADEV', DI);
```

```
QWTB: no uncertainty calculation
```

Display results

Log log figure is the best to see allan deviation results:

```
figure; hold on
loglog(DO.tau.v, DO.adev.v, '-b')
loglog(DO.tau.v, DO.adev.v + DO.adev.u, '-k')
loglog(DO.tau.v, DO.adev.v - DO.adev.u, '-k')
xlabel('\tau (sec)');
ylabel('\sigma_y(\tau)');
title(['period = ' num2str(DI.fs.v)]);
grid('on'); hold off
```



3

FPSWF – Four Parameter Sine Wave Fitting

Description

.id — FPSWF

.name — Four Parameter Sine Wave Fit

.desc — Fits a sine wave to the recorded data by means of least squares fitting using 4 parameter (frequency, amplitude, phase and offset) model. An estimate of signal frequency is required. Due to non-linear characteristic, converge is not always achieved. When run in Matlab, function 'lsqnonlin' in Optimization toolbox is used. When run in GNU Octave, function 'leasqr' in GNU Octave Forge package optim is used.

.citation —

.remarks — Algorithm works essentially only for simple sine wave. Algorithm is very sensitive to distortion. Algorithm requires good estimate of signal frequency.

.license — MIT License

.requires

t — Time series of sampled data

y — Sampled values

fest — Estimate of signal frequency

.returns

f — Frequency of main signal component

A — Amplitude of main signal component
ph — Phase of main signal component
O — Offset of signal

.providesGUF — no

.providesMCM — no

Example

Four parameter sine wave fitting

Example for algorithm FPSWF.

FPSWF is an algorithm for estimating the frequency, amplitude, and phase of the sine waveform. The algorithm use least squares method. Algorithm requires good estimate of frequency.

Contents

- Generate sample data
- Call algorithm
- Display results

Generate sample data

Two quantities are prepared: t and y, representing 1 second of sinus waveform of nominal frequency 1 kHz, nominal amplitude 1 V, nominal phase 1 rad and offset 1 V sampled at sampling frequency 10 kHz.

```
DI = [];
Anom = 2; fnom = 100; phnom = 1; Onom = 0.2;
DI.t.v = [0:1/1e4:1-1/1e4];
DI.y.v = Anom*sin(2*pi*fnom*DI.t.v + phnom) + Onom;
```

Lets make an estimate of frequency 0.2 percent higher than nominal value:

```
DI.fest.v = 100.2;
```

Call algorithm

Use QWTB to apply algorithm FPSWF to data DI.

```
CS.verbose = 1;
DO = qwtb('FPSWF', DI, CS);
```

```
QWTB: no uncertainty calculation
Fitting started

Local minimum found.

Optimization completed because the size of the
   gradient is less than
the default value of the function tolerance.



Fitting finished
```

Display results

Results is the amplitude, frequency and phase of sampled waveform.

```
A = DO.A.v
f = DO.f.v
ph = DO.ph.v
O = DO.O.v
```

```

A =

     2.0000


f =
    100


ph =

     1.0000


O =

     0.2000
```

Errors of estimation in parts per milion:

```
Aerrppm = (DO.A.v - Anom)/Anom .* 1e6
ferrppm = (DO.f.v - fnom)/fnom .* 1e6
pherrppm = (DO.ph.v - phnom)/phnom .* 1e6
Oerrppm = (DO.O.v - Onom)/Onom .* 1e6
```

```

Aerrppm =

    4.8894e-07


ferrppm =

       0


pherrppm =

    4.8850e-09


Oerrppm =

   -1.1102e-08
```

4

INL-DNL – Integral and Differential Non-Linearity of ADC

Description

.id — INL-DNL

.name — Integral and Differential Non-Linearity of ADC

.desc — Calculates Integral and Differential Non-Linearity of an ADC. The histogram of measured data is used to calculate INL and DNL estimators. ADC has to sample a pure sine wave. To estimate all transition levels the amplitude of the sine wave should overdrive the full range of the ADC by at least 120

.citation — Estimators are based on Tamás Virosztek, MATLAB-based ADC testing with sinusoidal excitation signal (in Hungar- ian), B.Sc. Thesis, 2011. Implementation: Virosztek, T., Pálfi V., Renczes B., Kollár I., Balogh L., Sárhegyi A., Márkus J., Bilau Z. T., ADCTest project site: `http://www.mit.bme.hu/projects/adctest` 2000-2014

.remarks — Based on the ADCTest Toolbox v4.3, November 25, 2014.

.license — UNKNOWN

.requires

- bitres — Bit resolution of the ADC
- codes — Sampled values represented as ADC codes (not converted to voltage)

.returns

- DNL — Differential Non-Linearity

.providesGUF — no

.providesMCM — no

Example

Integral and Differential Non Linearity of ADC

Example for algorithm INL-DNL

INL-DNL is an algorithm for estimating Integral and Differential Non-Linearity of an ADC. ADC has to sample a pure sine wave. To estimate all transition levels the amplitude of the sine wave should overdrive the full range of the ADC by at least 120%. If not so, non estimated transition levels will be assumed to be 0 and the results may be less accurate. As an input ADC codes are required.';

See also 'Virosztek, T., Pálfi V., Renczes B., Kollár I., Balogh L., Sárhegyi A., Márkus J., Bilau Z. T., ADCTest project site: `http://www.mit.bme.hu/projects/adctest` 2000-2014';

Contents

- Generate sample data
- Call algorithm

Generate sample data

Suppose a sine wave of nominal frequency 10 Hz and nominal amplitude 1.5 V is sampled by ADC with bit resolution of 4 and full range of 1 V. First quantity bitres with number of bits of resolution of the ADC is prepared and put into input data structure DI.

```
DI = [];
DI.bitres.v = 4;
```

Waveform is constructed. Amplitude is selected to overload the ADC.

```
t=[0:1/1e4:1-1/1e4];
Anom = 3.5; fnom = 2; phnom = 0;
wvfrm = Anom*sin(2*pi*fnom*t + phnom);
```

Next ADC code values are calculated. It is simulated by quantization and scaling of the sampled waveform. In real measurement code values can be obtained directly from the ADC. Suppose ADC range is -2..2.

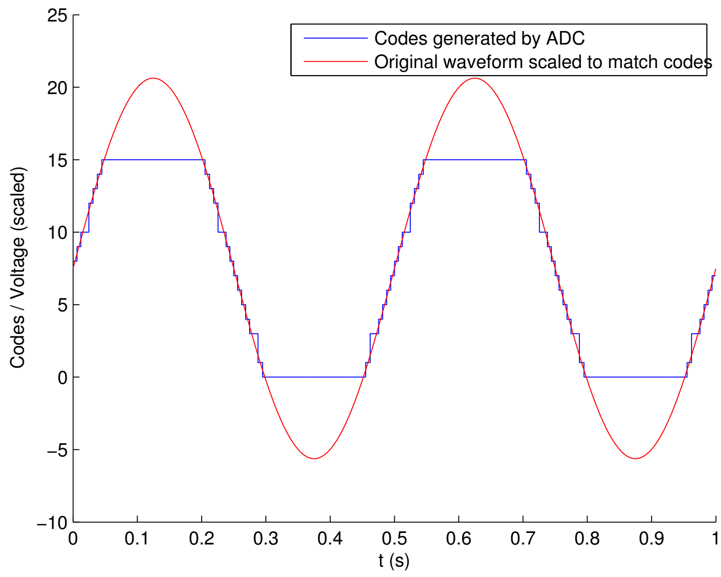
```
codes = wvfrm;
rmin = -2; rmax = 2;
levels = 2.^DI.bitres.v - 1;
codes(codes<rmin) = rmin;
codes(codes>rmax) = rmax;
codes = round((codes-rmin)./(rmax-rmin).*levels);
```

Now lets introduce ADC error. Instead of generating code 2 ADC erroneously generates code 3 and instead of 11 it generates 10.

```
codes(codes==2) = 3;
codes(codes==11) = 10;
codes = codes + min(codes);
```

Create quantity codes and plot a figure with sampled sine wave and codes.

```
DI.codes.v = codes;
figure
hold on
stairs(t, codes);
wvfrm = (wvfrm - rmin)./(rmax-rmin).*levels;
plot(t, wvfrm, '-r');
xlabel('t (s)')
ylabel('Codes / Voltage (scaled)');
legend('Codes generated by ADC','Original waveform
   scaled to match codes');
hold off
```



Call algorithm

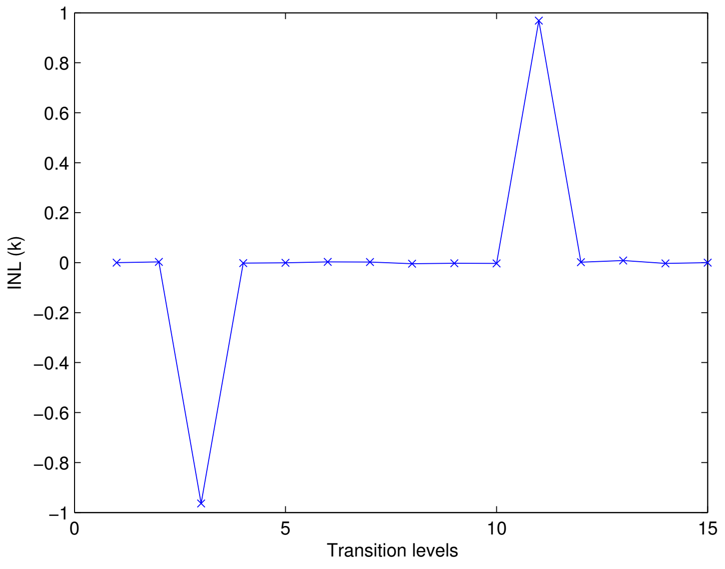
Apply INL algorithm to the input data DI.

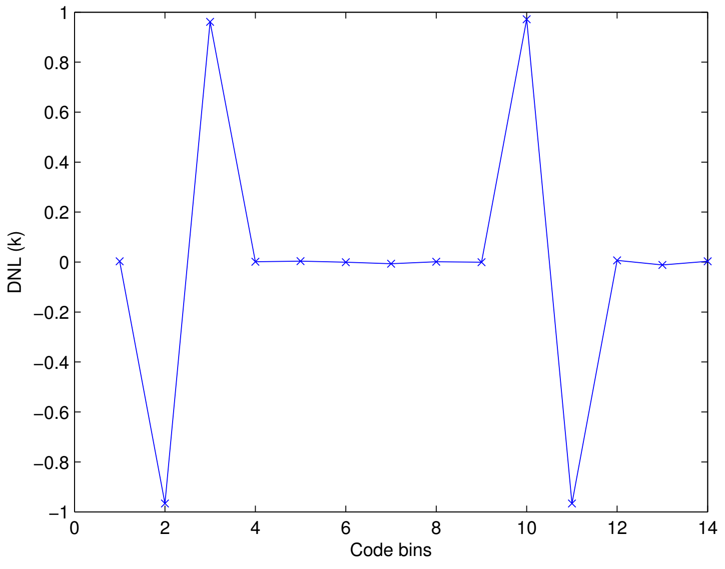
```
DO = qwtb('INL-DNL', DI);
```

```
QWTB: no uncertainty calculation
```

Plot results of integral non-linearity. One can clearly observe defects on codes 3 and 11.

```
figure
plot(DO.INL.v, '-x');
xlabel('Transition levels')
ylabel('INL (k)')
% Plot results of differential non-linearity. One can
   clearly observe defects on transitions 2-3 and
% 10-11.
figure
plot(DO.DNL.v, '-x');
xlabel('Code bins')
ylabel('DNL (k)')
```


INL (k)
Transition levels
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
5
10
15


DNL (k)
Code bins
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
2
4
6
8
10
12
14

5

SFDR – Spurious Free Dynamic Range

Description

.id — SFDR

.name — Spurious Free Dynamic Range

.desc — Calculates Spurious Free Dynamic Range of an ADC. A FFT method with Blackman windowing is used to calculate a spectrum and SFDR is estimated in decibels relative to carrier amplitude. ADC has to sample a pure sine wave. SFDR is calculated according IEEE Std 1057-2007

.citation — Implementation: Virosztek, T., Pálfi V., Renczes B., Kollár I., Balogh L., Sárhegyi A., Márkus J., Bilau Z. T., ADCTest project site: `http://www.mit.bme.hu/projects/adctest` 2000-2014

.remarks — Based on the ADCTest Toolbox v4.3, November 25, 2014.

.license — UNKNOWN

.requires

y — Sampled values

.returns

SFDRdBc — Spurious Free Dynamic Range in decibels relative to carrier (dBc)

.providesGUF — no

.providesMCM — no

Example

Spurious Free Dynamic Range by means of Fast fourier transform

Example for algorithm SFDR.

Calculates SFDR by calculating FFT spectrum.

Contents

Generate sample data

First quantity y representing 1 second of signal containing spurious component is prepared. Main signal component has nominal frequency 1 kHz, nominal amplitude 2 V, nominal phase 1 rad and offset 1 V sampled at sampling frequency 10 kHz.

```
DI = [];
fsnom = 1e4; Anom = 4; fnom = 100; phnom = 1; Onom =
    0.2;
t = [0:1/fsnom:1-1/fsnom];
DI.y.v = Anom*sin(2*pi*fnom*t + phnom);
```

A spurious component with amplitude at 1/100 of main carrier frequency is added. Thus by definition the SFDR in dBc has to be 40.

```
DI.y.v = DI.y.v + Anom./100*sin(2*pi*fnom*3.5*t + phnom
    );
```

Call algorithm

Use QWTB to apply algorithm SFDR to data DI.

```
DO = qwtb('SFDR', DI);
```

```
QWTB: no uncertainty calculation
```

Display results

Result is the SFDR (dBc).

```
SFDR = DO.SFDRdBc.v
```

```

SFDR =

    40.0000
```

6

MADEV – Modified Allan Deviation

Description

.id — MADEV

.name — Modified Allan Deviation

.desc — Compute the modified Allan deviation for a set of time-domain frequency data.

.citation — D.W. Allan and J.A. Barnes, "A Modified Allan Variance with Increased Oscillator Characterization Ability", Proc. 35th Annu. Symp. on Freq. Contrl., pp. 470-474, May 1981. Implementation: Implementation by M. A. Hopcroft, mhopeng@gmail.com, Matlab Central, online: `http://www.mathworks.com/matlabcentral/fileexchange/26637-allan-modified` Test data by W. J. Riley, "The Calculation of Time Domain Frequency Stability", online: `http://www.wriley.com/paper1ht.htm`

.remarks — If tau is empty array, tau values are automatically generated. Tau values must be divisible by 1/fs. Invalid values are ignored. For tau values really used in the calculation see the output. Using sigma as uncertainty is probably not correct.

.license — BSD License

.requires

y — sampled values

fs — sampling frequency

tau — observation time

.returns

madev — modified Allan deviation

tau — observation time of result values

.providesGUF — yes

.providesMCM — no

Example

Modified Allan Deviation

Example for algorithm MADEV.

MADEV is an algorithm to compute the modified Allan deviation for a set of time-domain frequency data.

See also W. J. Riley, "The Calculation of Time Domain Frequency Stability". Implementation: M. A. Hopcroft, `mhopeng@gmail.com`, Matlab Central.'

Contents

- Generate sample data
- Call algorithm
- Display results

Generate sample data

```
DI = [];
%!demo
%ysin=2.*sin(2.*pi.*1/300.*[1:1:1e3]);
%! [x y s errors]=madev(1, ysin, 1, 'best averaging
   time is 300 s, i.e. cca one sine period');
% A random numbers with normal probability distribution
    function will be generated into data input |DI.y.v
   |.
DI.y.v = 1.5 + 3.*randn(1, 1e3);
% Next a drift is added:
DI.y.v = DI.y.v + [1:1:1e3]./100;
% Lets suppose a sampling frequency is 1 Hz:
DI.fs.v = 1;
```

```
% Let the algorithm generate all possible tau values
    automatically:
DI.tau.v = [];
```

Call algorithm

Use QWTB to apply algorithm MADEV to data DI.

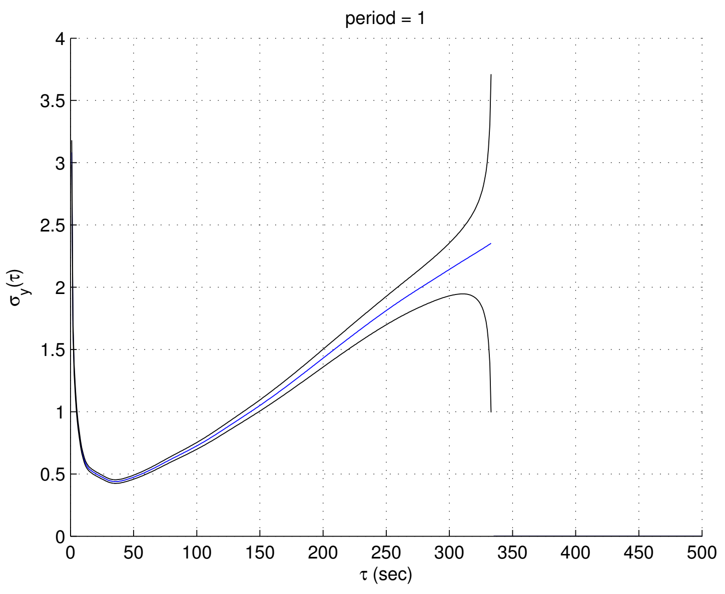
```
DO = qwtb('MADEV', DI);
```

```
QWTB: no uncertainty calculation
```

Display results

Log log figure is the best to see modified allan deviation results:

```
figure; hold on
loglog(DO.tau.v, DO.madev.v, '-b')
loglog(DO.tau.v, DO.madev.v + DO.madev.u, '-k')
loglog(DO.tau.v, DO.madev.v - DO.madev.u, '-k')
xlabel('\tau (sec)');
ylabel('\sigma_y(\tau)');
title(['period = ' num2str(DI.fs.v)]);
grid('on'); hold off
```


period = 1
$\sigma_y(\tau)$
τ (sec)
0
0.5
1
1.5
2
2.5
3
3.5
4
0
50
100
150
200
250
300
350
400
450
500

7

OADEV – Overlapping Allan Deviation

Description

.id — OADEV

.name — Overlapping Allan Deviation

.desc — Compute the overlapping Allan deviation for a set of time-domain frequency data.

.citation — D.A. Howe, D.W. Allan and J.A. Barnes, "Properties of Signal Sources and Measurement Methods', Proc. 35th Annu. Symp. on Freq. Contrl., pp. 1-47, May 1981. Implementation: Implementation by M. A. Hopcroft, mhopeng@gmail.com, Matlab Central, online: `http://www.mathworks.com/matlabcentral/fileexchange/26441-allan-overlap` Test data by W. J. Riley, "The Calculation of Time Domain Frequency Stability", online: `http://www.wriley.com/paper1ht.htm`

.remarks — If tau is empty array, tau values are automatically generated. Tau values must be divisible by 1/fs. Invalid values are ignored. For tau values really used in the calculation see the output. Using sigma as uncertainty is probably not correct.

.license — BSD License

.requires

y — sampled values

fs — sampling frequency

tau — observation time

.returns

oadev — overlapping Allan deviation
tau — observation time of result values

.providesGUF — yes

.providesMCM — no

Example

Allan Overlapping Deviation

Example for algorithm OADEV.

OADEV is an algorithm to compute the overlapping Allan deviation for a set of time-domain frequency data.

See also W. J. Riley, "The Calculation of Time Domain Frequency Stability". Implementation: M. A. Hopcroft, mhopeng@gmail.com, Matlab Central.'

Contents

Generate sample data

```
DI = [];
%!demo
%ysin=2.*sin(2.*pi.*1/300.*[1:1:1e3]);
%! [x y s errors]=adev(1, ysin, 1, 'best averaging time
    is 300 s, i.e. cca one sine period');
% A random numbers with normal probability distribution
    function will be generated into data input |DI.y.v
   |.
DI.y.v = 1.5 + 3.*randn(1, 1e3);
% Next a drift is added:
DI.y.v = DI.y.v + [1:1:1e3]./100;
% Lets suppose a sampling frequency is 1 Hz:
```

```
DI.fs.v = 1;
% Let the algorithm generate all possible tau values
    automatically:
DI.tau.v = [];
```

Call algorithm

Use QWTB to apply algorithm OADEV to data DI.

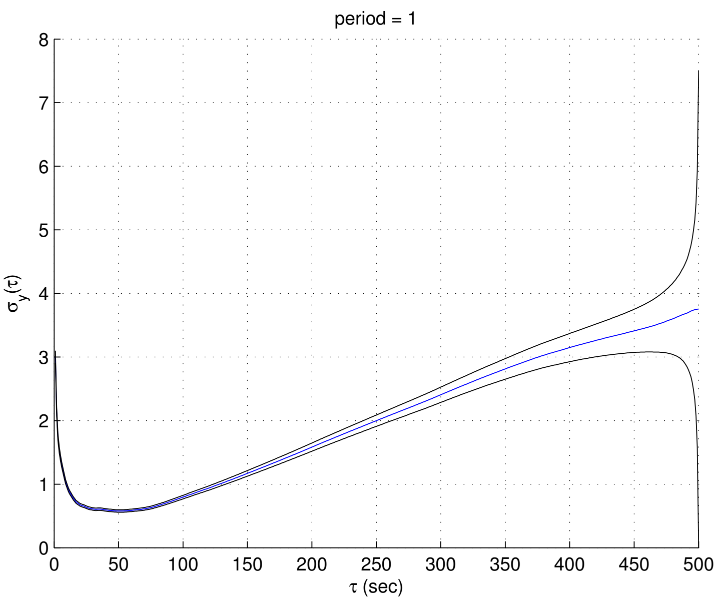
```
DO = qwtb('OADEV', DI);
```

```
QWTB: no uncertainty calculation
```

Display results

Log log figure is the best to see allan deviation results:

```
figure; hold on
loglog(DO.tau.v, DO.oadev.v, '-b')
loglog(DO.tau.v, DO.oadev.v + DO.oadev.u, '-k')
loglog(DO.tau.v, DO.oadev.v - DO.oadev.u, '-k')
xlabel('\tau (sec)');
ylabel('\sigma_y(\tau)');
title(['period = ' num2str(DI.fs.v)]);
grid('on'); hold off
```


period = 1
$\sigma_y(\tau)$
τ (sec)
0
1
2
3
4
5
6
7
8
0
50
100
150
200
250
300
350
400
450
500

8

PSFE – Phase Sensitive Frequency Estimator

Description

.id — PSFE

.name — Phase Sensitive Frequency Estimator

.desc — An algorithm for estimating the frequency, amplitude, and phase of the fundamental component in harmonically distorted waveforms. The algorithm minimizes the phase difference between the sine model and the sampled waveform by effectively minimizing the influence of the harmonic components. It uses a three-parameter sine-fitting algorithm for all phase calculations. The resulting estimates show up to two orders of magnitude smaller sensitivity to harmonic distortions than the results of the four-parameter sine fitting algorithm.

.citation — Lapuh, R., "Estimating the Fundamental Component of Harmonically Distorted Signals From Noncoherently Sampled Data," Instrumentation and Measurement, IEEE Transactions on , vol.64, no.6, pp.1419,1424, June 2015, doi: 10.1109/TIM.2015.2401211, URL: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7061456&isnumber=7104190`

.remarks — Very small errors, effective for harmonically distorted signals.

.license — MIT License

.requires

t — time series of sampled data

y — sampled values

.returns

f — Frequency of main signal component
A — Amplitude of main signal component
ph — Phase of main signal component

.providesGUF — no

.providesMCM — no

Example

Phase Sensitive Frequency Estimator

Example for algorithm PSFE.

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

See also Lapuh, R., "Estimating the Fundamental Component of Harmonically Distorted Signals From Noncoherently Sampled Data," Instrumentation and Measurement, IEEE Transactions on , vol.64, no.6, pp.1419,1424, June 2015, doi: 10.1109/TIM.2015.2401211, URL: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7061456&isnumber=7104190’`

Contents

Generate sample data

Two quantities are prepared: `t` and `y`, representing 1 second of sinus waveform of nominal frequency 100 Hz, nominal amplitude 1 V and nominal phase 1 rad, sampled at sampling frequency 10 kHz.

```
DI = [];
Anom = 1; fnom = 100; phnom = 1;
DI.t.v = [0:1/1e4:1-1/1e4];
DI.y.v = Anom*sin(2*pi*fnom*DI.t.v + phnom);
```

Add noise:

```
DI.y.v = DI.y.v + 1e-3.*randn(size(DI.y.v));
```

Call algorithm

Use QWTB to apply algorithm PSFE to data DI.

```
DO = qwtb('PSFE', DI);
```

```
QWTB: no uncertainty calculation
```

Display results

Results is the amplitude, frequency and phase of sampled waveform.

```
f = DO.f.v
A = DO.A.v
ph = DO.ph.v
```

```
f =

  100.0000


A =

    1.0000
```

```
ph =

    1.0000
```

Errors of estimation in parts per milion:

```
ferrppm = (DO.f.v - fnom)/fnom .* 1e6
Aerrppm = (DO.A.v - Anom)/Anom .* 1e6
pherrppm = (DO.ph.v - phnom)/phnom .* 1e6
```

```
ferrppm =

   -0.0015


Aerrppm =

    1.2432


pherrppm =

   21.2068
```

9

SINAD-ENOB – Ratio of signal to noise and distortion and Effective number of bits (in time space)

Description

.id — SINAD-ENOB

.name — Ratio of signal to noise and distortion and Effective number of bits (in time space)

.desc — Algorithm calculates Ratio of signal to noise and distortion and Effective number of bits in time space, therefore it is suitable for noncoherent measurements. Requires estimates of the main signal component parameters: frequency, amplitude, phase and offset. If these values are estimated by four parameter sine wave fit, the SINAD and ENOB will be calculated according IEEE Std 1241-2000. A large sine wave should be applied to the ADC input. Almost any error source in the sine wave input other than gain accuracy and dc offset can affect the test result.

.citation — IEEE Std 1241-2000, pages 52 - 54

.remarks —

.license — MIT License

.requires

t — time vector

y — sampled values

f — frequency of the main signal component

A — amplitude of the main signal component

ph — phase of the main signal component

O — offset of the main signal component

bitres — bit resolution of an ADC

range — Full scale range of an ADC

.returns

SINADdB — Ratio of signal to noise and distortion in decibels relative to the amplitude of the main signal component

ENOB — Effective number of bits

.providesGUF — no

.providesMCM — no

Example

Ratio of signal to noise and distortion and Effective number of bits (time space)

Example for algorithm SINAD-ENOB.

Algorithm calculates Ratio of signal to noise and distortion and Effective number of bits in time space. First signal is generated, then estimates of signal parameters are calculated by Four parameter sine wave fitting and next SINAD and ENOB are calculated. This example do not take into account a quantisation noise.

Contents

Generate sample data

Two quantities are prepared: t and y, representing 1 second of sinus waveform of nominal frequency 1 kHz, nominal amplitude 1 V, nominal phase 1 rad and offset 1 V sampled at sampling frequency 10 kHz.

```
DI = [];
Anom = 2; fnom = 100; phnom = 1; Onom = 0.2;
DI.t.v = [0:1/1e4:1-1/1e4];
DI.y.v = Anom*sin(2*pi*fnom*DI.t.v + phnom) + Onom;
```

Add a noise with normal distribution probability:

```
noisestd = 1e-4;
DI.y.v = DI.y.v + noisestd.*randn(size(DI.y.v));
```

Lets make an estimate of frequency 0.2 percent higher than nominal value:

```
DI.fest.v = 100.2;
```

Calculate estimates of signal parameters

Use QWTB to apply algorithm FPSWF to data DI.

```
CS.verbose = 1;
DO = qwtb('FPSWF', DI, CS);
```

```
QWTB: no uncertainty calculation
Fitting started

Local minimum found.

Optimization completed because the size of the
    gradient is less than
the default value of the function tolerance.



Fitting finished
```

Copy results to inputs

Take results of FPSWF and put them as inputs DI.

```
DI.f = DO.f;
DI.A = DO.A;
DI.ph = DO.ph;
DI.O = DO.O;
```

Suppose the signal was sampled by a 20 bit digitizer with full scale range of 6 V (+- 3V). (The signal is not quantised, so the quantization noise is not present. Thus the simulation and results are not fully correct.):

```
DI.bitres.v = 20;
DI.range.v = 3;
```

Calculate SINAD and ENOB

```
DO = qwtb('SINAD-ENOB', DI, CS);
```

```
QWTB: no uncertainty calculation
```

Display results:

Results are:

```
SINADdB = DO.SINADdB.v
ENOB = DO.ENOB.v
```

```
SINADdB =

    83.0029

```

```
ENOB =

    13.0790
```

Theoretical value of SINADdB is 20*log10(Anom./(noisestd.*sqrt(2))). Theoretical value of ENOB is log2(DI.range.v./(noisestd.*sqrt(12))). Absolute error of results are:

```
SINADdBtheor = 20*log10(Anom./(noisestd.*sqrt(2)));
ENOBtheor = log2(DI.range.v./(noisestd.*sqrt(12)));
SINADerror = SINADdB - SINADdBtheor
ENOBerror = ENOB - ENOBtheor
```

```

SINADerror =

    -0.0074


ENOBerror =

    -0.0012
```

10

SP-FFT – Spectrum by means of Fast Fourier Transform

Description

.id — SP-FFT

.name — Spectrum by means of Fast Fourier Transform

.desc — Calculates frequency and phase spectrum by means of Fast Fourier Transform algorithm. Result is normalized.

.citation —

.remarks —

.license — MIT License

.requires

- y — Sampled values
- fs — Sampling frequency

.returns

- f — Frequency series
- A — Amplitude series
- ph — Phase series

.providesGUF — no

.providesMCM — no

Example

Signal Spectrum by means of Fast fourier transform

Example for algorithm SP-FFT.

Calculates frequency and phase spectrum by means of Fast Fourier Transform algorithm. Result is normalized.

Contents

Generate sample data

Two quantities are prepared: `y` and `fs`, representing 1 second of signal containing 5 harmonic components and one inter-harmonic component. Main signal component has nominal frequency 1 kHz, nominal amplitude 2 V, nominal phase 1 rad and offset 1 V sampled at sampling frequency 10 kHz.

```
DI = [];
fsnom = 1e4; Anom = 2; fnom = 100; phnom = 1; Onom =
    0.2;
t = [0:1/fsnom:1-1/fsnom];
DI.y.v = Anom*sin(2*pi*fnom*t + phnom);
for i = 2:45
        DI.y.v = DI.y.v + Anom./i*sin(2*pi*fnom*i*t +
    phnom + i - 1);
end
DI.y.v = DI.y.v + 1*sin(2*pi*fnom*1.456*t + phnom);
DI.fs.v = fsnom;
```

Call algorithm

Use QWTB to apply algorithm `SP-FFT` to data `DI`.

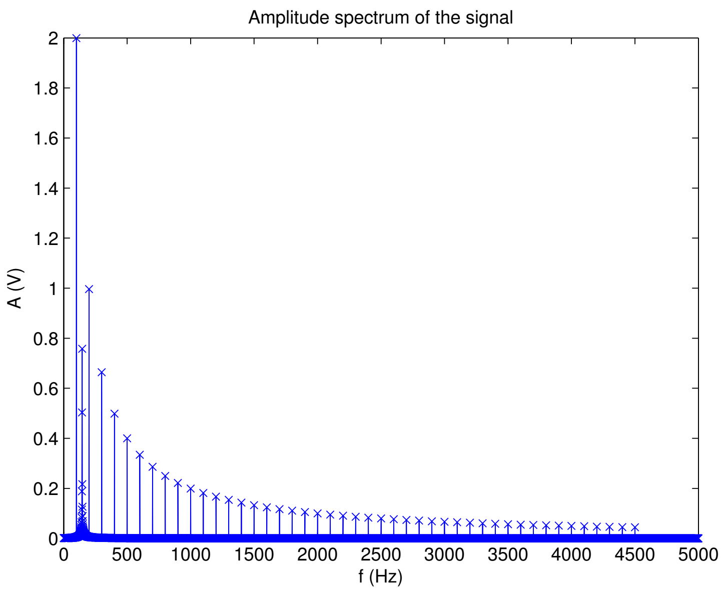
```
DO = qwtb('SP-FFT', DI);
```

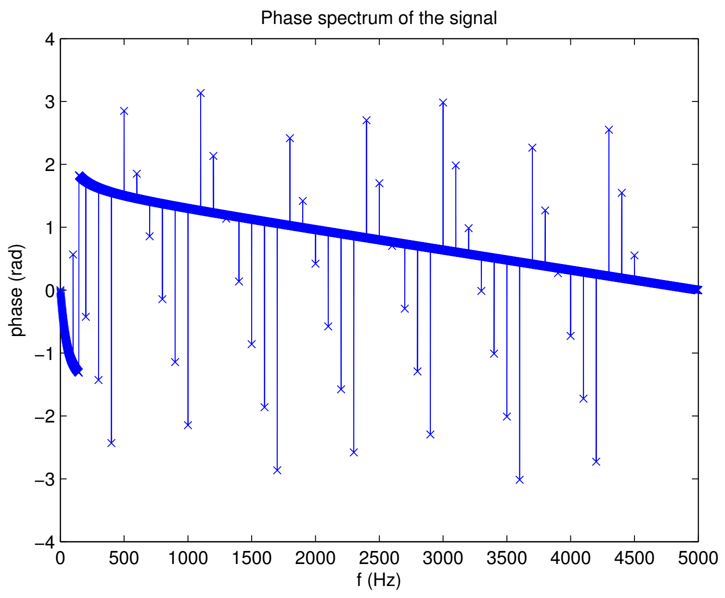
```
QWTB: no uncertainty calculation
```

Display results

Results is the amplitude and phase spectrum.

```
figure
plot(DO.f.v, DO.A.v, '-x')
xlabel('f (Hz)'); ylabel('A (V)'); title('Amplitude
   spectrum of the signal');
figure
plot(DO.f.v, DO.ph.v, '-x')
xlabel('f (Hz)'); ylabel('phase (rad)'); title('Phase
   spectrum of the signal');
```




Phase spectrum of the signal
phase (rad)
f (Hz)
4
3
2
1
0
−1
−2
−3
−4
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000