QWTB documentation

Implemented algorithms

2024-06-26

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

Contents

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.

ADEV – Allan Deviation

Description

Id: ADEV

Name: Allan Deviation

Description: 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 sampling frequency |fs| is not supplied, wrapper will calculate |fs| from sampling time |Ts| or if not supplied, mean of differences of time series |t| is used to calculate |Ts|. If observation time(s) |tau| is not supplied, 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.

License: BSD License

Provides GUF: yes

Provides MCM: no

Input Quantities

Required: fs or Ts or t, `y`

Optional: `tau`

Descriptions:

Ts – Sampling time
fs – Sampling frequency
t – Time series
tau – Observation time
y – Sampled values

Output Quantities:

adev – Allan deviation

tau – Observation time of resulted values

Example

Contents

Generate sample data

A random numbers with normal probability distribution function will be generated into input data `DI.y.v`. Next a drift will be added.

```
DI = [];
DI.y.v = 1.5 + 3.*randn(1, 1e3);
DI.y.v = DI.y.v + [1:1:1e3]./100;
```

Lets suppose a sampling frequency is 1 Hz. The algorithm will generate all possible tau values automatically.

```
DI.fs.v = 1;
```

Call algorithm

Use QWTB to apply algorithm `ADEV` to data `DI`.

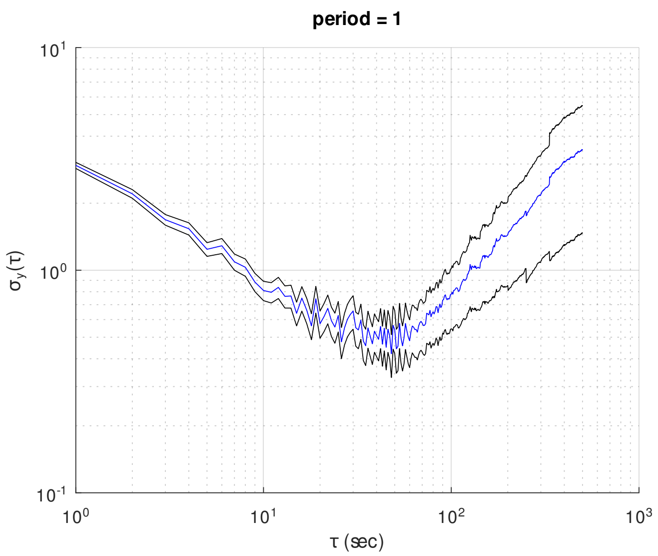
```
DO = qwtb('ADEV', DI);
```

```
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(1/DI.fs.v)]);
grid('on'); hold off
```



CCC – Calibration Curves Computing

Description

Id: CCC

Name: Calibration Curves Computing

Description: Calibration Curves Computing is a software for the evaluation of instrument calibration curves and developed at Istituto Nazionale di Ricerca Metrologica (INRIM). The software may be applied to pairs of measurement values of independent/explanatory variable and dependent/explained variable. If uncertainties associated with the data are available, they can be provided as inputs to the software. The regression models which can be addressed by the software are (fractional) polynomial curves. The software can perform the following kind of regression procedures: Ordinary least-squares regression (OLS), Weighted least-squares regression (WLS), Weighted total least-squares regression (WTLS).

Citation: A. Malengo and F. Pennecchi, A weighted total least-squares algorithm for any fitting model with correlated variables, Metrologia (2013), 50, 654.

Remarks: If |x| and |y| are matrices: wrapper suppose the measured data are a set of groups organized in such that every row of |x| and |y| is one set of measurement, uncertainties in |x| and |y| are neglected and Model 2b is selected. If you want Model 1b or Model 3b, it must be set in quantitiy |model|. If |x| and |y| are vectors: if uncertainties of |x| are zero and uncertainties of |y| contains all the same numbers, Model 1b is selected; if uncertainties of |x| are zero and uncertainties of |y| contains various numbers, Model 2b is selected; if uncertainties of |x| are nonzero Model 3b is selected. In every case the value of |model| overloads automatic determination of the model.

License: Dedicated license

Provides GUF: yes

Provides MCM: no

Input Quantities

Required: x, y, exponents

Optional: model

Parameters: exponents, model

Descriptions:

exponents – Exponents of polynomial used to fit, from -5 to 5 including 0, -0.5, 0.5

model – Identification of the model. 1a, 2a, 3a, 1b, 2b, 3b.

x – Independent/explanatory variable

y – Dependent/explained variable

Output Quantities:

coefs – Fitted coefficients

exponents – Exponents of polynomial used to fit, from -5 to 5 including 0, -0.5, 0.5

func – Inline function constructed for exponents with parameters ‘x‘ and ‘coefs.v‘

model – Model used for calculation.

yhat – Fitted values y

Example

Contents

Generate sample data

An dependence of amplitude error (Volts, ppm) on signal frequency (Hz) of an ADC was measured and uncertainties of measurement was estimated. The uncertainty of frequency can be considered as negligible.

```
%3.7+12.*x+3.*x.^2
f = [10 1e2 1e3 1e4 1e5];
err = [19.700 32.700 69.700 90.700 148.700];
err_unc = [4 10 13 20 33];
```

Set independent and dependent variables for CCC algorithm. Lets operate in semi logarithm space for easy plotting.

```
DI = [];
DI.x.v = log10(f);
DI.x.u = [];
DI.y.v = err;
DI.y.u = err_unc;
```

Suppose the ADC has quadratic dependence of the error on the signal frequency.

```
DI.exponents.v = [0 1 2];
```

Call algorithm

Use QWTB to apply algorithm CCC to data DI.

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

```
QWTB: no uncertainty calculation
QWTB: CCC wrapper: model was set by CCC wrapper to a value ‘
   Model 2a‘.
warning: inline is obsolete; use anonymous functions instead
```

Display results

Results is

```
disp(['offset           : ' num2str(DO.coefs.v(1)) ' +- '
   num2str(DO.coefs.u(1))])
disp(['linear coeff.    : ' num2str(DO.coefs.v(2)) ' +- '
   num2str(DO.coefs.u(2))])
disp(['quadratic coeff.: ' num2str(DO.coefs.v(3)) ' +- '
   num2str(DO.coefs.u(3))])
```

```
offset          : 12.6828 +- 16.9884
linear coeff.   : 1.9434 +- 19.1198
quadratic coeff.: 4.9055 +- 3.9754
```

Interpolate values

Interpolate fitted polynom at values `t`.

```
t = [0:0.1:6];
ty = DO.func.v(t, DO.coefs.v);
```

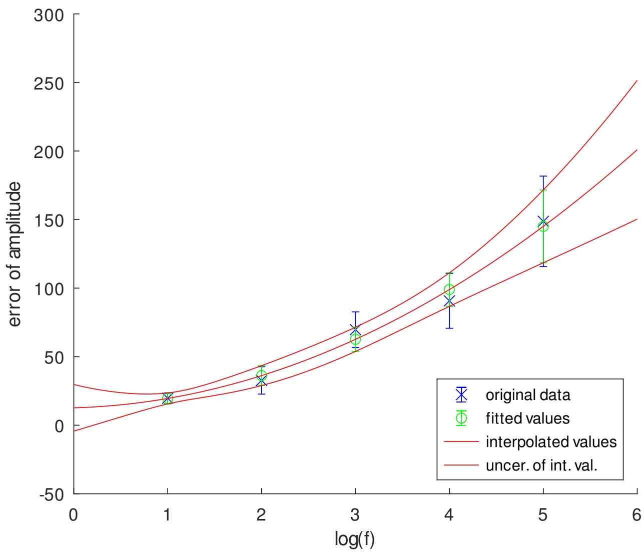
Calculate uncertainties of interpolated values (`S` is sensitivity matrix, `CC` is covariance matrix of coefficients, `CT` is covariance matrix of interpolated values, `uty` is uncertainty of interpolated values).

```
for i = 1:length(t);
        S = t(i).^DI.exponents.v;
        CC = diag(DO.coefs.u,0)*DO.coefs.c*diag(DO.coefs.u,0);
        CT(i)=S*CC*S';
end
uty=CT.^0.5;
```

Plot results

```
hold on
errorbar(DI.x.v, DI.y.v, DI.y.u, 'xb')
errorbar(DI.x.v, DO.yhat.v, DO.yhat.u, 'og')
plot(t, ty, '-r');
plot(t, ty + uty, '-r');
plot(t, ty - uty, '-r');
```

```
xlabel('log(f)')
ylabel('error of amplitude')
legend('original data','fitted values','interpolated values', '
    uncer. of int. val.','location','southeast')
hold off
```



FOAV – Fast Fully Overlapped Allan Variance

Description

Id: FFOAV

Name: Fast Fully Overlapped Allan Variance

Description: Fast, parallelizeable algorithm to calculate Fully Overlapped Allan Variance for generating smooth Allan Deviation plots whose serial running time is $\Theta(N^2)$.

Citation: S. M. Yadav, S. K. Shastri, G. B. Chakravarthi, V. Kumar, D. R. A and V. Agrawal, "A Fast, Parallel Algorithm for Fully Overlapped Allan Variance and Total Variance for Analysis and Modelling of Noise in Inertial Sensors," in IEEE Sensors Letters. doi: 10.1109/LSENS.2018.2829799. Github repository: `https://github.com/shrikanth95/Fast-Parallel-Fully-Overlapped`

Remarks: If sampling frequency |fs| is not supplied, wrapper will calculate |fs| from sampling time |Ts| or if not supplied, mean of differences of time series |t| is used to calculate |Ts|. The output is recaldulated to return deviation to correspond other algorithms.

License: MIT License

Provides GUF: no

Provides MCM: no

Input Quantities

Required: fs or Ts or t, y

Descriptions:

`Ts` – Sampling time
`fs` – Sampling frequency
`t` – Time series
`y` – Sampled values

Output Quantities:

`oadev` – Overlapped Allan deviation
`tau` – Observation time of resulted values

Example

Contents

NOT FINISHED! XXX

Generate sample data

A random numbers with normal probability distribution function will be generated into input data `DI.y.v`. Next a drift will be added.

```
DI = [];
DI.y.v = 1.5 + 3.*randn(1, 1e3);
DI.y.v = DI.y.v + [1:1:1e3]./100;
```

Lets suppose a sampling frequency is 1 Hz. The algorithm will generate all possible tau values automatically.

```
DI.fs.v = 1;
```

Call algorithm

Use QWTB to apply algorithm `ADEV` to data `DI`.

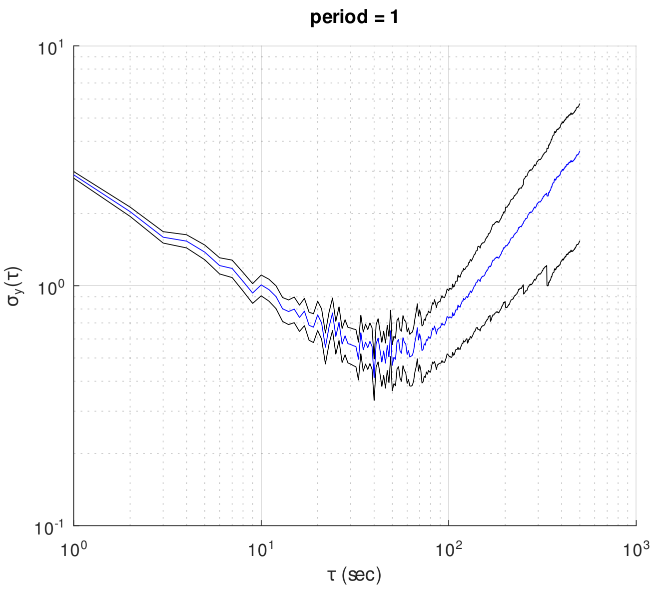
```
DO = qwtb('ADEV', DI);
```

```
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(1/DI.fs.v)]);
grid('on'); hold off
```



flicker_sim – Flickermeter simulator

Description

Id: flicker_sim

Name: Flickermeter simulator

Description: Calculates instantaneous flicker sensation Pinst and short-term flicker severity Pst.

Citation: Implemented according: IEC 61000-4-15, Electromagnetic compatibility (EMC), Testing and measurement techniques, Flickermeter, Edition 2.0, 2010-08; Wilhelm Mombauer: "Messung von Spannungsschwankungen und Flickern mit dem IEC-Flickermeter", ISBN 3-8007-2525-8, VDE-Verlag; Solcept Open Source Flicker Measurement-Simulator `https://www.solcept.ch/en/tools/flickersim/`; NPL Reference Flickermeter Design `http://www.npl.co.uk/electromagnetics/electrical-measurement/products-and-services/npl-reference-flickermeter-design`

Remarks: If sampling frequency |fs| is not supplied, wrapper will calculate |fs| from sampling time |Ts| or if not supplied, mean of differences of time series |t| is used to calculate |Ts|. Sampling frequency has to be higher than 7 kHz. If sampling f. is higher than 23 kHz, signal will be down sampled by algorithm. More than 600 s of signal is required. Requires either Signal Processing Toolbox when run in MATLAB or signal package when run in GNU Octave. Frequency of line (carrier frequency) |f_line| can be only 50 or 60 Hz

License: Boost Software License

Provides GUF: no

Provides MCM: no

Input Quantities

Required: fs or Ts or t, y, f_line

Descriptions:

- Ts – Sampling time
- f_line – Line frequency
- fs – Sampling frequency
- t – Time series
- y – Sampled values

Output Quantities:

- Pinst – Instantaneous flicker sensation
- Pst – Short-term flicker severity

Example

Contents

Generate sample data

A time series representing voltage measured on a power supply line will be generated. Modulation amplitude `dVV` in percents, modulation frequency `CPM` in changes per minute, line frequency `f_c`, and line amplitude `A_c` in volts are selected according Table 5 of EN61000-4-15/A1, line 4, collumn 3. Measurement time `siglen` and sampling frequency `f_s` are selected according recommendations of algorithm flicker_sim. Resulted Pst should be very near 1.

```
dVV = 0.894; CPM = 39; A_c = 230.*sqrt(2); f_c = 50; siglen =
   720; f_s = 20000;
% Frequency of the modulation (flicker) signal in hertz:
f_F = CPM / ( 60 * 2 );
% Time series:
t = linspace(0, siglen, siglen.*f_s);
```

```
% Sampled signal. Modulation is set in such way 10 minutes
   before end of signal modulation is zero.
y = A_c*sin(2*pi*f_c*t) .* ( 1 + (dVV/100)/2*sign(sin(2*pi*f_F*
   t - (siglen - 10).*f_F.*2.*pi)) );
```

Set input data.

```
DI = [];
DI.y.v = y;
DI.fs.v = f_s;
DI.f_line.v = f_c;
```

Call algorithm

Use QWTB to apply algorithm `flicker_sim` to data `DI`.

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

```
QWTB: no uncertainty calculation
warning: Invalid UTF-8 byte sequences have been replaced.
warning: called from
    alg_wrapper at line 25 column 14
    qwtb>check_and_run_alg at line 377 column 17
    qwtb at line 114 column 47
    publish>eval_code_helper at line 1079 column 8
    publish>eval_code at line 995 column 30
    publish at line 402 column 9
    all_algs_examples2tex at line 51 column 5
```

Display results

Short-term flicker severity:

```
Pst = DO.Pst
% Maximum of instantaneous flicker sensation:
Pinstmax = max(DO.Pinst)
```

```
error: max: wrong type argument 'scalar struct'
  in:

Pst = DO.Pst
% Maximum of instantaneous flicker sensation:
Pinstmax = max(DO.Pinst)
```

FourPSF – Standard Four Parameter Sine Wave Fit according IEEE Std 1241-2000

Description

Id: FourPSF

Name: Standard Four Parameter Sine Wave Fit according IEEE Std 1241-2000

Description: Fits a sine wave to the recorded data using 4 parameter (frequency, amplitude, phase and offset) model. The algorithm is according IEEE Standard for Terminology and Test methods for Analog-to-Digital Converters 1241-2000

Citation: IEEE Std 1241-2000, Implementation written by Zoltán Tamás Bilau, modified by Janos Markus. Id: sfit4.m,v 3.0 2004/04/19 11:20:09 markus Exp. Copyright (c) 2001-2004 by Istvan Kollar and Janos Markus. Modified 2016 Rado Lapuh

Remarks: If sampling time |Ts| is not supplied, wrapper will calculate |Ts| from sampling frequency |fs| or if not supplied, mean of differences of time series |t| is used to calculate |Ts|.

License: UNKNOWN

Provides GUF: no

Provides MCM: no

Input Quantities

Required: Ts or fs or t, y

Descriptions:

- `Ts` – Sampling time
- `fs` – Sampling frequency
- `t` – Time series
- `y` – Sampled values

Output Quantities:

- `A` – Amplitude of main signal component
- `O` – Offset of signal
- `f` – Frequency of main signal component
- `ph` – Phase of main signal component

Example

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;
```

Call algorithm

Use QWTB to apply algorithm `FourPSF` to data `DI`.

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

```
QWTB: no uncertainty calculation
QWTB: FourPSF wrapper: sampling time was calculated from time
   series
```

Display results

Results is the amplitude, frequency, phase and offset 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 = -4.4409e-10
ferrppm = 0
pherrppm = 8.8818e-10
Oerrppm = 5.5511e-10
```

FPNLSF – Four Parameter Non-Linear Sine Fit

Description

Id: FPNLSF

Name: Four Parameter Non-Linear Sine Fit

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

Citation:

Remarks: If Time series |t| is not supplied, wrapper will calculate |t| from sampling frequency |fs| or if not supplied, sampling time |Ts| is used to calculate |t|.

License: MIT License

Provides GUF: no

Provides MCM: no

Input Quantities

Required: t or fs or Ts, y, fest

Descriptions:

Ts – Sampling time

`fest` – Estimate of signal frequency
`fs` – Sampling frequency
`t` – Time series
`y` – Sampled values

Output Quantities:

`A` – Amplitude of main signal component
`O` – Offset of signal
`f` – Frequency of main signal component
`ph` – Phase of main signal component

Example

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;
```

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 `FPNLSF` to data `DI`.

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

```
QWTB: no uncertainty calculation
Fitting started
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 = 100.000
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 = -7.7716e-10
ferrppm = -4.0927e-08
pherrppm = 5.6228e-06
Oerrppm = -4.1217e-08
```

GenNHarm – Basic signal generator

Description

Id: GenNHarm

Name: Basic signal generator

Description: An algorithm for generating sampled waveforms with multiple harmonic or interharmonic components and noise level.

Citation: N/A

Remarks: If times of samples |t| are not defined, wrapper will calculate |t| from: sampling frequency |fs| or sampling period |Ts|, and from: number of samples |L| or number of main signal periods |M|. Vectors |f|, |A|, |ph|, |O| defines harmonic frequencies. First value is the main signal component. If |f|, |A|, |ph|, |O| are scalar, and |thd_k1|>0 and |nharm|>1, than harmonics are added to the signal to make required |thd_k1|.

License: MIT License

Provides GUF: no

Provides MCM: no

Input Quantities

Required: Ts or fs or t

Optional: M, noise

Parameters: noise

Descriptions:

M – Number of main signal component periods
Ts – Sampling time
fs – Sampling frequency

`noise` – Noise level of the signal
`t` – Time series

Output Quantities:

`thd_k1` – Total harmonic distortion

Example

Contents

Generate simple sine waveform

Consider we want to simulate a sampled voltage waveform with frequency 1, amplitude 1, phase 1 and offset 1. The waveform was sampled with sampling frequency 1 kHz and number of samples was 1000. The waveform contains also 3rd harmonic with amplitude of 0.1 V, and noise at level 1 mV.

```
DI = [];
DI.fs.v = 1e3;
DI.L.v = 1e3;
DI.f.v = [1 3];
DI.A.v = [1 0.1];
DI.ph.v = [0 pi];
DI.O.v = [0 0];
DI.noise.v = 0.001;
DO = qwtb('GenNHarm', DI);
```

```
QWTB: no uncertainty calculation
QWTB: GenNHarm wrapper: time series was calculated from
   sampling frequency and number of samples.
```

```
QWTB: GenNHarm wrapper: time series was calculated from number
    of samples.
```

Calculated THD_k1

The output structure contains the calculated value of the THD_k1 quantity:

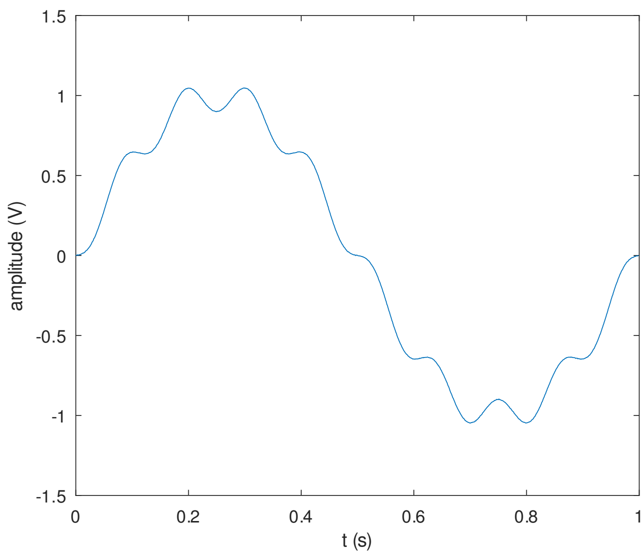
```
DO.thd_k1.v
```

```
ans = 0.1000
```

Plot

The generated waveform:

```
figure
plot(DO.t.v, DO.y.v, '-');
xlabel('t (s)'), ylabel('amplitude (V)')
```

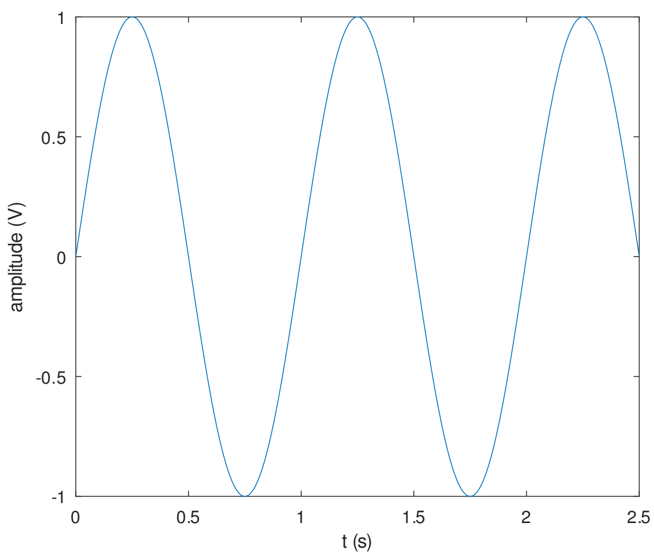


Generate simple sine waveform using number of periods

Consider we want to simulate 2.5 periods of a sine waveform.

```
DI = [];
DI.fs.v = 1e3;
DI.M.v = 2.5;
DI.f.v = [1];
DI.A.v = [1];
DI.ph.v = [0];
DI.O.v = [0];
DI.noise.v = 0;
DO = qwtb('GenNHarm', DI);
figure
plot(DO.t.v, DO.y.v, '-');
xlabel('t (s)'), ylabel('amplitude (V)')
```

```
QWTB: no uncertainty calculation
QWTB: GenNHarm wrapper: time series was calculated from
   sampling frequency and number of samples.
QWTB: GenNHarm wrapper: time series was calculated from number
    of main signal component periods.
```

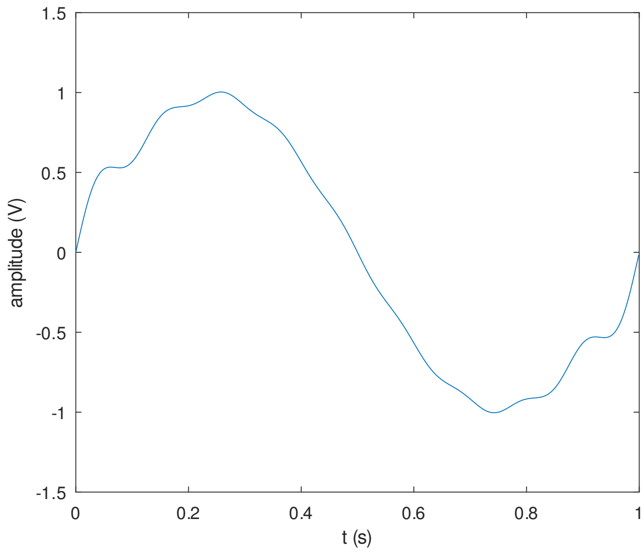


Generate waveform with automatically calculated harmonics

Consider waveform with 10 harmonic components, while the amplitudes of the harmonics will be calculated based on the supplied THD_k1 value. The `f`, `A`, `ph` and `O` quantities will contain only values for first harmonic. The rest harmonics will be added by the waveform generator.

```
DI = [];
DI.fs.v = 1e3;
DI.L.v = 1e3;
DI.f.v = [1];
DI.A.v = [1];
DI.ph.v = [0];
DI.O.v = [0];
DI.thd_k1.v = 0.1;
DI.nharm.v = 9;
DO = qwtb('GenNHarm', DI);
figure
plot(DO.t.v, DO.y.v, '-')
xlabel('t (s)'), ylabel('amplitude (V)')
```

```
QWTB: no uncertainty calculation
QWTB: GenNHarm wrapper: time series was calculated from
   sampling frequency and number of samples.
QWTB: GenNHarm wrapper: time series was calculated from number
    of samples.
```

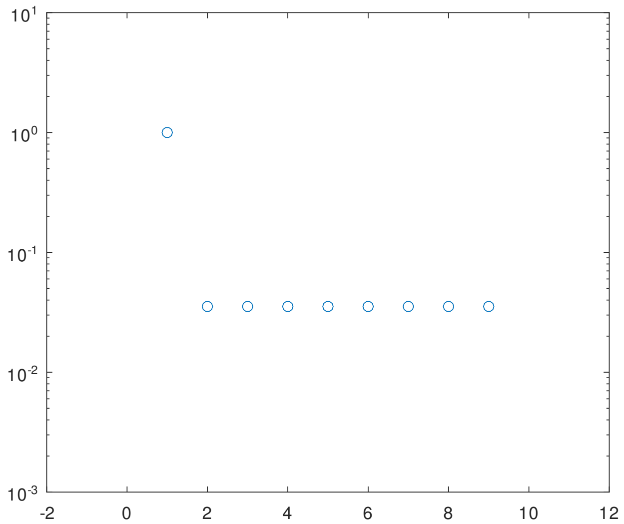


Check using FFT

We can check the generated waveform by calculating the spectrum. Output of GenNHarm will be used as input into the algorithm SP-WFFT.

```
DIspec = DO;
DIspec.fs.v = DI.fs.v;
DOspec = qwtb('SP-WFFT', DIspec);
figure
semilogy(DOspec.f.v, DOspec.A.v, 'o');
xlim([-2 12]); ylim([1e-3 10]);
```

```
QWTB: no uncertainty calculation
warning: axis: omitting non-positive data in log plot
warning: called from
    __plt__>__plt2vv__ at line 502 column 10
    __plt__>__plt2__ at line 248 column 14
    __plt__ at line 115 column 16
    semilogy at line 65 column 10
    publish>eval_code_helper at line 1079 column 8
    publish>eval_code at line 995 column 30
    publish at line 402 column 9
    all_algs_examples2tex at line 51 column 5
```


10^1
10^0
10^{-1}
10^{-2}
10^{-3}
-2
0
2
4
6
8
10
12

iDFT2p – 2-point interpolated DFT frequency estimator

Description

Id: iDFT2p

Name: 2-point interpolated DFT frequency estimator

Description: An algorithm for estimating the frequency, amplitude, phase and offset of the fundamental component using interpolated discrete Fourier transform. Rectangular or Hann window can be used for DFT.

Citation: Krzysztof Duda: Interpolation algorithms of DFT for parameters estimation of sinusoidal and damped sinusoidal signals. In S. M. Salih, editor, Fourier Transform - Signal Processing, chapter 1, pages 3-32, InTech, 2012. `http://www.intechopen.com/books/fourier-transform-signal-processing/interpolated-dft` . Source code from: Rado Lapuh, "Sampling with 3458A, Understanding, Programming, Sampling and Signal Processing", ISBN 978-961-94476-0-4, 1st. ed., Ljubljana, Left Right d.o.o., 2018

Remarks: If sampling time |Ts| is not supplied, wrapper will calculate |Ts| from sampling frequency |fs| or if not supplied, mean of differences of time series |t| is used to calculate |Ts|. The optional parameter |window| can be set to values 'rectangular' or 'Hann'. If parameter is not supplied, Hann window will be used.

License: Implementation: MIT License

Provides GUF: no

Provides MCM: no

Input Quantities

Required: Ts or fs or t, y
Optional: window
Parameters: window
Descriptions:

Ts – Sampling time
fs – Sampling frequency
t – Time series
window – DFT window: 'Hann' or 'rectangular'
y – Sampled values

Output Quantities:

A – Amplitude of main signal component
O – Offset of main signal component
f – Frequency of main signal component
ph – Phase of main signal component

Example

Contents

Generate sample data

Two quantities are prepared: `Ts` and `y`, representing 0.5 second of sinus waveform of nominal frequency 100 Hz, nominal amplitude 1 V and nominal phase 1 rad, sampled with sampling time 0.1 ms, with offset 0.1 V. The sampling is not coherent.

```
DI = [];
Anom = 1; fnom = 100; phnom = 1; Onom = 0.1;
DI.Ts.v = 1e-4;
t = [0:DI.Ts.v:0.5];
DI.y.v = Anom*sin(2*pi*fnom*t + phnom) + Onom;
```

Call algorithm

First a rectangular window will be selected to estimate main signal properties. Use QWTB to apply algorithm `iDFT3p` to data `DI` and put results into `DOr`.

```
DI.window.v = 'rectangular';
DOr = qwtb('iDFT2p', DI);
```

```
QWTB: no uncertainty calculation
```

Next a Hann window will be selected to estimate main signal properties Results will be put into `DOh`.

```
DI.window.v = 'Hann';
DOh = qwtb('iDFT2p', DI);
```

```
QWTB: no uncertainty calculation
```

Display results

Results is the amplitude, frequency and phase of sampled waveform. For rectangular window, the error from nominal in parts per milion is:

```
f_re = (DOr.f.v - fnom)./fnom .* 1e6
A_re = (DOr.A.v - Anom)./Anom .* 1e6
ph_re = (DOr.ph.v - phnom)./phnom .* 1e6
O_re = (DOr.O.v - Onom)./Onom .* 1e6
```

```
f_re = -0.8083
A_re = 40.315
ph_re = 217.94
O_re = 1682.6
```

For Hann window:

```
f_he = (DOh.f.v - fnom)./fnom .* 1e6
A_he = (DOh.A.v - Anom)./Anom .* 1e6
ph_he = (DOh.ph.v - phnom)./phnom .* 1e6
```

```
O_he = (DOh.O.v - Onom)./Onom .* 1e6
```

```
f_he = 1.8426e-04
A_he = -4.6039e-03
ph_he = 6.2442
O_he = -0.6862
```

iDFT3p – 3-point interpolated DFT frequency estimator

Description

Id: iDFT3p

Name: 3-point interpolated DFT frequency estimator

Description: An algorithm for estimating the frequency, amplitude, phase and offset of the fundamental component using interpolated discrete Fourier transform. Rectangular or Hann window can be used for DFT.

Citation: Krzysztof Duda: Interpolation algorithms of DFT for parameters estimation of sinusoidal and damped sinusoidal signals. In S. M. Salih, editor, Fourier Transform - Signal Processing, chapter 1, pages 3-32, InTech, 2012. `http://www.intechopen.com/books/fourier-transform-signal-processing/interpolated-dft` . Source code from: Rado Lapuh, "Sampling with 3458A, Understanding, Programming, Sampling and Signal Processing", ISBN 978-961-94476-0-4, 1st. ed., Ljubljana, Left Right d.o.o., 2018

Remarks: If sampling time |Ts| is not supplied, wrapper will calculate |Ts| from sampling frequency |fs| or if not supplied, mean of differences of time series |t| is used to calculate |Ts|. The optional parameter |window| can be set to values 'rectangular' or 'Hann'. If parameter is not supplied, Hann window will be used.

License: Implementation: MIT License

Provides GUF: no

Provides MCM: no

Input Quantities

Required: Ts or fs or t, y
Optional: window
Parameters: window
Descriptions:

Ts – Sampling time
fs – Sampling frequency
t – Time series
window – DFT window: ‘Hann‘ or ‘rectangular‘
y – Sampled values

Output Quantities:

A – Amplitude of main signal component
O – Offset of main signal component
f – Frequency of main signal component
ph – Phase of main signal component

Example

Contents

Generate sample data

Two quantities are prepared: `Ts` and `y`, representing 0.5 second of sinus waveform of nominal frequency 100 Hz, nominal amplitude 1 V and nominal phase 1 rad, sampled with sampling time 0.1 ms, with offset 0.1 V. The sampling is not coherent.

```
DI = [];
Anom = 1; fnom = 100; phnom = 1; Onom = 0.1;
DI.Ts.v = 1e-4;
t = [0:DI.Ts.v:0.5];
DI.y.v = Anom*sin(2*pi*fnom*t + phnom) + Onom;
```

Call algorithm

First a rectangular window will be selected to estimate main signal properties. Use QWTB to apply algorithm `iDFT3p` to data `DI` and put results into `DOr`.

```
DI.window.v = 'rectangular';
DOr = qwtb('iDFT3p', DI);
```

```
QWTB: no uncertainty calculation
```

Next a Hann window will be selected to estimate main signal properties Results will be put into `DOh`.

```
DI.window.v = 'Hann';
DOh = qwtb('iDFT3p', DI);
```

```
QWTB: no uncertainty calculation
```

Display results

Results is the amplitude, frequency and phase of sampled waveform. For rectangular window, the error from nominal in parts per milion is:

```
f_re = (DOr.f.v - fnom)./fnom .* 1e6
A_re = (DOr.A.v - Anom)./Anom .* 1e6
ph_re = (DOr.ph.v - phnom)./phnom .* 1e6
O_re = (DOr.O.v - Onom)./Onom .* 1e6
```

```
f_re = 0.016607
A_re = 41.257
ph_re = 88.368
O_re = 1682.6
```

For Hann window:

```
f_he = (DOh.f.v - fnom)./fnom .* 1e6
A_he = (DOh.A.v - Anom)./Anom .* 1e6
ph_he = (DOh.ph.v - phnom)./phnom .* 1e6
```

```
O_he = (DOh.O.v - Onom)./Onom .* 1e6
```

```
f_he = -3.7790e-06
A_he = 3.9702e-07
ph_he = 6.2737
O_he = -0.6862
```

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

Description

Id: INL-DNL

Name: Integral and Differential Non-Linearity of ADC

Description: 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

Provides GUF: no

Provides MCM: no

Input Quantities

Required: bitres, codes

Descriptions:

bitres – Bit resolution of an ADC

codes – Sampled values represented as ADC codes (not converted to voltage)

Output Quantities:

DNL – Differential Non-Linearity
INL – Integral Non-Linearity

Example

Contents

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;
```

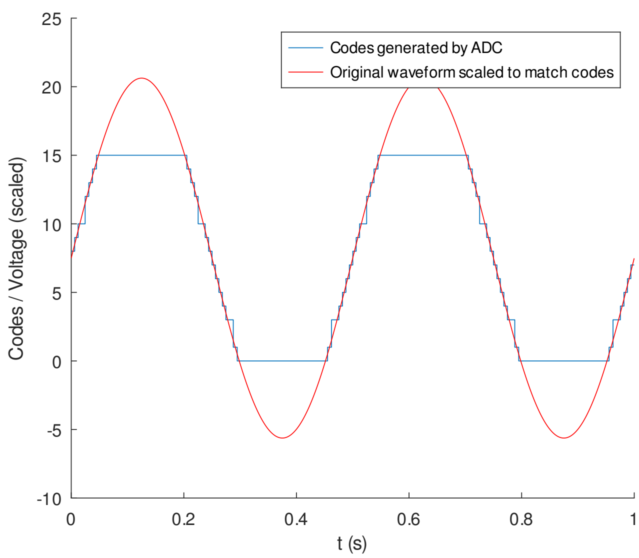
```
codes = round((codes-rmin)./(rmax-rmin).*levels);
```

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

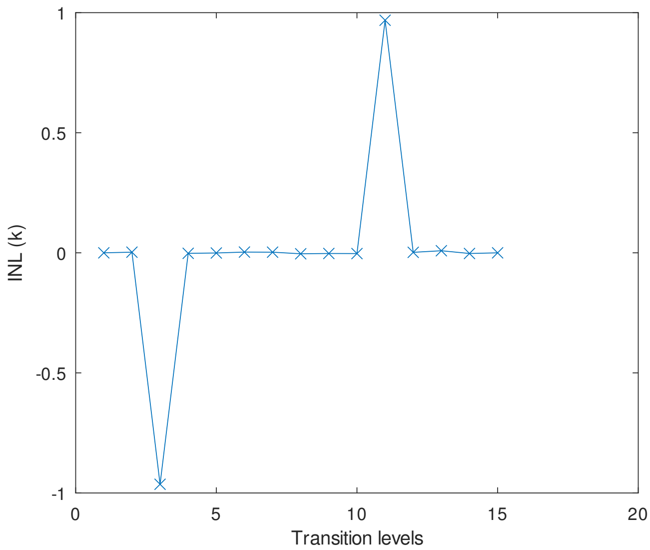
Apply INL algorithm to the input data DI.

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

```
QWTB: no uncertainty calculation
warning: Invalid UTF-8 byte sequences have been replaced.
warning: called from
    ProcessHistogramTest at line 52 column 3
    alg_wrapper at line 35 column 5
    qwtb>check_and_run_alg at line 377 column 17
    qwtb at line 114 column 47
    publish>eval_code_helper at line 1079 column 8
    publish>eval_code at line 995 column 30
    publish at line 402 column 9
    all_algs_examples2tex at line 51 column 5

warning: Invalid UTF-8 byte sequences have been replaced.
warning: called from
    ProcessHistogramTest at line 273 column 14
    alg_wrapper at line 35 column 5
    qwtb>check_and_run_alg at line 377 column 17
    qwtb at line 114 column 47
    publish>eval_code_helper at line 1079 column 8
    publish>eval_code at line 995 column 30
    publish at line 402 column 9
    all_algs_examples2tex at line 51 column 5
```

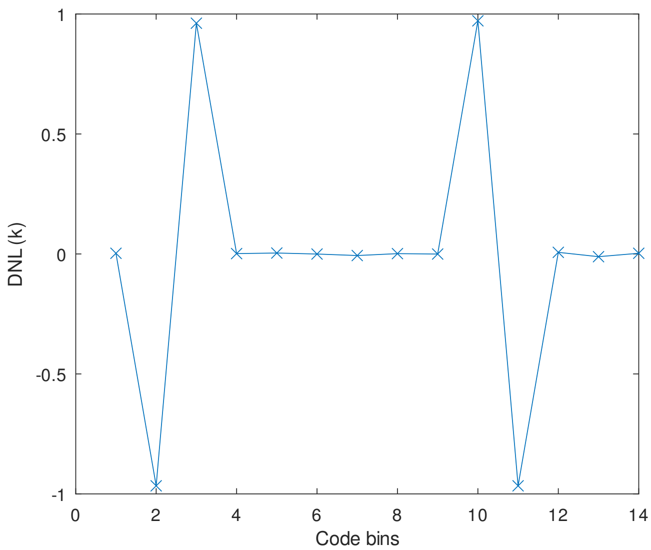
Plot results of integral non-linearity. One can clearly observe defects on codes 3 and 11.

```
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)')
```



ISOTS28037 – NPL's Software to Support ISO/TS 28037:2010(E)

Description

Id: ISOTS28037

Name: NPL's Software to Support ISO/TS 28037:2010(E)

Description: NPL's Software to Support ISO/TS 28037:2010(E) software implements the algorithms described in the ISO Technical Specification "Determination and use of straight-line calibration functions" and has been developed by the National Physical Laboratory (NPL) in the United Kingdom. The software is available as a compressed ZIP folder from the web sites of NPL at www.npl.co.uk/mathematics-scientific-computing/software-support-for-metrology/software-downloads-(ssfm) and the International Organization for Standardization at standards.iso.org/iso/ts/28037/. Downloadable at https://www.npl.co.uk/resources/software/iso-ts-28037-2010e

Citation: `https://standards.iso.org/iso/ts/28037/`, `https://www.npl.co.uk/resources/software/iso-ts-28037-2010e`

Remarks: Implements methods WSL, GDR. GMR nor GGMR are not implemented.

License: NPL license

Provides GUF: yes

Provides MCM: no

Input Quantities

Required: `x`, `y`

Optional: tol, xhat

Parameters: tol

Descriptions:

- tol – Tolerance for convergence of iterative method GDR. Iterations stops when increments of parameters a, b is smaller than tol*norm([a b]).
- x – Independent variable
- xhat – Independent variable values to be (extra/inter)polated (forward evaluated).
- y – Dependent variable

Output Quantities:

- chisq – 95
- coefs – Fitted coefficients
- exponents – Exponents of polynomial used to fit, for now only [0 1] possible
- func – Inline function constructed for exponents with parameters ‘x‘ and ‘coefs.v‘
- model – Model used for calculation.
- model_rejected – If 1, straight-line model is rejected. Estimated as chi_sq_obs > chi_sq.
- yhat – Fitted values y

Example

Contents


```
% Example for algorithm ISOTS28037
```

Generate sample data

Set independent and dependent variables.

```
DI.x.v = [1 2 3 4 5];
DI.x.u = [0.1 0.1 0.1 0.1 0.1];
DI.y.v = [1.1 1.9 3.1 3.9 5.1];
DI.y.u = [0.1 0.1 0.1 0.1 0.1];

% Set values for interpolation:
DI.xhat.v = [0:0.1:6];
DI.xhat.u = 0.1 + zeros(size(DI.xhat.v));
```

Call algorithm

Use QWTB to apply algorithm `ISOTS28037` to data `DI`.

```
DO = qwtb('ISOTS28037', DI);
```

```
QWTB: no uncertainty calculation
```

Display results

Results is

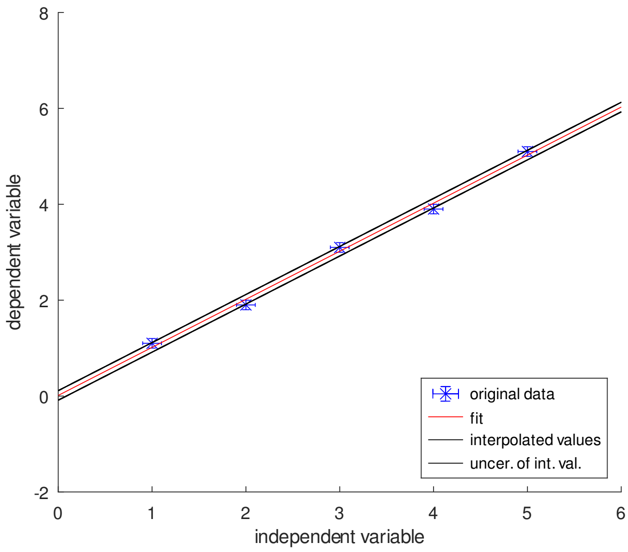
```
disp(['offset         : ' num2str(DO.coefs.v(1)) ' +- '
   num2str(DO.coefs.u(1))])
disp(['linear coeff.  : ' num2str(DO.coefs.v(2)) ' +- '
   num2str(DO.coefs.u(2))])
```

```
offset          : 1.0024 +- 0.044802
linear coeff.   : 0.012791 +- 0.14858
```

Plot results

```
figure
hold on
errorbar(DI.x.v, DI.y.v, DI.x.u, DI.x.u, DI.y.u, DI.y.u, '~>xb'
   )
```

```
plot(DI.xhat.v, DO.yhat.v, 'r-')
plot(DI.xhat.v, DO.yhat.v + DI.xhat.u, 'k-')
plot(DI.xhat.v, DO.yhat.v - DI.xhat.u, 'k-')
xlabel('independent variable')
ylabel('dependent variable')
legend('original data','fit', 'interpolated values', 'uncer. of
    int. val.','location','southeast')
hold off
```



MADEV – Modified Allan Deviation

Description

Id: MADEV

Name: Modified Allan Deviation

Description: 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 sampling frequency |fs| is not supplied, wrapper will calculate |fs| from sampling time |Ts| or if not supplied, mean of differences of time series |t| is used to calculate |Ts|. If observation time(s) |tau| is not supplied, 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.

License: BSD License

Provides GUF: yes

Provides MCM: no

Input Quantities

Required: fs or Ts or t, y

Optional: `tau`

Descriptions:

`Ts` – Sampling time
`fs` – Sampling frequency
`t` – Time series
`tau` – Observation time
`y` – Sampled values

Output Quantities:

`madev` – Modified Allan deviation
`tau` – Observation time of resulted values

Example

Contents

Generate sample data

A random numbers with normal probability distribution function will be generated into input data `DI.y.v`. Next a drift will be added.

```
DI = [];
DI.y.v = 1.5 + 3.*randn(1, 1e3);
DI.y.v = DI.y.v + [1:1:1e3]./100;
```

Lets suppose a sampling frequency is 1 Hz. The algorithm will generate all possible tau values automatically.

```
DI.fs.v = 1;
```

Call algorithm

Use QWTB to apply algorithm `MADEV` to data `DI`.

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

```
QWTB: no uncertainty calculation
```

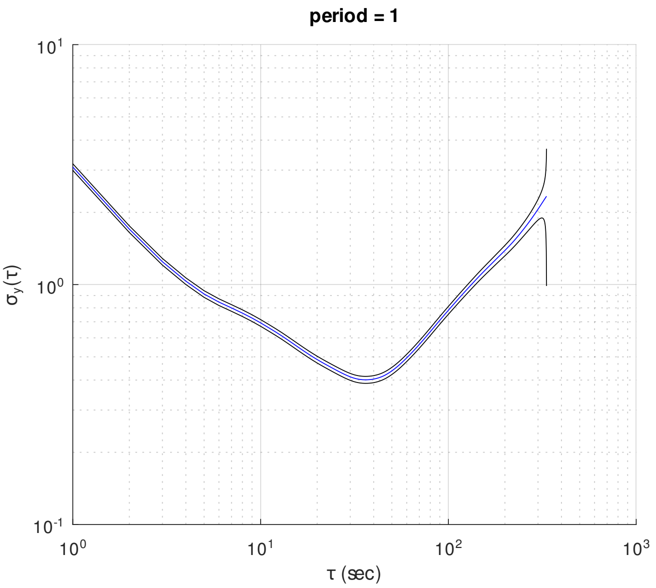
Display results

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
```

```
warning: axis: omitting non-positive data in log plot
warning: called from
    __plt__>__plt2vv__ at line 502 column 10
    __plt__>__plt2__ at line 248 column 14
    __plt__ at line 115 column 16
    loglog at line 65 column 10
    publish>eval_code_helper at line 1079 column 8
    publish>eval_code at line 995 column 30
    publish at line 402 column 9
    all_algs_examples2tex at line 51 column 5

warning: axis: omitting non-positive data in log plot
warning: called from
    __plt__>__plt2vv__ at line 502 column 10
    __plt__>__plt2__ at line 248 column 14
    __plt__ at line 115 column 16
    loglog at line 65 column 10
    publish>eval_code_helper at line 1079 column 8
    publish>eval_code at line 995 column 30
    publish at line 402 column 9
```

```
    all_algs_examples2tex at line 51 column 5

warning: axis: omitting non-positive data in log plot
warning: called from
    __plt__>__plt2vv__ at line 502 column 10
    __plt__>__plt2__ at line 248 column 14
    __plt__ at line 115 column 16
    loglog at line 65 column 10
    publish>eval_code_helper at line 1079 column 8
    publish>eval_code at line 995 column 30
    publish at line 402 column 9
    all_algs_examples2tex at line 51 column 5

warning: axis: omitting non-positive data in log plot
warning: called from
    __plt__>__plt2vv__ at line 502 column 10
    __plt__>__plt2__ at line 248 column 14
    __plt__ at line 115 column 16
    loglog at line 65 column 10
    publish>eval_code_helper at line 1079 column 8
    publish>eval_code at line 995 column 30
    publish at line 402 column 9
    all_algs_examples2tex at line 51 column 5

warning: axis: omitting non-positive data in log plot
warning: called from
    __plt__>__plt2vv__ at line 502 column 10
    __plt__>__plt2__ at line 248 column 14
    __plt__ at line 115 column 16
    loglog at line 65 column 10
    publish>eval_code_helper at line 1079 column 8
    publish>eval_code at line 995 column 30
    publish at line 402 column 9
    all_algs_examples2tex at line 51 column 5
```


period = 1
10^1
10^0
10^{-1}
$\sigma_y(\tau)$
10^0
10^1
10^2
10^3
τ (sec)

OADEV – Overlapping Allan Deviation

Description

Id: OADEV

Name: Overlapping Allan Deviation

Description: 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 sampling frequency |fs| is not supplied, wrapper will calculate |fs| from sampling time |Ts| or if not supplied, mean of differences of time series |t| is used to calculate |Ts|. If observation time(s) |tau| is not supplied, 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.

License: BSD License

Provides GUF: yes

Provides MCM: no

Input Quantities

Required: fs or Ts or t, y
Optional: tau
Descriptions:

- Ts – Sampling time
- fs – Sampling frequency
- t – Time series
- tau – Observation time
- y – Sampled values

Output Quantities:

- oadev – Overlapping Allan deviation
- tau – Observation time of resulted values

Example

Contents

Generate sample data

A random numbers with normal probability distribution function will be generated into input data DI.y.v. Next a drift will be added.

```
DI = [];
DI.y.v = 1.5 + 3.*randn(1, 1e3);
DI.y.v = DI.y.v + [1:1:1e3]./100;
```

Lets suppose a sampling frequency is 1 Hz. The algorithm will generate all possible tau values automatically.

```
DI.fs.v = 1;
```

Call algorithm

Use QWTB to apply algorithm `OADEV` to data `DI`.

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

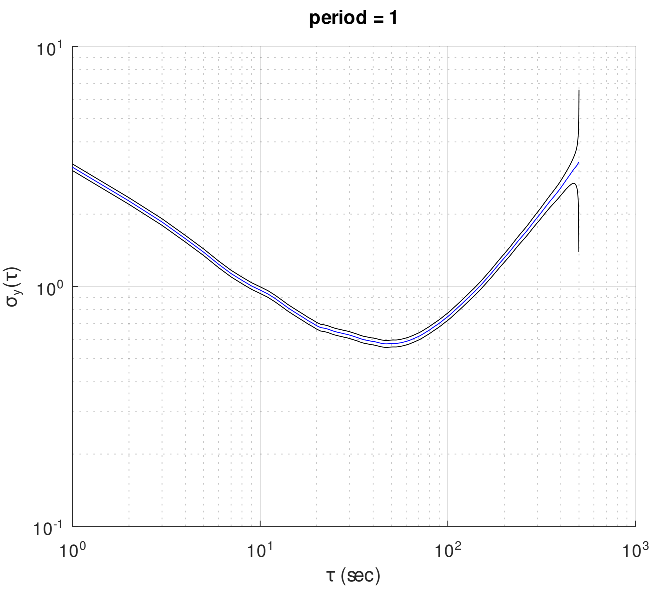
```
QWTB: no uncertainty calculation
```

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
```

```
warning: axis: omitting non-positive data in log plot
warning: called from
    __plt__>__plt2vv__ at line 502 column 10
    __plt__>__plt2__ at line 248 column 14
    __plt__ at line 115 column 16
    loglog at line 65 column 10
    publish>eval_code_helper at line 1079 column 8
    publish>eval_code at line 995 column 30
    publish at line 402 column 9
    all_algs_examples2tex at line 51 column 5
```


period = 1
10^1
10^0
10^{-1}
$\sigma_y(\tau)$
10^0
10^1
10^2
10^3
τ (sec)

PSFE – Phase Sensitive Frequency Estimator

Description

Id: PSFE

Name: Phase Sensitive Frequency Estimator

Description: 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. PSFE requires more than two periods of sampled signal and at least 6 samples in the Record.

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`. Source code from: Rado Lapuh, "Sampling with 3458A, Understanding, Programming, Sampling and Signal Processing", ISBN 978-961-94476-0-4, 1st. ed., Ljubljana, Left Right d.o.o., 2018

Remarks: If sampling time |Ts| is not supplied, wrapper will calculate |Ts| from sampling frequency |fs| or if not supplied, mean of differences of time series |t| is used to calculate |Ts|.

License: MIT License

Provides GUF: no

Provides MCM: no

Input Quantities

Required: Ts or fs or t, y

Descriptions:

Ts – Sampling time
fs – Sampling frequency
t – Time series
y – Sampled values

Output Quantities:

A – Amplitude of main signal component
O – Offset of main signal component
f – Frequency of main signal component
ph – Phase of main signal component

Example

Contents

Generate sample data

Two quantities are prepared: `Ts` and `y`, representing 1 second of sinus waveform of nominal frequency 100 Hz, nominal amplitude 1 V and nominal phase 1 rad, sampled with sampling time 0.1 ms.

```
DI = [];
Anom = 1; fnom = 100; phnom = 1;
DI.Ts.v = 1e-4;
t = [0:DI.Ts.v:1-DI.Ts.v];
DI.y.v = Anom*sin(2*pi*fnom*t + 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 = 100.000
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 = -0.014954
Aerrppm = 3.6938
pherrppm = 25.678
```

SFDR – Spurious Free Dynamic Range

Description

Id: SFDR

Name: Spurious Free Dynamic Range

Description: Calculates Spurious Free Dynamic Range of a signal based on an amplitude spectrum.

Citation: Implementation: Martin Sira

Remarks: Samples are expected in quantity ‘y‘, and algorithm ‘SP-WFFT‘ is used to calculate the amplitude spectrum. Alternatively a spectrum can be directly delivered in quantity ‘A‘, use of blackman DFT window is expected.

License: MIT

Provides GUF: no

Provides MCM: no

Input Quantities

Required: `y` or `A`

Descriptions:

`A` – Amplitude spectrum
`y` – Sampled values

Output Quantities:

`SFDR` – Spurious Free Dynamic Range, relative to carrier (V/V)

SFDRdBc – Spurious Free Dynamic Range, relative to carrier, in decibel (dB)

Example

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
QWTB: no uncertainty calculation
```

Display results

Result is the SFDR (dBc).

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

```
SFDR = 40.000
```

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)

Description: 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: If Time series |t| is not supplied, wrapper will calculate |t| from sampling frequency |fs| or if not supplied, sampling time |Ts| is used to calculate |t|.

License: MIT License

Provides GUF: no

Provides MCM: no

Input Quantities

Required: t or fs or Ts, y, f, A, ph, O, bitres, FSR

Descriptions:

A – Amplitude of main signal component
FSR – Full scale range of an ADC
O – Offset of signal
Ts – Sampling time
bitres – Bit resolution of an ADC
f – Frequency of main signal component
fs – Sampling frequency
ph – Phase of main signal component
t – Time series
y – Sampled values

Output Quantities:

ENOB – Effective number of bits

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

---

Example

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 `FPNLSF` to data `DI`.

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

```
QWTB: no uncertainty calculation
Fitting started
Fitting finished
```

Copy results to inputs

Take results of `FPNLSF` 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 `FSR` 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.FSR.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 = 82.936
ENOB = 13.068
```

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.FSR.v./(noisestd.*sqrt(12)));
SINADerror = SINADdB - SINADdBtheor
ENOBerror = ENOB - ENOBtheor
```

```
SINADerror = -0.074709
ENOBerror = -0.012410
```

SP-WFFT – Spectrum by means of Windowed Discrete Fourier Transform

Description

Id: SP-WFFT

Name: Spectrum by means of Windowed Discrete Fourier Transform

Description: Calculates amplitude and phase spectrum by means of Discrete Fourier Transform with windowing and/or zero padding. Result is normalized. Follwing windows are implemented: barthann, bartlett, blackman, blackmanharris, blackmannuttall, bohman, cheb, flattop_matlab, flattop_SFT3F, flattop_SFT4F, flattop_SFT5F, flattop_SFT3M, flattop_SFT4M, flattop_SFT5M, flattop_248D, gaussian, hamming, hanning, kaiser, nuttall, parzen, rect, triang, tukey, welch.

Citation: Various sources. See algorithm scripts for details. A. V. Oppenheim and R. W. Schafer, Discrete-Time Signal Processing. Peter Lynch, "The Dolph-Chebyshev Window: A Simple Optimal Filter", Monthly Weather Review, Vol. 125, pp. 655-660, April 1997, `http://www.maths.tcd.ie/~plynch/Publications/Dolph.pdf` . C. Dolph, "A current distribution for broadside arrays which optimizes the relationship between beam width and side-lobe level", Proc. IEEE, 34, pp. 335-348. `https://www.mathworks.com/help/signal/ref/flattopwin.html` . D‘Antona, Gabriele, and A. Ferrero. Digital Signal Processing for Measurement Systems. New York: Springer Media, 2006, pp. 70–72. Gade, Svend, and Henrik Herlufsen. "Use of Weighting Functions in DFT/FFT Analysis (Part I)." Windows to FFT Analysis (Part I): Bruel and Kjaer Technical Review, No. 3, 1987, pp. 1-28. G. Heinzel, ‘Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new flat-top windows‘, IEEE, 2003. Fredric J. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform, Proceedings of the IEEE", Vol. 66, No. 1, January 1978, Page 67, Equation 38. Implemented by: Sylvain Pelissier, Andreas Weingessel, Muthiah Annamalai, Andre Carezia, Vera Nováková Zachovalova, Paul Kienzle, Paul Kienzle, Laurent Mazet, Muthiah Annamalai, Mike Gross, Peter V. Lanspeary.

Remarks: If sampling frequency |fs| is not supplied, wrapper will calculate |fs| from sampling time |Ts| or if not supplied, first two elements of time series |t| are used to calculate |fs|. If |window| is not specified, a rectangular (none) window will be used. If |window| is ‘cheb‘ and |cheb_att| is not specified, a value of 100 dB is set. If |window| is ‘gaussian‘ and |gaussian_width| is not specified, a value of 1 is set. If |window| is ‘kaiser‘ and |kaiser_att| is not specified, a value of 0.5 is set. If |window| is ‘tukey‘ and |tukey_ratio| is not specified, a value of 0.5 is set.

License: Mixed license - every window function has its own license.

Provides GUF: no

Provides MCM: no

Input Quantities

Required: fs or Ts or t, y

Optional: window, cheb_att, gaussian_width, kaiser_att, fft_padding

Parameters: window, cheb_att, gaussian_width, kaiser_att, fft_padding

Descriptions:

Ts – Sampling time

cheb_att – Only for Dolph-Chebyshev window: stop-band attenuation in dB

fft_padding – Zero padding of signal in samples

fs – Sampling frequency

gaussian_width – Only for Gaussian window: width of the window in Hz

kaiser_att – Only for Kaiser window: stop-band attenuation in FFT bins

t – Time series

window – Name of window function

y – Sampled values

Output Quantities:

A – Amplitude spectrum
ENBW – Effective Noise BandWidth
NENBW – Normalized Equivalent Noise BandWidth
NL – Average spectral noise level
NLD – Average spectral density noise level
SD – Spectral density
SNR – Signal to noise ratio
SNRdB – Signal to noise ratio in decibels
f – Frequency series
noise_rms – RMS noise amplitude
ph – Phase spectrum
w – Window coefficients
w – Window coefficients

Example

Contents

Generate sample data

Construct 1 second of signal sampled at 50 Hz containing two harmonic components at 1 and 8 Hz and one interharmonic component at 15.5 Hz with various amplitudes and phases.

```
clear DI
DI.fs.v = 50;
fnom = [1;    8; 15.5];
Anom = [1; 0.5;  0.3];
pnom = [0;    1;    2];
DI.t.v = [0 : 1/DI.fs.v : 1 - 1/DI.fs.v];
DI.y.v = zeros(size(DI.t.v));
```

```
for i = 1:length(fnom)
        DI.y.v = DI.y.v + Anom(i).*sin(2.*pi.*fnom(i).*DI.t.v +
    pnom(i));
end
%
```

Call algorithm

Calculate amplitude and phase spectrum and store results into DO. Window function is not specified, therefore rectangle (none) window will be used.

```
DO = qwtb('SP-WFFT', DI);
%
% Set window function to blackman and calculate windowed
   amplitude and phase spectrum and store results into |DOw|.
DI.window.v = 'blackman';
DOw = qwtb('SP-WFFT', DI);
%
% Set zero padding to 10 times the signal length and calculate
   zero padded windowed amplitude and
% phase spectrum and stre results into |DOwz|.
DI.fft_padding.v = 10.*length(DI.y.v);
DOwz = qwtb('SP-WFFT', DI);
%
```

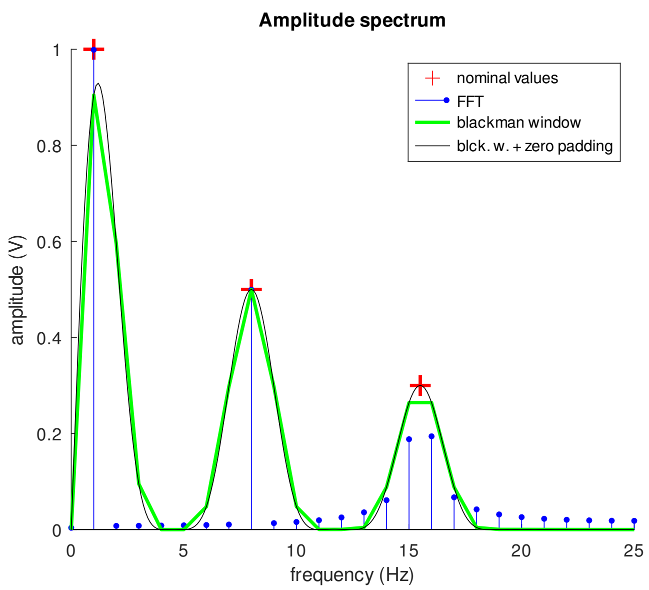
```
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
```

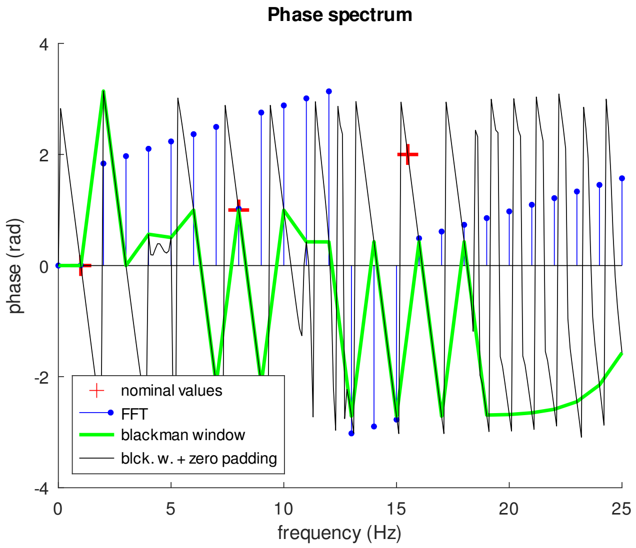
Display results

Plot amplitude spectrum.

```
figure; hold on;
plot(fnom, Anom, '+r', 'markersize', 10, 'linewidth', 2);
stem(DO.f.v, DO.A.v, '-ob', 'filled', 'markersize', 3);
plot(DOw.f.v, DOw.A.v, '-g','linewidth',2);
plot(DOwz.f.v, DOwz.A.v, '-k');
```

```
legend('nominal values','FFT', 'blackman window', 'blck. w. +
   zero padding')
title('Amplitude spectrum')
xlabel('frequency (Hz)'); ylabel('amplitude (V)');
hold off
% Plot phase spectrum
figure; hold on;
plot(fnom, pnom, '+r', 'markersize', 10, 'linewidth', 2);
stem(DO.f.v, DO.ph.v, '-ob', 'filled', 'markersize', 3);
plot(DOw.f.v, DOw.ph.v, '-g','linewidth',2);
plot(DOwz.f.v, DOwz.ph.v, '-k');
legend('nominal values','FFT', 'blackman window', 'blck. w. +
   zero padding', 'location', 'southwest')
title('Phase spectrum');
xlabel('frequency (Hz)'); ylabel('phase (rad)');
hold off
%
```





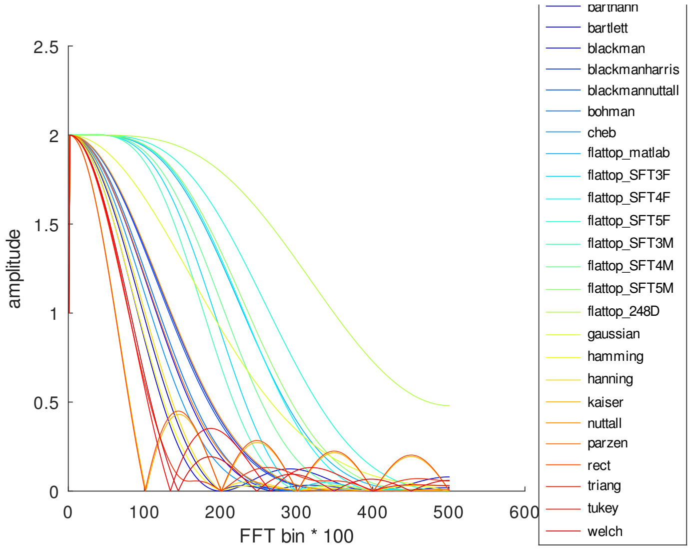
Compare window coefficients

Different windows has different peak widths, heights of side lobes and side lobes roll off ratio. To see the window differences a zero padding is used. First a simple signal of ones is prepared and zero padding is specified to 100x the length of signal. Next for every window a spectrum is calculated and plotted.

```
clear DI
DI.y.v = ones(1,10);
DI.fs.v = 1;
DI.fft_padding.v = 100.*length(DI.y.v);
avail_windows = {'barthann' 'bartlett' 'blackman' '
   blackmanharris' 'blackmannuttall' 'bohman' 'cheb' '
   flattop_matlab' 'flattop_SFT3F' 'flattop_SFT4F' '
   flattop_SFT5F' 'flattop_SFT3M' 'flattop_SFT4M' '
   flattop_SFT5M' 'flattop_248D' 'gaussian' 'hamming' 'hanning
   ' 'kaiser' 'nuttall' 'parzen' 'rect' 'triang' 'tukey' '
   welch'};
col = jet(length(avail_windows));
figure; hold on;
for i = 1:length(avail_windows)
        DI.window.v = avail_windows{i};
        DO = qwtb('SP-WFFT', DI);
        plot(DO.A.v, '-', 'color', col(i,:));
end
h=legend(avail_windows);
```

```
h=legend('location','eastoutside');
set(h,'FontSize',8, 'interpreter','none');
xlabel('FFT bin * 100');
ylabel('amplitude');
hold off;
```

```
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
QWTB: no uncertainty calculation
```


amplitude
2.5
2
1.5
1
0.5
0
0
100
200
300
400
500
600
FFT bin * 100
bartlett
blackman
blackmanharris
blackmannuttall
bohman
cheb
flattop_matlab
flattop_SFT3F
flattop_SFT4F
flattop_SFT5F
flattop_SFT3M
flattop_SFT4M
flattop_SFT5M
flattop_248D
gaussian
hamming
hanning
kaiser
nuttall
parzen
rect
triang
tukey
welch

SplineResample – Spline Resample

Description

Id: SplineResample

Name: resampling name

Description: Splines are used to resample sampled data to a new sampling frequency.

Citation: no citation

Remarks: no remark

License: no license

Provides GUF: no

Provides MCM: no

Input Quantities

> **Required:** Ts or fs or t, y, fest
>
> **Optional:** method, D
>
> **Parameters:** method, D
>
> **Descriptions:**
>
> D – Denominator to reduce resampled bandwidth
> Ts – Sampling period
> fest – Estimate of fundamental component frequency
> fs – Sampling frequency
> method – Method of resampling: ‘keepN‘, ‘minimizefs‘, ‘poweroftwo‘
> t – Time series
> y – Sampled values

Output Quantities:

Pf – Integer upsampling factor
Qf – Integer decimation factor
Ts – Sampling period after resampling, equal to fsest.
fs – Sampling frequency after resampling, equal to fsest.
method – Method of resampling.
t – Time series after resampling, equal to test
y – Resampled samples

Example

Contents

Generate sampled data

Three quantities have to be prepared: time series t and signal y, representing 2000 samples of sinus waveform of nominal frequency 49.77 Hz, nominal amplitude 1 V and nominal phase 0 rad, sampled with sampling period 0.25 ms. Signal simulates non-coherent sampling.

```
f = 49.77;
A = 1;
N = 2000;
fs = 4000;
Ts = 1/fs;
Sampled.t.v = [0 : N-1] * Ts;
Sampled.y.v = A*sin(2 * pi * f * Sampled.t.v);
```

Signal frequency estimate

Get estimate of signal frequency to be coherent after resampling. For example, algorithm PSFE can be used:

```
Estimate = qwtb('PSFE', Sampled);
Sampled.fest.v = Estimate.f.v;
```

```
QWTB: no uncertainty calculation
QWTB: PSFE wrapper: sampling time was calculated from time
   series
```

Call algorithm

```
Resampled = qwtb('SplineResample', Sampled);
```

```
QWTB: no uncertainty calculation
QWTB: SplineResample wrapper: sampling time was calculated
   from time series
```

Get spectra

```
SpectrumNonCoherent = qwtb('SP-WFFT', Sampled);
SpectrumResampled = qwtb('SP-WFFT', Resampled);
```

```
QWTB: no uncertainty calculation
QWTB: SP-WFFT wrapper: sampling frequency was calculated from
   time series
warning: QWTB: value of quantity `y` is column vector, it was
   automatically transposed.
warning: called from
    qwtb>check_gen_datain at line 969 column 25
    qwtb>check_and_run_alg at line 342 column 16
    qwtb at line 114 column 47
    publish>eval_code_helper at line 1079 column 8
    publish>eval_code at line 995 column 30
```

```
    publish at line 402 column 9
    all_algs_examples2tex at line 51 column 5

QWTB: no uncertainty calculation
```

Compare estimated amplitudes

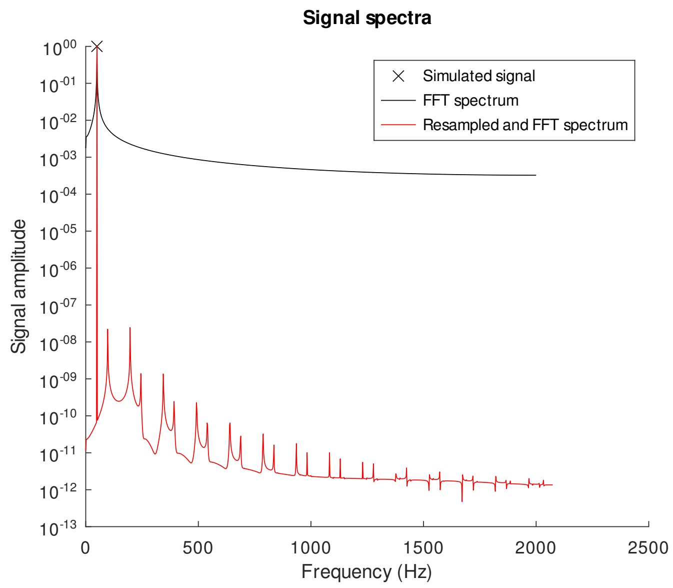
```
printf('Frequency and amplitude of main signal component.\n');
printf('Simulated:                      f = %.2f, A = %.5f\n',
   f, A)
id = find(SpectrumNonCoherent.A.v == max(SpectrumNonCoherent.A.
   v));
printf('As estimated\n')
printf('Non coherent sampling and FFT:  f = %.2f, A = %.5f\n',
   SpectrumNonCoherent.f.v(id), SpectrumNonCoherent.A.v(id));
id = find(SpectrumResampled.A.v == max(SpectrumResampled.A.v));
printf('Resampled to coherent and FFT: f = %.2f, A = %.5f\n',
   SpectrumResampled.f.v(id), SpectrumResampled.A.v(id));
```

```
Frequency and amplitude of main signal component.
Simulated:                      f = 49.77, A = 1.00000
As estimated
Non coherent sampling and FFT:  f = 50.00, A = 0.97996
Resampled to coherent and FFT: f = 49.77, A = 1.00000
```

Plot

```
hold on
semilogy(f, A, 'xk')
semilogy(SpectrumNonCoherent.f.v, abs(SpectrumNonCoherent.A.v),
    '-k')
semilogy(SpectrumResampled.f.v, abs(SpectrumResampled.A.v), '-r
   ')
hold off
xlabel('Frequency (Hz)')
ylabel('Signal amplitude')
```

```
legend('Simulated signal', 'FFT spectrum', 'Resampled and FFT
    spectrum')
title('Signal spectra')
```



ThreePSF – Standard Three Parameter Sine Wave Fit according IEEE Std 1241-2000

Description

Id: ThreePSF

Name: Standard Three Parameter Sine Wave Fit according IEEE Std 1241-2000

Description: Fits a sine wave to the recorded data using 3 parameter (amplitude, phase and offset) model. The algorithm is according IEEE Standard for Terminology and Test methods for Analog-to-Digital Converters 1241-2000. Algorithm requires exact value of signal frequency.

Citation: IEEE Std 1241-2000, Source code from: Rado Lapuh, "Sampling with 3458A, Understanding, Programming, Sampling and Signal Processing", ISBN 978-961-94476-0-4, 1st. ed., Ljubljana, Left Right d.o.o., 2018

Remarks: If sampling time |Ts| is not supplied, wrapper will calculate |Ts| from sampling frequency |fs| or if not supplied, mean of differences of time series |t| is used to calculate |Ts|.

License: UNKNOWN

Provides GUF: no

Provides MCM: no

Input Quantities

Required: Ts or fs or t, f

Descriptions:

`Ts` – Sampling time
`f` – Signal frequency
`fs` – Sampling frequency
`t` – Time series

Output Quantities:

`A` – Amplitude of main signal component
`O` – Offset of signal
`ph` – Phase of main signal component

Example

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

Call algorithm

Use QWTB to apply algorithm `ThreePSF` to data `DI`.

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

```
QWTB: no uncertainty calculation
```

Display results

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

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

```
A = 2.0000
ph = 1.0000
O = 0.2000
```

Errors of estimation in parts per milion:

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

```
Aerrppm = 4.4409e-10
pherrppm = 2.2204e-09
Oerrppm = 1.9429e-09
```

WaveformGenerator – Waveform Generator

Description

Id: WaveformGenerator

Name:

Description: Complex waveform generator with harmonics, interharmonics, modulation abilities, reference output and reference values for phasor measurement units.

Citation: First version of the algorithm: U. Pogliano, J.-P. Braun, B. Voljc, and R. Lapuh, 'Software Platform for PMU Algorithm Testing', IEEE Transactions on Instrumentation and Measurement, vol. 62, no. 6, pp. 1400–1406, Jun. 2013, doi: 10.1109/TIM.2013.2239051.

Remarks: If sampling time |Ts| is not supplied, wrapper will calculate |Ts| from sampling frequency |fs| or if not supplied, mean of differences of time series |t| is used to calculate |Ts|. First frequency in |f| is considered as the main signal frequency. Modulation is not yet ready.

License: MIT License

Provides GUF: no

Provides MCM: no

Input Quantities

Required: Ts or fs or t, L, f, A, ph

Optional: l, Fs

Descriptions:

A – Amplitude of signal components
Fs – PMU frames per second
L – Number of samples
Ts – Sampling time
f – Frequency of signal components
fs – Sampling frequency
l – Number of samples before t=0
ph – Phase of signal components
t – Time series

Output Quantities:

t – Time stamps
y – Samples

Example

Contents

Set acquisition properties

First acquisition quantities are prepared: sampling frequency `fs` set to 10 kHz, and length of the record `L` set to 1000, representing 0.1 s long record.

```
DI = [];
DI.fs.v = 1e4;
DI.L.v = 1e3;
```

Set waveform properties

The actual waveform will consist of 3 signal components. Main signal component frequency will be 50 Hz, amplitude 1 V. Third harmonic amplitude will be set to 0.15 V. Last component will be interharmonic of frequency 79 Hz and amplitude of 0.1 V.

```
DI.f.v =  [50  150 757];
DI.A.v =  [ 1 0.15 0.05];
DI.ph.v = [ 0    0   0];
```

Call algorithm

Use QWTB to apply algorithm `WaveformGenerator` to data `DI`.

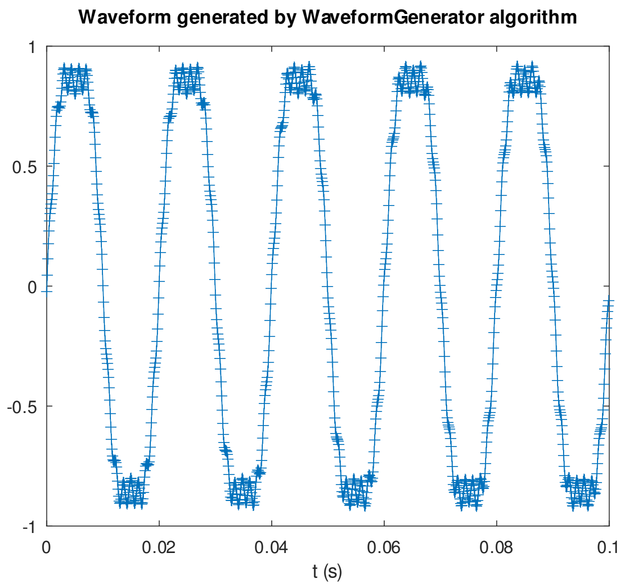
```
CS.verbose = 1;
DO = qwtb('WaveformGenerator', DI, CS);
```

```
QWTB: no uncertainty calculation
```

Display results

Results are the time stamps and samples.

```
plot(DO.t.v, DO.y.v, '-+')
xlabel('t (s)')
title('Waveform generated by WaveformGenerator algorithm')
```



wlsfit – Weighted Least Square Fitting Algortihm

Description

Id: wlsfit

Name: Weighted Least Square Fitting Algortihm

Description: Least Square fitting algortihm using ordinary (OLS) or weighted (WLS) fitting on a n-positive polynomial order. If uncertainty of |y| is defined, WLS is used. If value of |w| is defined, it is used instead of unc. of |y|. If no |w| nor unc. of |y|, OLS is used.

Citation:

Remarks: Implemented by Ricardo Iuzzolino, Estefania Luna, INTI

License: MIT License

Provides GUF: yes

Provides MCM: no

Input Quantities

Required: `x`, `y`, `n`

Optional: `w`

Parameters: `n`, `w`

Descriptions:

`n` – Degree of the polynomial for the regression (n>0)

`w` – Weigths - if defined, y.u is replaced by weights. If not defined, y.u is used as weghts.

`x` – Reference values

- `y` – Observed or measured values, y.u is used as weights for the case of WLS

Output Quantities:

- `coefs` – Fitted coefficients
- `exponents` – Exponents of polynomial used to fit
- `func` – Anonymous function constructed for exponents with parameters ‘x‘ and ‘coefs.v‘
- `model` – Model used for calculation.
- `yhat` – Fitted values y

Example

Contents

Set example data

Set independent and dependent variables for `wlsfit` algorithm, OLS method. Lets operate in semi logarithm space for easy plotting.

```
DIols = [];
DIols.x.v = log10([10 1e2 1e3 1e4 1e5]);
DIols.x.u = [];
DIols.y.v = [19.700 32.700 69.700 90.700 148.700];
% polynomial order:
DIols.n.v = 2;

% Create data for WLS method (add uncertainties of y):
DIwls = DIols;
DIwls.y.u = [4 10 13 20 33];
```

Call algorithm

Use QWTB to apply algorithm wlsfit to data DIwls.

```
DOols = qwtb('wlsfit', DIols);
DOwls = qwtb('wlsfit', DIwls);
```

```
QWTB: no uncertainty calculation
QWTB: wlsfit wrapper: No y uncertainties nor weights -> using
   OLS fitting
QWTB: no uncertainty calculation
QWTB: wlsfit wrapper: Using WLS fitting based on y
   uncertainties.
```

Display results

Results is

```
disp('')
disp([DOols.model.v ':'])
disp(['offset           : ' num2str(DOols.coefs.v(1)) ' +- '
   num2str(DOols.coefs.u(1))])
disp(['linear coeff.    : ' num2str(DOols.coefs.v(2)) ' +- '
   num2str(DOols.coefs.u(2))])
disp(['quadratic coeff.: ' num2str(DOols.coefs.v(3)) ' +- '
   num2str(DOols.coefs.u(3))])
disp([DOwls.model.v ':'])
disp(['offset           : ' num2str(DOwls.coefs.v(1)) ' +- '
   num2str(DOwls.coefs.u(1))])
disp(['linear coeff.    : ' num2str(DOwls.coefs.v(2)) ' +- '
   num2str(DOwls.coefs.u(2))])
disp(['quadratic coeff.: ' num2str(DOwls.coefs.v(3)) ' +- '
   num2str(DOwls.coefs.u(3))])
```

```

Ordinary Least Squares:
offset          : 14.5 +- 2.1448
linear coeff.   : -0.11429 +- 1.6345
quadratic coeff.: 5.2857 +- 0.26726
```

```
Weighted Least Squares, weights based on u(y):
offset          : 12.6828 +- 16.9884
linear coeff.   : 1.9434 +- 19.1198
quadratic coeff.: 4.9055 +- 3.9754
```

Interpolate values

Interpolate fitted polynom at values `t`.

```
t = [0:0.1:6];
tyols = DOols.func.v(t, DOols.coefs.v);
tywls = DOwls.func.v(t, DOwls.coefs.v);
```

Calculate uncertainties of interpolated values (`S` is sensitivity matrix, `CC` is covariance matrix of coefficients, `CT` is covariance matrix of interpolated values, `uty` is uncertainty of interpolated values).

```
for i = 1:length(t);
        S = t(i).^[0:DIols.n.v];
        CC = diag(DOols.coefs.u,0)*DOols.coefs.c*diag(DOols.
    coefs.u,0);
        CT(i)=S*CC*S';
end
utyols=CT.^0.5;

for i = 1:length(t);
        S = t(i).^[0:DIwls.n.v];
        CC = diag(DOwls.coefs.u,0)*DOwls.coefs.c*diag(DOwls.
    coefs.u,0);
        CT(i)=S*CC*S';
end
utywls=CT.^0.5;
```

Plot results

```
hold on
% input data:
errorbar(DIwls.x.v, DIwls.y.v, DIwls.y.u, 'xb')
```

```
% outputs:
plot(DIols.x.v, DOols.yhat.v, 'or')
errorbar(DIwls.x.v, DOwls.yhat.v, DOwls.yhat.u, 'og')
plot(t, tyols, '--r');
plot(t, tywls, '-g');
plot(t, tyols + utyols, '--r');
plot(t, tywls + utywls, '-g');
plot(t, tyols - utyols, '--r');
plot(t, tywls - utywls, '-g');
xlabel('log(f)')
ylabel('error of amplitude')
legend('original data','fitted values, OLS', 'fitted values,
    WLS','interpolated values, OLS', 'interpolated values, WLS'
    , 'uncer. of int. val., OLS', 'uncert. of int. val., WLS','
    location','southeast')
hold off
```