



QWTB documentation

Implemented algorithms

**QWTB version 0.2**

<https://qwtb.github.io/qwtb/>

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>ADEV – Allan Deviation</b>	<b>4</b>
<b>3</b>	<b>FourPSF – Standard Four Parameter Sine Wave Fit according IEEE Std 1241-2000</b>	<b>8</b>
<b>4</b>	<b>FPNLSF – Four Parameter Non-Linear Sine Fit</b>	<b>12</b>
<b>5</b>	<b>iDFT2p – 2-point interpolated DFT frequency estimator</b>	<b>17</b>
<b>6</b>	<b>iDFT3p – 3-point interpolated DFT frequency estimator</b>	<b>22</b>
<b>7</b>	<b>INL-DNL – Integral and Differential Non-Linearity of ADC</b>	<b>27</b>
<b>8</b>	<b>MADEV – Modified Allan Deviation</b>	<b>32</b>
<b>9</b>	<b>OADEV – Overlapping Allan Deviation</b>	<b>36</b>
<b>10</b>	<b>PSFE – Phase Sensitive Frequency Estimator</b>	<b>40</b>
<b>11</b>	<b>SFDR – Spurious Free Dynamic Range</b>	<b>44</b>
<b>12</b>	<b>SINAD-ENOB – Ratio of signal to noise and distortion and Effective number of bits (in time space)</b>	<b>47</b>
<b>13</b>	<b>SP-FFT – Spectrum by means of Fast Fourier Transform</b>	<b>52</b>
<b>14</b>	<b>SP-WFFT – Spectrum by means of Windowed Discrete Fourier Transform</b>	<b>56</b>

# 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 participating 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.wiley.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

### 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](mailto:mhopeng@gmail.com), Matlab Central.'

### Contents

- Generate sample data
- Call algorithm
- Display results

### 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.

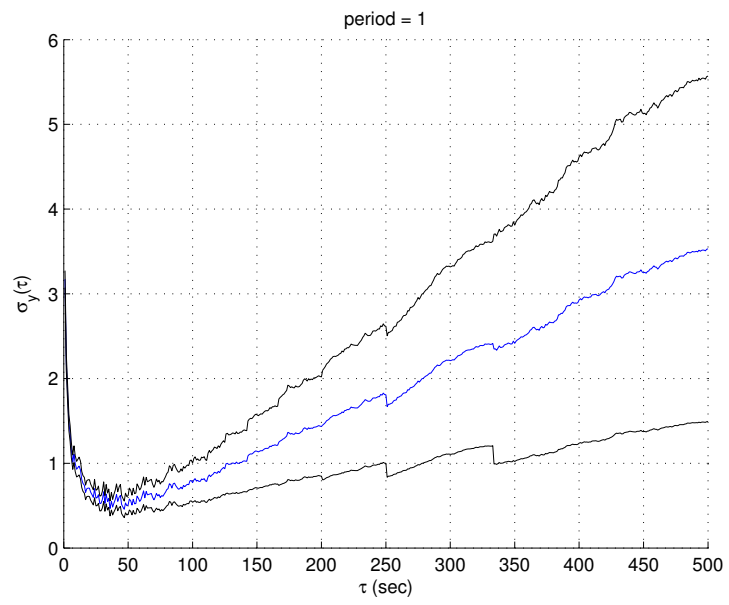
```
D0 = qwtb('ADEV', DI);
```

QWTB: no uncertainty calculation

### Display results

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

```
figure; hold on
loglog(D0.tau.v, D0.adev.v, '-b')
loglog(D0.tau.v, D0.adev.v + D0.adev.u, '-k')
loglog(D0.tau.v, D0.adev.v - D0.adev.u, '-k')
xlabel('\tau (sec)');
ylabel('\sigma_y(\tau)');
title(['period = ' num2str(1/DI.fs.v)]);
grid('on'); hold off
```



# 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  $|T_s|$  is not supplied, wrapper will calculate  $|T_s|$  from sampling frequency  $|f_s|$  or if not supplied, mean of differences of time series  $|t|$  is used to calculate  $|T_s|$ .

**License:** UNKNOWN

**Provides GUF:** no

**Provides MCM:** no

**Input Quantities**

**Required:**  $T_s$  or  $f_s$  or  $t$ ,  $y$



**Descriptions:**

$T_s$  – Sampling time  
 $f_s$  – 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

**Four parameter sine wave fitting**

Example for algorithm FourPSF.

FourPSF is an algorithm for estimating the frequency, amplitude, phase and offset of the sine waveform according standard IEEE Std 1241-2000’;

**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;
```

**Call algorithm**

Use QWTB to apply algorithm FourPSF to data DI.

```
CS.verbose = 1;  
D0 = 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 = D0.A.v  
f = D0.f.v  
ph = D0.ph.v  
O = D0.O.v
```

```
A =  
  
    2.0000  
  
f =  
  
    100  
  
ph =  
  
    1.0000  
  
O =
```

```
0.2000
```

Errors of estimation in parts per milion:

```
Aerrppm = (D0.A.v - Anom)/Anom .* 1e6  
ferrppm = (D0.f.v - fnom)/fnom .* 1e6  
pherrppm = (D0.ph.v - phnom)/phnom .* 1e6  
Oerrppm = (D0.O.v - Onom)/Onom .* 1e6
```

```
Aerrppm =  
2.2204e-09
```

```
ferrppm =  
0
```

```
pherrppm =  
8.8818e-09
```

```
Oerrppm =  
1.3878e-09
```

# 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  $Ts$ ,  $y$ ,  $f_{est}$

**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

**Four parameter sine wave fitting**

Example for algorithm FPNLSF.

FPNLSF 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 FPNLSF to data DI.

```
CS.verbose = 1;  
D0 = qwtb('FPNLSF', 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 = D0.A.v  
f = D0.f.v  
ph = D0.ph.v  
O = D0.O.v
```

```
A =
```

```

2.0000

f =

100

ph =

1.0000

O =

0.2000

```

Errors of estimation in parts per milion:

```

Aerrppm = (D0.A.v - Anom)/Anom .* 1e6
ferrppm = (D0.f.v - fnom)/fnom .* 1e6
pherrppm = (D0.ph.v - phnom)/phnom .* 1e6
Oerrppm = (D0.O.v - Onom)/Onom .* 1e6

```

```

Aerrppm =

4.8894e-07

ferrppm =

0

pherrppm =

4.8850e-09

```

```
0errppm =
```

```
-1.1102e-08
```



# 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> Implemented by Rado Lapuh, 2016.

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

### 2-point interpolated DFT frequency estimator

Example for algorithm iDFT3p.

iDFT2p is an algorithm for estimating the frequency, amplitude, and phase of the fundamental component using interpolated discrete Fourier transform. Rectangular or Hann window can be used for DFT.’; See also 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> Implemented by Rado Lapuh, 2016.’;

### Contents

- Generate sample data
- Call algorithm
- Display results

### 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 = (D0r.f.v - fnom)./fnom .* 1e6  
A_re = (D0r.A.v - Anom)./Anom .* 1e6  
ph_re = (D0r.ph.v - phnom)./phnom .* 1e6  
O_re = (D0r.O.v - Onom)./Onom .* 1e6
```

```
f_re =  
  
    -0.8083  
  
A_re =  
  
    40.3148  
  
ph_re =  
  
    217.9412  
  
O_re =  
  
    1.6826e+03
```

For Hann window:

```
f_he = (D0h.f.v - fnom)./fnom .* 1e6  
A_he = (D0h.A.v - Anom)./Anom .* 1e6  
ph_he = (D0h.ph.v - phnom)./phnom .* 1e6  
O_he = (D0h.O.v - Onom)./Onom .* 1e6
```

```
f_he =  
  
    1.8426e-04
```

```
A_he =  
    -0.0046  
  
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> Implemented by Rado Lapuh, 2016.

**Remarks:** If sampling time  $|T_s|$  is not supplied, wrapper will calculate  $|T_s|$  from sampling frequency  $|f_s|$  or if not supplied, mean of differences of time series  $|t|$  is used to calculate  $|T_s|$ . 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:**  $T_s$  or  $f_s$  or  $t$ ,  $y$

**Optional:** window

**Parameters:** window

**Descriptions:**

$T_s$  – Sampling time

$f_s$  – 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

### 3-point interpolated DFT frequency estimator

Example for algorithm *iDFT3p*.

*iDFT3p* is 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.’; See also 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> Implemented by Rado Lapuh, 2016.’;

### Contents

- Generate sample data
- Call algorithm
- Display results

### 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 = (D0r.f.v - fnom)./fnom .* 1e6  
A_re = (D0r.A.v - Anom)./Anom .* 1e6  
ph_re = (D0r.ph.v - phnom)./phnom .* 1e6  
O_re = (D0r.O.v - Onom)./Onom .* 1e6
```

```
f_re =  
  
    0.0166  
  
A_re =  
  
    41.2567  
  
ph_re =  
  
    88.3681  
  
O_re =  
  
    1.6826e+03
```

For Hann window:

```
f_he = (D0h.f.v - fnom)./fnom .* 1e6  
A_he = (D0h.A.v - Anom)./Anom .* 1e6  
ph_he = (D0h.ph.v - phnom)./phnom .* 1e6  
O_he = (D0h.O.v - Onom)./Onom .* 1e6
```

```
f_he =  
  
   -3.7790e-06
```

```
A_he =  
    3.9679e-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 Hungarian), 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

### 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.

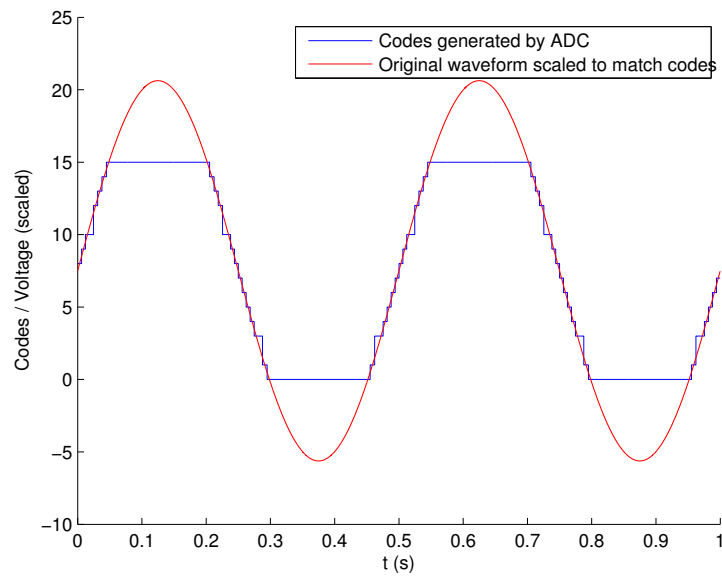
```
codes = wvfrm;
rmin = -2; rmax = 2;
levels = 2.^DI.bitres.v - 1;
codes(codes<rmin) = rmin;
codes(codes>rmax) = rmax;
codes = round((codes-rmin)./(rmax-rmin).*levels);
```

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

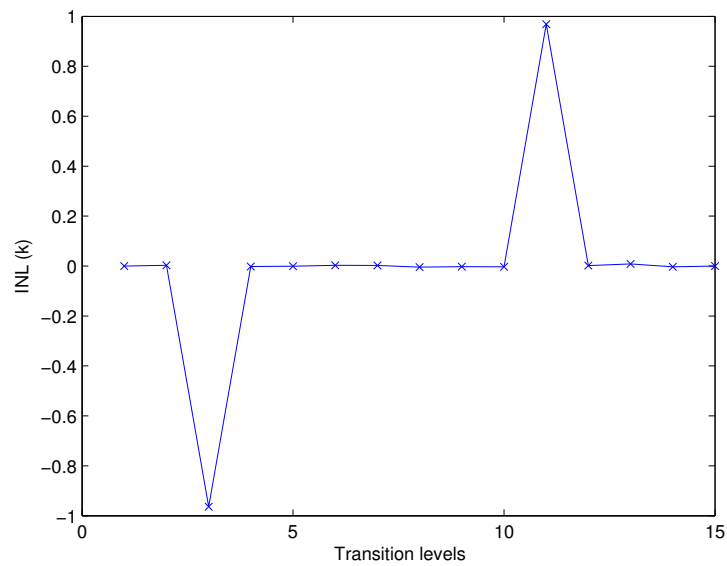
Apply INL algorithm to the input data DI.

```
D0 = qwtb('INL-DNL', DI);
```

QWTB: no uncertainty calculation

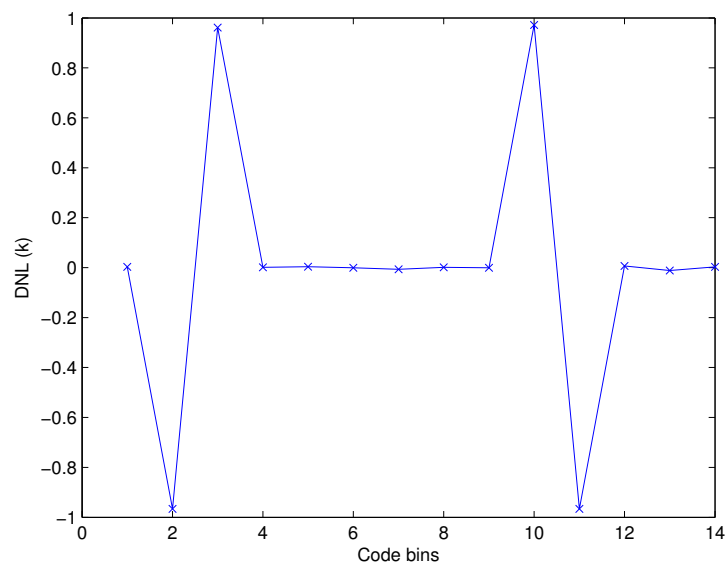
Plot results of integral non-linearity. One can clearly observe defects on codes 3 and 11.

```
figure
plot(D0.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)')
```



# 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.wiley.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

### 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](mailto:mhopeng@gmail.com), Matlab Central.'

### Contents

- Generate sample data
- Call algorithm
- Display results

### 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.

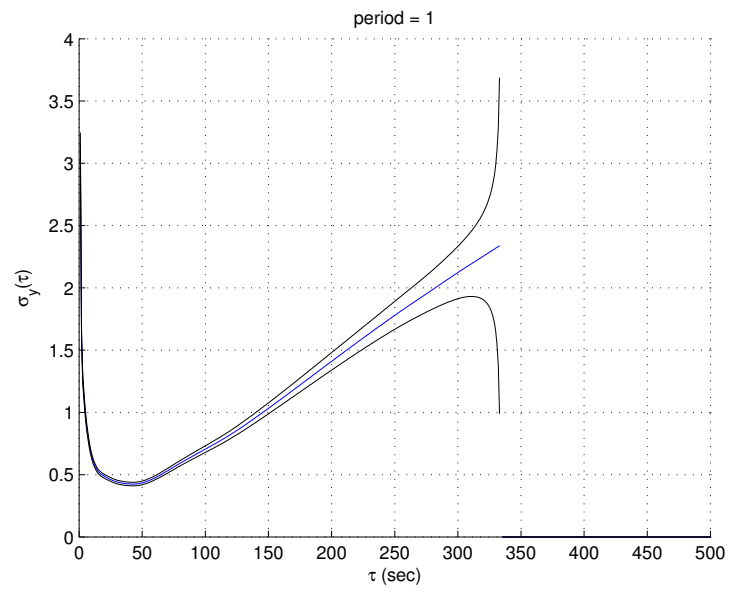
```
D0 = qwtb('MADEV', DI);
```

```
QWTB: no uncertainty calculation
```

### Display results

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

```
figure; hold on
loglog(D0.tau.v, D0.madev.v, '-b')
loglog(D0.tau.v, D0.madev.v + D0.madev.u, '-k')
loglog(D0.tau.v, D0.madev.v - D0.madev.u, '-k')
xlabel('\tau (sec)');
ylabel('\sigma_y(\tau)');
title(['period = ' num2str(DI.fs.v)]);
grid('on'); hold off
```



# 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, mhopen@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.wiley.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

### 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](mailto:mhopeng@gmail.com), Matlab Central.'

### Contents

- Generate sample data
- Call algorithm
- Display results

### 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.

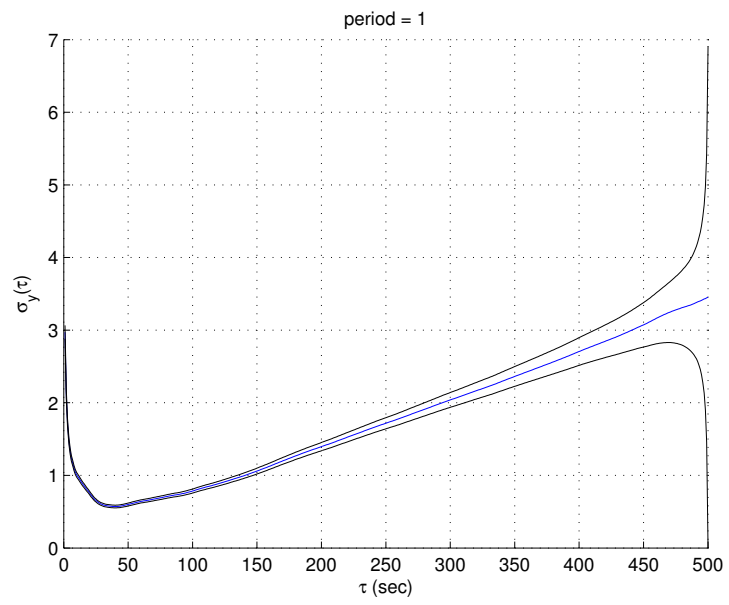
```
D0 = qwtb('OADEV', DI);
```

QWTB: no uncertainty calculation

### Display results

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

```
figure; hold on
loglog(D0.tau.v, D0.oadev.v, '-b')
loglog(D0.tau.v, D0.oadev.v + D0.oadev.u, '-k')
loglog(D0.tau.v, D0.oadev.v - D0.oadev.u, '-k')
xlabel('\tau (sec)');
ylabel('\sigma_y(\tau)');
title(['period = ' num2str(DI.fs.v)]);
grid('on'); hold off
```



# 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.

**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:** If sampling time  $|T_s|$  is not supplied, wrapper will calculate  $|T_s|$  from sampling frequency  $|f_s|$  or if not supplied, mean of differences of time series  $|t|$  is used to calculate  $|T_s|$ .

**License:** MIT License

**Provides GUF:** no

**Provides MCM:** no



**Input Quantities**

**Required:**  $T_s$  or  $f_s$  or  $t$ ,  $y$

**Descriptions:**

$T_s$  – Sampling time  
 $f_s$  – Sampling frequency  
 $t$  – Time series  
 $y$  – Sampled values

**Output Quantities:**

$A$  – Amplitude of main signal component  
 $f$  – Frequency of main signal component  
 $ph$  – Phase of main signal component

---

## 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
- Call algorithm

- Display results

### Generate sample data

Two quantities are prepared:  $T_s$  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.

```
D0 = qwtb('PSFE', DI);
```

QWTB: no uncertainty calculation

### Display results

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

```
f = D0.f.v  
A = D0.A.v  
ph = D0.ph.v
```

```
f =  
  
    100.0000  
  
A =  
  
    1.0000  
  
ph =  
  
    1.0000
```

Errors of estimation in parts per milion:

```
ferrppm = (D0.f.v - fnom)/fnom .* 1e6  
Aerrppm = (D0.A.v - Anom)/Anom .* 1e6  
pherrppm = (D0.ph.v - phnom)/phnom .* 1e6
```

```
ferrppm =  
  
    0.0082  
  
Aerrppm =  
  
   -4.5238  
  
pherrppm =  
  
   12.4392
```

# SFDR – Spurious Free Dynamic Range

---

## Description

**Id:** SFDR

**Name:** Spurious Free Dynamic Range

**Description:** 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

**Provides GUF:** no

**Provides MCM:** no

**Input Quantities**

**Required:**  $y$

**Descriptions:**

$y$  – Sampled values

**Output Quantities:**

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

---

## 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
- Call algorithm
- Display results

#### 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.

```
D0 = qwtb('SFDR', DI);
```

```
QWTB: no uncertainty calculation
```

**Display results**

Result is the SFDR (dBc).

```
SFDR = D0.SFDRdBc.v
```

```
SFDR =
```

```
40.0000
```

# 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  $f_s$  or  $T_s$ ,  $y$ ,  $f$ ,  $A$ ,  $\phi$ ,  $O$ , bitres, FSR

**Descriptions:**

$A$  – Amplitude of main signal component  
 $FSR$  – Full scale range of an ADC  
 $O$  – Offset of signal  
 $T_s$  – Sampling time  
bitres – Bit resolution of an ADC  
 $f$  – Frequency of main signal component  
 $f_s$  – Sampling frequency  
 $\phi$  – 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

### 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
- Calculate estimates of signal parameters
- Copy results to inputs
- Calculate SINAD and ENOB
- 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;
```

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;
D0 = qwtb('FPNLSF', 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 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 =  
  
82.9229  
  
ENOB =  
  
13.0657
```

Theoretical value of SINADdB is  $20 \cdot \log_{10}(\text{Anom.}/(\text{noisestd.} \cdot \sqrt{2}))$ . Theoretical value of ENOB is  $\log_2(\text{DI.range.v.}/(\text{noisestd.} \cdot \sqrt{12}))$ . 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 =  
  
-0.0874  
  
ENOBerror =  
  
-0.0145
```

# SP-FFT – Spectrum by means of Fast Fourier Transform

---

## Description

**Id:** SP-FFT

**Name:** Spectrum by means of Fast Fourier Transform

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

**Citation:**

**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  $|fs|$ .

**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:

A – Amplitude spectrum  
 f – Frequency series  
 ph – Phase spectrum

---

## 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
- Call algorithm
- Display results

#### 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.

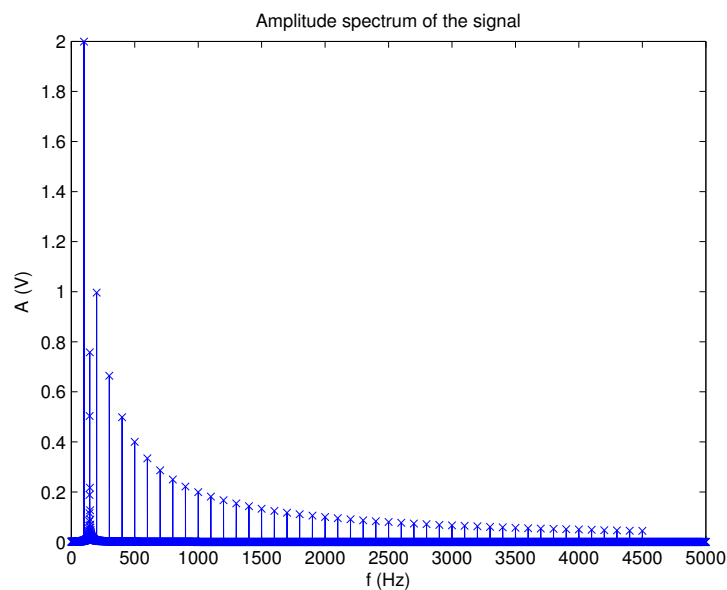
```
D0 = qwtb('SP-FFT', DI);
```

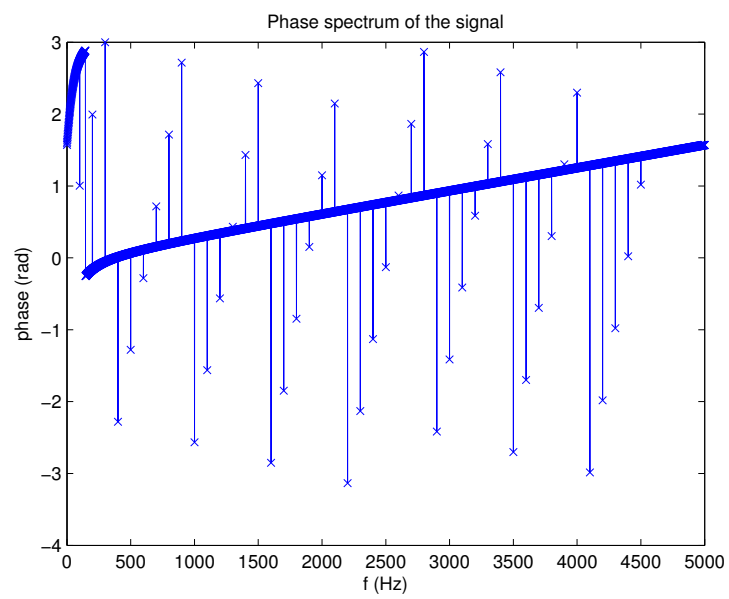
QWTB: no uncertainty calculation

**Display results**

Results is the amplitude and phase spectrum.

```
figure
plot(D0.f.v, D0.A.v, '-x')
xlabel('f (Hz)'); ylabel('A (V)'); title('Amplitude spectrum of
the signal');
figure
plot(D0.f.v, D0.ph.v, '-x')
xlabel('f (Hz)'); ylabel('phase (rad)'); title('Phase spectrum
of the signal');
```





# 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. Following windows are implemented: barthann, bartlett, blackman, blackman-harris, 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. Schaffer, 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 Novaková 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
- f – Frequency series
- ph – Phase spectrum

---

## Example

### Windowed Discrete Fourier Transform

Example for algorithm SP-WFFT.

Windowed Discrete Fourier Transform is method to calculate frequency and phase spectrum. Example shows the effect of coherent sampling, non-coherent sampling, windowing and zero padding on estimating amplitude and phase by means of FFT. Last part compares all windows in frequency space.

### Contents

- Generate sample data
- Call algorithm
- Display results
- Compare window coefficients

### 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 D0. Window function is not specified, therefore rectangle (none) window will be used.

```

D0 = qwtb('SP-WFFT', DI);
%
% Set window function to blackman and calculate windowed
% amplitude and phase spectrum and store results into |D0w|.
DI.window.v = 'blackman';
D0w = qwtb('SP-WFFT', DI);
%
% Set zero padding to 10 times the signal length and calculate
% zero padded windowed amplitude and
% phase spectrum and store results into |D0wz|.
DI.fft_padding.v = 10.*length(DI.y.v);
D0wz = qwtb('SP-WFFT', DI);
%
```

```

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

### Display results

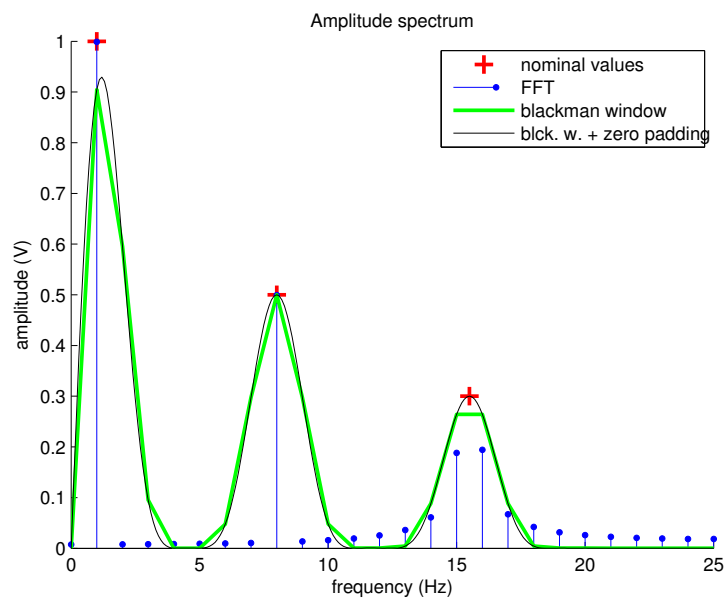
Plot amplitude spectrum.

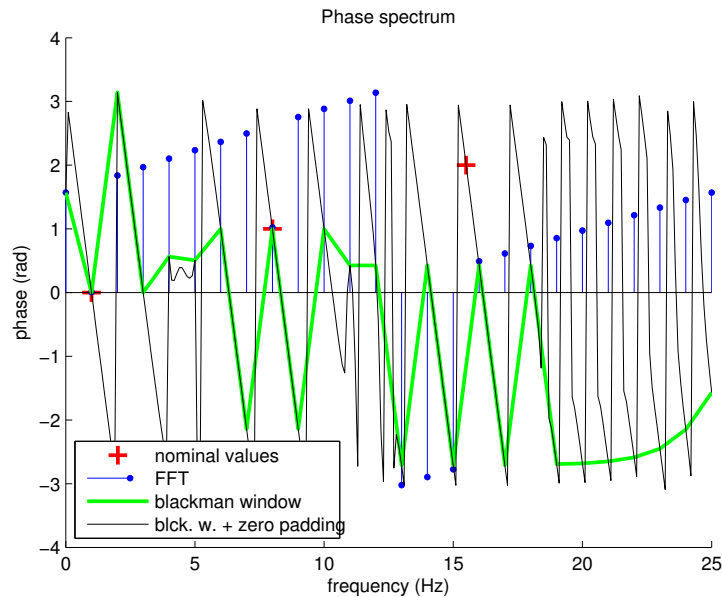
```

figure; hold on;
plot(fnom, Anom, '+r', 'markersize', 10, 'linewidth', 2);
stem(D0.f.v, D0.A.v, '-ob', 'filled', 'markersize', 3);
plot(D0w.f.v, D0w.A.v, '-g', 'linewidth', 2);
plot(D0wz.f.v, D0wz.A.v, '-k');
```

```

legend('nominal values','FFT', 'blackman window', 'blk. 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', 'blk. w. +
      zero padding', 'location', 'southwest')
title('Phase spectrum');
xlabel('frequency (Hz)'); ylabel('phase (rad)');
hold off
%
```





### Compare window coefficients

Different windows have 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};
    D0 = qwtb('SP-WFFT', DI);
    plot(D0.A.v, '-', 'color', col(i,:));
```

[illegible]

