# Contents

1 AllanTools
   1.1 Installation .................................................. 1
   1.2 Basic usage .................................................. 2
   1.3 Jupyter notebooks with examples .............................. 3
   1.4 Authors ....................................................... 4

2 Implemented statistics functions ................................. 5
   2.1 Low-level access to the statistics functions .................. 5
   2.2 Statistics .................................................... 5
   2.3 Real-Time Statistics ........................................... 11
   2.4 Noise Generation ............................................ 12
   2.5 Utilities ..................................................... 13

3 API ........................................................................ 17
   3.1 The Dataset() class ............................................. 17
   3.2 The Plot() class ............................................... 18

4 Development ................................................................ 21
   4.1 To-do list ......................................................... 21
   4.2 Documentation generation ...................................... 21
   4.3 Tests ............................................................... 22
   4.4 Notes for Pypi .................................................... 22

5 References .................................................................. 25
   5.1 Code ............................................................... 25
   5.2 Papers ............................................................. 25

Bibliography ................................................................ 27

Index ......................................................................... 29
A python library for calculating Allan deviation and related time & frequency statistics. LGPL v3+ license.

- Development at https://github.com/aewallin/allantools
- Installation package at https://pypi.python.org/pypi/AllanTools
- Discussion group at https://groups.google.com/d/forum/allantools
- Documentation available at https://allantools.readthedocs.org

Input data should be evenly spaced observations of either fractional frequency, or phase in seconds. Deviations are calculated for given tau values in seconds.
**Function** | **Description**
--- | ---
`adev()` | Allan deviation
`oadev()` | Overlapping Allan deviation
`mdev()` | Modified Allan deviation
`tdev()` | Time deviation
`hdev()` | Hadamard deviation
`ohdev()` | Overlapping Hadamard deviation
`totdev()` | Total deviation
`mtotdev()` | Modified total deviation
`ttotdev()` | Time total deviation
`htotdev()` | Hadamard total deviation
`theo1()` | Theo1 deviation
`mtie()` | Maximum Time Interval Error
`tierms()` | Time Interval Error RMS
`gradev()` | Gap resistant overlapping Allan deviation

Noise generators for creating synthetic datasets are also included:

- violet noise with $f^2$ PSD
- white noise with $f^0$ PSD
- pink noise with $f^{-1}$ PSD
- Brownian or random walk noise with $f^{-2}$ PSD

More details on available statistics and noise generators: full list of available functions

see /tests for tests that compare allantools output to other (e.g. Stable32) programs. More test data, benchmarks, ipython notebooks, and comparisons to known-good algorithms are welcome!

### 1.1 Installation

Install from pypi:

```
pip install allantools
```

Latest version + examples, tests, test data, iPython notebooks: clone from github, then install

```
python setup.py install
```

(see `python setup.py --help install` for install options)

These commands should be run as root for system-wide installation, or you can use the `--user` option to install for your account only. Exact command names may vary depending on your OS / package manager / target python version.

### 1.2 Basic usage

#### 1.2.1 Minimal example, phase data

We can call allantools with only one parameter - an array of phase data. This is suitable for time-interval measurements at 1 Hz, for example from a time-interval-counter measuring the 1PPS output of two clocks.
when only one input parameter is given, phase data in seconds is assumed when no rate parameter is given, rate=1.0 is the default when no taus parameter is given, taus='octave' is the default

1.2.2 Frequency data example

Note that allantools assumes non-dimensional frequency data input. Normalization, by e.g. dividing all data points with the average frequency, is left to the user.

New in 2016.11: simple top-level API, using dedicated classes for data handling and plotting.

1.3 Jupyter notebooks with examples

Jupyter notebooks are interactive python scripts, embedded in a browser, allowing you to manipulate data and display plots like easily. For guidance on installing jupyter, please refer to https://jupyter.org/install.

See /examples for some examples in notebook format.

github formats the notebooks into nice web-pages, for example

1.4 Authors

- Anders E.E. Wallin, anders.e.e.wallin "at" gmail.com, https://github.com/aewallin
- Danny Price, https://github.com/telegraphic
- Cantwell G. Carson, carsone "at" gmail.com
- Frédéric Meynadier, https://github.com/fmeynadier
- Yan Xie, https://github.com/yxie-git
- Erik Benkler, https://github.com/EBenkler
2.1 Low-level access to the statistics functions

The deviation functions are generally of the form:

\[
(tau\_out, adev, adeverr, n) = \text{allantools.adev}(data, rate=1.0, data\_type='phase', \text{taus=None})
\]

**Inputs:**
- `data` = list of phase measurements in seconds, or list of fractional frequency measurements (nondimensional)
- `rate` = sample rate of data in Hz, i.e. interval between phase measurements is 1/rate seconds.
- `data_type` = either “phase” or “freq”
- `taus` = list of tau-values for ADEV computation. The keywords “all”, “octave”, or “decade” can also be used.

**Outputs**
- `tau_out` = list of tau-values for which deviations were computed
- `adev` = list of ADEV (or another statistic) deviations
- `adeverr` = list of estimated errors of allan deviations. some functions instead return a confidence interval (err_l, err_h)
- `n` = list of number of pairs in allan computation. standard error is adeverr = adev/sqrt(n)

2.2 Statistics

`allantools.adev(data, rate=1.0, data_type='phase', taus=None)`

**Allan deviation.** Classic - use only if required - relatively poor confidence.
\[
\sigma^2_{\text{ADEV}}(\tau) = \frac{1}{2\tau^2}((x_{n+2} - 2x_{n+1} + x_n)^2) = \frac{1}{2(N-2)\tau^2} \sum_{n=1}^{N-2} (x_{n+2} - 2x_{n+1} + x_n)^2
\]

where \(x_n\) is the time-series of phase observations, spaced by the measurement interval \(\tau\), and with length \(N\). Or alternatively calculated from a time-series of fractional frequency:

\[
\sigma^2_{\text{ADEV}}(\tau) = \frac{1}{2}(\bar{y}_{n+1} - \bar{y}_n)^2
\]

where \(\bar{y}_n\) is the time-series of fractional frequency at averaging time \(\tau\)

NIST [SP1065] eqn (6) and (7), pages 14 and 15.

**Parameters**

- `data (np.array)` – Input data. Provide either phase or frequency (fractional, adimensional).
- `rate (float)` – The sampling rate for data, in Hz. Defaults to 1.0
- `data_type (['phase', 'freq'])` – Data type, i.e. phase or frequency. Defaults to “phase”.
- `taus (np.array)` – Array of tau values, in seconds, for which to compute statistic. Optionally set taus=[“all”|”octave”|”decade”] for automatic tau-list generation.

**Returns**

- `(taus2, ad, ade, ns)` (tuple) – Tuple of values

\[\sigma^2_{O\text{ADEV}}(m\tau_0) = \frac{1}{2(m\tau_0)^2(N-2m)} \sum_{n=1}^{N-2m} (x_{n+2m} - 2x_{n+1m} + x_n)^2\]

where \(\sigma^2_{O\text{ADEV}}(m\tau_0)\) is the overlapping Allan deviation at an averaging time of \(\tau = m\tau_0\), and \(x_n\) is the time-series of phase observations, spaced by the measurement interval \(\tau_0\), with length \(N\).

NIST [SP1065] eqn (11), page 16.

**Parameters**

- `data (np.array)` – Input data. Provide either phase or frequency (fractional, adimensional).
- `rate (float)` – The sampling rate for data, in Hz. Defaults to 1.0
- `data_type (['phase', 'freq'])` – Data type, i.e. phase or frequency. Defaults to “phase”.
- `taus (np.array)` – Array of tau values, in seconds, for which to compute statistic. Optionally set taus=[“all”|”octave”|”decade”] for automatic tau-list generation.

**Returns**

- `(taus2, ad, ade, ns)` (tuple) – Tuple of values
• \texttt{taus2 (np.array)} – Tau values for which \texttt{td} computed
• \texttt{ad (np.array)} – Computed oadev for each tau value
• \texttt{ade (np.array)} – oadev errors
• \texttt{ns (np.array)} – Values of N used in each oadev calculation

\texttt{allantools.mdev (data, rate=1.0, data_type='phase', taus=None)}

Modified Allan deviation. Used to distinguish between White and Flicker Phase Modulation.

\[
\sigma_{MDEV}^2(m\tau_0) = \frac{1}{2(m\tau_0)^2(N - 3m + 1)} \sum_{j=1}^{N-3m+1} \left\{ \sum_{i=j}^{j+m-1} x_{i+2m} - 2x_{i+m} + x_i \right\}^2
\]

see http://www.leapsecond.com/tools/adev_lib.c

NIST [SP1065] eqn (14), page 17.

Parameters
• \texttt{data (np.array)} – Input data. Provide either phase or frequency (fractional, adimensional).
• \texttt{rate (float)} – The sampling rate for data, in Hz. Defaults to 1.0
• \texttt{data_type ({'phase', 'freq'})} – Data type, i.e. phase or frequency. Defaults to “phase”.
• \texttt{taus (np.array)} – Array of tau values, in seconds, for which to compute statistic. Optionally set \texttt{taus=['all'|'octave'|'decade']} for automatic tau-list generation.

Returns
• \texttt{(taus2, md, mde, ns) (tuple)} – Tuple of values
• \texttt{taus2 (np.array)} – Tau values for which \texttt{md} computed
• \texttt{md (np.array)} – Computed mdev for each tau value
• \texttt{mde (np.array)} – mdev errors
• \texttt{ns (np.array)} – Values of N used in each mdev calculation

\texttt{allantools.hdev (data, rate=1.0, data_type='phase', taus=None)}

Hadamard deviation. Rejects frequency drift, and handles divergent noise.

\[
\sigma_{HDEV}^2(\tau) = \frac{1}{6\tau^2(N - 3)} \sum_{i=1}^{N-3} (x_{i+3} - 3x_{i+2} + 3x_{i+1} - x_i)^2
\]

where \( x_i \) is the time-series of phase observations, spaced by the measurement interval \( \tau \), and with length \( N \).

NIST [SP1065] eqn (17) and (18), page 20

Parameters
• \texttt{data (np.array)} – Input data. Provide either phase or frequency (fractional, adimensional).
• \texttt{rate (float)} – The sampling rate for data, in Hz. Defaults to 1.0
• \texttt{data_type ({'phase', 'freq'})} – Data type, i.e. phase or frequency. Defaults to “phase”.
• \texttt{taus (np.array)} – Array of tau values, in seconds, for which to compute statistic. Optionally set \texttt{taus=['all'|'octave'|'decade']} for automatic tau-list generation.
allantools.ohdev (data, rate=1.0, data_type='phase', taus=None)

**Overlapping Hadamard deviation.** Better confidence than normal Hadamard.

\[
\sigma_{OHDEV}^2(m\tau_0) = \frac{1}{6(m\tau_0)^2(N-3m)} \sum_{i=1}^{N-3m} (x_{i+3m} - 3x_{i+2m} + 3x_{i+m} - x_i)^2
\]

where \(x_i\) is the time-series of phase observations, spaced by the measurement interval \(\tau_0\), and with length \(N\).

**Parameters**

- **data** (np.array) – Input data. Provide either phase or frequency (fractional, adimensional).
- **rate** (float) – The sampling rate for data, in Hz. Defaults to 1.0
- **data_type** ({'phase', 'freq'}) – Data type, i.e. phase or frequency. Defaults to “phase”.
- **taus** (np.array) – Array of tau values, in seconds, for which to compute statistic. Optionally set taus=[“all”|“octave”|“decade”] for automatic tau-list generation.

**Returns**

- (taus2, hd, hde, ns) (tuple) – Tuple of values
- **taus2** (np.array) – Tau values for which td computed
- **hd** (np.array) – Computed hdev for each tau value
- **hde** (np.array) – hdev errors
- **ns** (np.array) – Values of \(N\) used in each hdev calculation

allantools.tdev (data, rate=1.0, data_type='phase', taus=None)

**Time deviation.** Based on modified Allan variance.

\[
\sigma_{TDEV}^2(\tau) = \frac{\tau^2}{3} \sigma_{MDEV}^2(\tau)
\]

Note that TDEV has a unit of seconds.

NIST [SP1065] eqn (15), page 18.

**Parameters**

- **data** (np.array) – Input data. Provide either phase or frequency (fractional, adimensional).
- **rate** (float) – The sampling rate for data, in Hz. Defaults to 1.0
- **data_type** ({'phase', 'freq'}) – Data type, i.e. phase or frequency. Defaults to “phase”.
- **taus** (np.array) – Array of tau values, in seconds, for which to compute statistic. Optionally set taus=[“all”|“octave”|“decade”] for automatic tau-list generation.

**Returns**

- (taus, tdev, tdev_error, ns) (tuple) – Tuple of values
- **taus** (np.array) – Tau values for which td computed
- **tdev** (np.array) – Computed time deviations (in seconds) for each tau value
- **tdev_errors** (np.array) – Time deviation errors
- **ns** (np.array) – Values of \(N\) used in mdev_phase()
Notes

http://en.wikipedia.org/wiki/Time_deviation

**allantools.totdev** *(data, rate=1.0, data_type='phase', taus=None)*

**Total deviation.** Better confidence at long averages for Allan deviation.

\[
\sigma_{\text{TOTDEV}}^2(m\tau_0) = \frac{1}{2(m\tau_0)^2(N-2)} \sum_{i=2}^{N-1} (x_{i-m}^* - 2x_i^* + x_{i+m}^*)^2
\]

Where \(x_i^*\) is a new time-series of length \(3N - 4\) derived from the original phase time-series \(x_n\) of length \(N\) by reflection at both ends.

**FIXME:** better description of reflection operation. the original data \(x\) is in the center of \(x^*\): \(x^*(1-j) = 2x(1) - x(1+j)\) for \(j=1..N-2\) \(x^*(i) = x(i)\) for \(i=1..N\) \(x^*(N+j) = 2x(N) - x(N-j)\) for \(j=1..N-2\) \(x^*\) has length \(3N-4\) \(\tau = m^*\tau_0\)

NIST [SP1065] eqn (25) page 23

**FIXME:** bias correction http://www.wriley.com/CI2.pdf page 5

**Parameters**

- **phase** *(np.array)* – Phase data in seconds. Provide either phase or frequency.
- **frequency** *(np.array)* – Fractional frequency data (nondimensional). Provide either frequency or phase.
- **rate** *(float)* – The sampling rate for phase or frequency, in Hz
- **taus** *(np.array)* – Array of tau values for which to compute measurement

**allantools.mtotdev** *(data, rate=1.0, data_type='phase', taus=None)*

**PRELIMINARY - REQUIRES FURTHER TESTING.** Modified Total deviation. Better confidence at long averages for modified Allan

**FIXME:** bias-correction http://www.wriley.com/CI2.pdf page 6

The variance is scaled up (divided by this number) based on the noise-type identified. WPM 0.94 FPM 0.83 WFM 0.73 FFM 0.70 RWFM 0.69

**Parameters**

- **data** *(np.array)* – Input data. Provide either phase or frequency (fractional, adimensional).
- **rate** *(float)* – The sampling rate for data, in Hz. Defaults to 1.0
- **data_type** *({'phase', 'freq'})* – Data type, i.e. phase or frequency. Defaults to “phase”.
- **taus** *(np.array)* – Array of tau values, in seconds, for which to compute statistic. Optionally set taus=[“all””octave””decade”] for automatic tau-list generation.
- **SP1065 eqn (27) page 25 (NIST) –

**allantools.ttotdev** *(data, rate=1.0, data_type='phase', taus=None)*

**Time Total Deviation**

Modified total variance scaled by \(\tau^2 / 3\)

NIST [SP1065] eqn (28) page 26. Note that [SP1065] erroneously has tau-cubed here (!).

**allantools.htotdev** *(data, rate=1.0, data_type='phase', taus=None)*

2.2. Statistics
PRELIMINARY - REQUIRES FURTHER TESTING. Hadamard Total deviation. Better confidence at long averages for Hadamard deviation

Computed for N fractional frequency points y_i with sampling period tau0, analyzed at tau = m*tau0. 1. remove linear trend by averaging first and last half and divide by interval. 2. extend sequence by uninverted even reflection. 3. compute Hadamard for extended, length 9m, sequence.

FIXME: bias corrections from http://www.wiley.com/C12.pdf W FM 0.995 alpha= 0 F FM 0.851 alpha=-1 RW FM 0.771 alpha=-2 FW FM 0.717 alpha=-3 RR FM 0.679 alpha=-4

Parameters

• **data** (np.array) – Input data. Provide either phase or frequency (fractional, adimensional).

• **rate** (float) – The sampling rate for data, in Hz. Defaults to 1.0

• **data_type** ("phase", "freq") – Data type, i.e. phase or frequency. Defaults to “phase”.

• **taus** (np.array) – Array of tau values, in seconds, for which to compute statistic. Optionally set taus=[“all”"l""octave”"l""decade”] for automatic tau-list generation.

allantools.theo1(data, rate=1.0, data_type='phase', taus=None)

PRELIMINARY - REQUIRES FURTHER TESTING. Theo1 is a two-sample variance with improved confidence and extended averaging factor range.

\[ \sigma^2_{\text{THEO1}}(m\tau_0) = \frac{1}{(m\tau_0)^2(N-m)} \sum_{i=1}^{N-m} \sum_{\delta=0}^{m/2-1} \frac{1}{m/2-\delta} \left\{ (x_i - x_{i-\delta+m/2}) + (x_{i+m} - x_{i+\delta+m/2}) \right\}^2 \]

Where 10 <= m <= N – 1 is even.

FIXME: bias correction

NIST [SP1065] eq (30) page 29

Parameters

• **data** (np.array) – Input data. Provide either phase or frequency (fractional, adimensional).

• **rate** (float) – The sampling rate for data, in Hz. Defaults to 1.0

• **data_type** ("phase", "freq") – Data type, i.e. phase or frequency. Defaults to “phase”.

• **taus** (np.array) – Array of tau values, in seconds, for which to compute statistic. Optionally set taus=[“all”"l”"octave”"l”"decade”] for automatic tau-list generation.

allantools.mtie(data, rate=1.0, data_type='phase', taus=None)

Maximum Time Interval Error.

Parameters

• **data** (np.array) – Input data. Provide either phase or frequency (fractional, adimensional).

• **rate** (float) – The sampling rate for data, in Hz. Defaults to 1.0

• **data_type** ("phase", "freq") – Data type, i.e. phase or frequency. Defaults to “phase”.

Chapter 2. Implemented statistics functions
• **taus** (*np.array*) – Array of tau values, in seconds, for which to compute statistic. Optionally set taus=["all","octave","decade"] for automatic tau-list generation.

**Notes**

this seems to correspond to Stable32 setting “Fast(u)” Stable32 also has “Decade” and “Octave” modes where the dataset is extended somehow?

**allantools.tierms** (*data*, *rate*=1.0, *data_type='phase'*, *taus=None*)

Time Interval Error RMS.

**Parameters**

- **data** (*np.array*) – Input data. Provide either phase or frequency (fractional, adimensional).
- **rate** (*float*) – The sampling rate for data, in Hz. Defaults to 1.0
- **data_type** (*{'phase', 'freq'}*) – Data type, i.e. phase or frequency. Defaults to “phase”.
- **taus** (*np.array*) – Array of tau values, in seconds, for which to compute statistic. Optionally set taus=["all","octave","decade"] for automatic tau-list generation.

2.3 Real-Time Statistics

class **allantools.oadev_realtime** (*afs=[1], tau0=1.0, auto_afs=False, pts_per_decade=4*)

Overlapping Allan deviation in real-time from a stream of phase/frequency samples.

Dobrogowski & Kasznia https://doi.org/10.1109/FREQ.2007.4319204

**add_phase** (*xnew*)

add new phase point, in units of seconds

**update_S** (*idx*)

update S, sum-of-squares

class **allantools.tdev_realtime** (*afs=[1], tau0=1.0, auto_afs=False, pts_per_decade=4*)

Time deviation and Modified Allan deviation in real-time from a stream of phase/frequency samples.

Dobrogowski & Kasznia https://doi.org/10.1109/FREQ.2007.4319204

**add_phase** (*xnew*)

add new phase point

**update_S3n** (*idx*)

eqn (13) of paper

**update_S** (*idx*)

update S, sum-of-squares

**mdev**()

scale tdev to output mdev

class **allantools.ohdev_realtime** (*afs=[1], tau0=1.0, auto_afs=False, pts_per_decade=4*)

Overlapping Hadamard deviation in real-time from a stream of phase/frequency samples.

[Dobrogowski2007] Dobrogowski & Kasznia https://doi.org/10.1109/FREQ.2007.4319204

**add_phase** (*xnew*)

add new phase point

2.3. Real-Time Statistics
**update_S(idx)**
update S, sum-of-squares

## 2.4 Noise Generation

allantools.noise.white(*num_points=1024, b0=1.0, fs=1.0*)
White noise generator

Generate time series with white noise that has constant PSD = b0, up to the nyquist frequency fs/2.

The PSD is at ‘height’ b0 and extends from 0 Hz up to the nyquist frequency fs/2 (prefactor $\sqrt{b0 \times fs/2.0}$)

**Parameters**
- **num_points** (*int, optional*) – number of samples
- **b0** (*float, optional*) – desired power-spectral density in $[X^2/Hz]$ where X is the unit of x
- **fs** (*float, optional*) – sampling frequency, i.e. 1/fs is the time-interval between datapoints

**Returns** White noise sample

**Return type** *numpy.array*

allantools.noise.brown(*num_points=1024, b_minus2=1.0, fs=1.0*)
Brownian or random walk (diffusion) noise with $1/f^2$ PSD

Not really a color… rather Brownian or random-walk. Obtained by integrating white-noise.

**Parameters**
- **num_points** (*int, optional*) – number of samples
- **b_minus2** (*float, optional*) – desired power-spectral density is $b2 \times f^{-2}$
- **fs** (*float, optional*) – sampling frequency, i.e. 1/fs is the time-interval between datapoints

**Returns** Random walk sample

**Return type** *numpy.array*

allantools.noise.violet(*num_points=1024, b2=1, fs=1*)
Violet noise with $f^2$ PSD

Obtained by differentiating white noise

**Parameters**
- **num_points** (*int, optional*) – number of samples
- **b2** (*float, optional*) – desired power-spectral density is $b2 \times f^2$
- **fs** (*float, optional*) – sampling frequency, i.e. 1/fs is the time-interval between datapoints

**Returns** Violet noise sample

**Return type** *numpy.array*
allantools.noise.pink(num_points=1024, depth=80)

Pink noise (approximation) with 1/f PSD


**Parameters**
- **num_points** (*int, optional*) – number of samples
- **depth** (*int, optional*) – number of iteration for each point. High numbers are slower but generates a more correct spectrum on low-frequencies end.

**Returns** Pink noise sample

**Return type** numpy.array

### 2.5 Utilities

allantools.frequency2phase(freqdata, rate)

integrate fractional frequency data and output phase data

**Parameters**
- **freqdata** (*np.array*) – Data array of fractional frequency measurements (nondimensional)
- **rate** (*float*) – The sampling rate for phase or frequency, in Hz

**Returns** phasedata – Time integral of fractional frequency data, i.e. phase (time) data in units of seconds. For phase in units of radians, see phase2radians()

**Return type** np.array

allantools.phase2frequency(phase, rate)

Convert phase in seconds to fractional frequency

**Parameters**
- **phase** (*np.array*) – Data array of phase in seconds
- **rate** (*float*) – The sampling rate for phase, in Hz

**Returns** Data array of fractional frequency

**Return type** y

allantools.phase2radians(phasedata, v0)

Convert phase in seconds to phase in radians

**Parameters**
- **phasedata** (*np.array*) – Data array of phase in seconds
- **v0** (*float*) – Nominal oscillator frequency in Hz

**Returns** phase data in radians

**Return type** fi

allantools.edf_simple(N, m, alpha)

Equivalent degrees of freedom. Simple approximate formulae.

**Parameters**
• **N** *(int)* – the number of phase samples

• **m** *(int)* – averaging factor, \( \tau = m \times \tau_0 \)

• **alpha** *(int)* – exponent of \( f \) for the frequency PSD: ‘wp’ returns white phase noise. \( \alpha = +2 \) ‘wf’ returns white frequency noise. \( \alpha = 0 \) ‘fp’ returns flicker phase noise. \( \alpha = +1 \) ‘ff’ returns flicker frequency noise. \( \alpha = -1 \) ‘rf’ returns random walk frequency noise. \( \alpha = -2 \) If the input is not recognized, it defaults to idealized, uncorrelated noise with \((N-1)\) degrees of freedom.

**Notes**


**Returns** *edf* – Equivalent degrees of freedom

**Return type** *float*

```python
allantools.edf_greenhall(alpha, d, m, N, overlapping=False, modified=False, verbose=False)
```

returns Equivalent degrees of freedom

**Parameters**

• **alpha** *(int)* – noise type, +2...-4

• **d** *(int)* – 1 first-difference variance 2 Allan variance 3 Hadamard variance require \( \alpha + 2d > 1 \)

• **m** *(int)* – averaging factor \( \tau = m \times \tau_0 = m \times (1/rate) \)

• **N** *(int)* – number of phase observations (length of time-series)

• **overlapping** *(bool)* – True for oadev, ohdev

• **modified** *(bool)* – True for mdev, tdev

**Returns**

• **edf** *(float)* – Equivalent degrees of freedom

• Greenhall, Riley, 2004

• [https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20050061319.pdf](https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20050061319.pdf)

• **UNCERTAINTY OF STABILITY VARIANCES BASED ON FINITE DIFFERENCES**

**Notes**

Used for the following deviations (see [http://www.wriley.com/CI2.pdf](http://www.wriley.com/CI2.pdf) page 8) adev() oadev() mdev() tdev() hdev() ohdev()

```python
allantools.edf_totdev(N, m, alpha)
```

Equivalent degrees of freedom for Total Deviation

**FIXME:** what is the right behavior for \( \alpha \) outside 0,-1,-2?

NIST SP1065 page 41, Table 7

```python
allantools.edf_mtotdev(N, m, alpha)
```

Equivalent degrees of freedom for Modified Total Deviation

NIST SP1065 page 41, Table 8
Three Cornered Hat Method

Given three clocks A, B, C, we seek to find their variances $\sigma^2_A$, $\sigma^2_B$, $\sigma^2_C$. We measure three phase differences, assuming no correlation between the clocks, the measurements have variances:

\[
\begin{align*}
\sigma^2_{AB} & = \sigma^2_A + \sigma^2_B \\
\sigma^2_{BC} & = \sigma^2_B + \sigma^2_C \\
\sigma^2_{CA} & = \sigma^2_C + \sigma^2_A
\end{align*}
\]

Which allows solving for the variance of one clock as:

\[
\sigma^2_A = \frac{1}{2}(\sigma^2_{AB} + \sigma^2_{CA} - \sigma^2_{BC})
\]

and similarly cyclic permutations for $\sigma^2_B$ and $\sigma^2_C$.

Parameters

- `phasedata_ab (np.array)` – phase measurements between clock A and B, in seconds
- `phasedata_bc (np.array)` – phase measurements between clock B and C, in seconds
- `phasedata_ca (np.array)` – phase measurements between clock C and A, in seconds
- `rate (float)` – The sampling rate for phase, in Hz
- `taus (np.array)` – The tau values for deviations, in seconds
- `function (allantools deviation function)` – The type of statistic to compute, e.g. allantools.oadev

Returns

- `tau_ab (np.array)` – Tau values corresponding to output deviations
- `dev_a (np.array)` – List of computed values for clock A

References

http://www.wriley.com/3-CornHat.htm
3.1 The Dataset() class

New in version 2016.11

This class allows simple data handling.

class allantools.Dataset(data=None, rate=1.0, data_type='phase', taus=None)

Dataset class for Allantools

Example

```python
import numpy as np
# Load random data
a = allantools.Dataset(data=np.random.rand(1000))
# compute mdev
a.compute("mdev")
print(a.out["stat"])
```

compute() returns the result of the computation and also stores it in the object’s out member.

```
__init__(data=None, rate=1.0, data_type='phase', taus=None)
```

Initialize object with input data

Parameters

- **data** *(np.array)* – Input data. Provide either phase or frequency (fractional, adimensional)
- **rate** *(float)* – The sampling rate for data, in Hz. Defaults to 1.0
- **data_type** *(['phase', 'freq'])* – Data type, i.e. phase or frequency. Defaults to "phase".
- **taus** *(np.array)* – Array of tau values, in seconds, for which to compute statistic. Optionally set taus=['all'"'octave'"'decade"] for automatic calculation of taus list

Returns A Dataset() instance
Return type Dataset

inp = None
  input data Dict,

out = None
  output data Dict, to be populated by compute()

set_input (data, rate=1.0, data_type='phase', taus=None)
  Optional method if you chose not to set inputs on init

Parameters
  • data (np.array) – Input data. Provide either phase or frequency (fractional, adimensional)
  • rate (float) – The sampling rate for data, in Hz. Defaults to 1.0
  • data_type ({'phase', 'freq'}) – Data type, i.e. phase or frequency. Defaults to “phase”.
  • taus (np.array) – Array of tau values, in seconds, for which to compute statistic. Optionally set taus=[“all”|”octave”|”decade”] for automatic

compute (function)
  Evaluate the passed function with the supplied data.
  Stores result in self.out.

  Parameters function (str) – Name of the allantools function to evaluate
  Returns result – The results of the calculation.
  Return type dict

write_results (filename, digits=5, header_params={})
  Output result to text

  Save calculation results to disk. Will overwrite any existing file.

  Parameters
    • filename (str) – Path to the output file
    • digits (int) – Number of significant digits in output
    • header_params (dict) – Arbitrary dict of params to be included in header

  Returns

  Return type None

3.2 The Plot() class

New in version 2016.11

This class allows simple data plotting.

class allantools.Plot (no_display=False)
  A class for plotting data once computed by Allantools

  Example
import allantools
import numpy as np
a = allantools.Dataset(data=np.random.rand(1000))
a.compute("mdev")
b = allantools.Plot()
b.plot(a)
b.show()
CHAPTER 4

Development

4.1 To-do list

Here follows an un-ordered to do list:

- Statistics and core algorithms
  - The mtie_phase_fast approach to MTIE, using a binary tree (see BREGNI reference)
  - TheoH
  - Confidence intervals based on identified noise-type and equivalent degrees of freedom.
  - Bias corrections for biased statistics (totdev, mtotdev, htotdev, theo1)
  - Multi-threading for faster processing of (very) large datasets
  - Faster algorithms for mtotdev() and htotdev() which are currently very slow
- Improve documentation
- Improve packaging for PyPi and/or other packaging systems: Conda?, PPA for Ubuntu/Debian?
- Tests for different noise types according to IEEE 1139, include power-spectral-density calculations
- Conversion between phase noise and Allan variance
- Phase noise calculations and plots
- Comparison to other libraries such as GPSTk

Make sure your patch does not break any of the tests, and does not significantly reduce the readability of the code.

4.2 Documentation generation

See /docs for documentation in sphinx format. On Ubuntu this requires the python-sphinx and python-numpydoc packages. html/pdf documentation using sphinx can be built locally with:
this generates html documentation in docs/_build/html and pdf documentation in docs/_build/latex.

The sphinx documentation is also auto-generated online
  
  • https://allantools.readthedocs.org

### 4.3 Tests

The tests compare the output of allantools to other programs such as Stable32. Tests may be run using py.test (http://pytest.org). Package managers may install it with different binary names (e.g. pytest-3 for the python3 version on debian).

Slow tests are marked ‘slow’ and tests failing because of a known reason are marked ‘fails’. To run all tests:

```
$ py.test
```

To exclude known failing tests:

```
$ py.test -m "not fails" --duration=10
```

To exclude tests that run slowly:

```
$ py.test -m "not slow" --duration=10
```

To exclude both (note option change) and also check docstrings is ReST files

```
$ py.test -k "not (slow or fails)" --duration=10 --doctest-glob='*.rst'
```

To run the above command without installing the package:

```
$ python setup.py test --addopts "-k 'not (fails or slow)'"
```

Test coverage may be obtained with the (https://pypi.python.org/pypi/coverage) module:

```
coverage run --source allantools setup.py test --addopts "-k 'not (fails or slow)'"
coverage report  # Reports on standard output
coverage html  # Writes annotated source code as html in ./htmlcov/
```

On Ubuntu this requires packages python-pytest and python-coverage.

Testing on multiple python versions can be done with tox (https://testrun.org/tox)

```
$ tox
```

Tests run continuously on Travis-CI at https://travis-ci.org/aewallin/allantools

### 4.4 Notes for Pypi

Creating a source distribution

```
python setup.py sdist
```

This creates a package in dist/ Testing the source distribution. The install takes a long time while compiling numpy and scipy.

```
$ virtualenv tmp
$ tmp/bin/pip install dist/AllanTools-2016.2.tar.gz
$ tmp/bin/python
>>> import allantools
```

Uploading to PyPi. (requires a ~/.pypirc with username and password)

```
$ twine upload dist/*
```

Check result at https://pypi.python.org/pypi/AllanTools
CHAPTER 5

References

5.1 Code

- http://www.mathworks.com/matlabcentral/fileexchange/26659-allan-v3-0
- http://www.leapsecond.com/tools/adev_lib.c

5.2 Papers
Bibliography


[Rubiola2015] The Omega Counter, a Frequency Counter Based on the Linear Regression https://arxiv.org/abs/1506.05009


Index

Symbols

__init__() (allantools.Dataset method), 17
__init__() (allantools.Plot method), 19

A
add_phase() (allantools.oadev_realtime method), 11
add_phase() (allantools.ohdev_realtime method), 11
add_phase() (allantools.tdev_realtime method), 11
addev() (in module allantools), 5

B
brown() (in module allantools.noise), 12

C
compute() (allantools.Dataset method), 18

D
Dataset (class in allantools), 17

E
edf_greenhall() (in module allantools), 14
edf_mtotdev() (in module allantools), 14
edf_simple() (in module allantools), 13
edf_totdev() (in module allantools), 14

F
frequency2phase() (in module allantools), 13

H
hdev() (in module allantools), 7
htotdev() (in module allantools), 9

I
inp (allantools.Dataset attribute), 18

M
mdev() (allantools.tdev_realtime method), 11
mdev() (in module allantools), 7
mtie() (in module allantools), 10
mtotdev() (in module allantools), 9

O
oadev() (in module allantools), 6
oadev_realtime (class in allantools), 11
ohdev() (in module allantools), 7
ohdev_realtime (class in allantools), 11
out (allantools.Dataset attribute), 18

P
phase2frequency() (in module allantools), 13
phase2radians() (in module allantools), 13
pink() (in module allantools.noise), 12
Plot (class in allantools), 18
plot() (allantools.Plot method), 19

S
save() (allantools.Plot method), 19
set_input() (allantools.Dataset method), 18
show() (allantools.Plot method), 19

T
tdev() (in module allantools), 8
tdev_realtime (class in allantools), 11
theo1() (in module allantools), 10
tree_cornered_hat_phase() (in module allantools), 14
tierms() (in module allantools), 11
totdev() (in module allantools), 9
ttotdev() (in module allantools), 9

U
update_S() (allantools.oadev_realtime method), 11
update_S() (allantools.ohdev_realtime method), 12
update_S() (allantools.tdev_realtime method), 11
update_S3n() (allantools.tdev_realtime method), 11

V
violet() (in module allantools.noise), 12
W

white() (in module allantools.noise), 12
write_results() (allantools.Dataset method), 18