
multiptetau Documentation

Release 0.3.3.post10

Paul Müller

Jan 04, 2023

CONTENTS

1	General	1
1.1	Recommended literature	1
1.2	Obtaining multipletau	1
1.3	Citing multipletau	1
1.4	Usage	2
2	Methods	3
2.1	Autocorrelation	3
2.2	Cross-correlation	4
2.3	Cross-correlation (NumPy)	6
2.4	Constants	6
3	Examples	7
3.1	Comparison of correlation methods	7
	Python Module Index	11
	Index	13

GENERAL

Multipletau provides a multiple- algorithm for Python 2.7 and Python 3.x with `numpy` as its sole dependency.

Multiple- correlation is computed on a logarithmic scale (less data points are computed) and is thus much faster than conventional correlation on a linear scale such as `numpy.correlate()`.

1.1 Recommended literature

- Klaus Schaetzel and Rainer Peters; *Noise on multiple-tau photon correlation data*. Proc. SPIE 1430, Photon Correlation Spectroscopy: Multicomponent Systems, 109 (June 1, 1991); <http://doi.org/10.1117/12.44160>
- Thorsten Wohland, Rudolf Rigler, and Horst Vogel; *The Standard Deviation in Fluorescence Correlation Spectroscopy*. Biophysical Journal, 80 (June 1, 2001); [http://dx.doi.org/10.1016/S0006-3495\(01\)76264-9](http://dx.doi.org/10.1016/S0006-3495(01)76264-9)

1.2 Obtaining multipletau

If you have Python and `numpy` installed, simply run

```
pip install multipletau
```

The source code of multipletau is available at <https://github.com/FCS-analysis/multipletau>.

1.3 Citing multipletau

The multipletau package should be cited like this (replace “x.x.x” with the actual version of multipletau used and “DD Month YYYY” with a matching date).

cite

Paul Müller (2012) *Python multiple-tau algorithm* (Version x.x.x) [Computer program]. Available at <https://pypi.python.org/pypi/multipletau/> (Accessed DD Month YYYY)

You can find out what version you are using by typing (in a Python console):

```
>>> import multipletau
>>> multipletau.__version__
'0.3.0'
```

1.4 Usage

The package is straightforward to use. Here is a quick example:

```
>>> import numpy as np
>>> import multipletau
>>> a = np.linspace(2,5,42)
>>> v = np.linspace(1,6,42)
>>> multipletau.correlate(a, v, m=2)
array([[ 0.          , 569.56097561],
       [ 1.          , 549.87804878],
       [ 2.          , 530.37477692],
       [ 4.          , 491.85812017],
       [ 8.          , 386.39500297]])
```

METHODS

Summary:

<code>autocorrelate(a[, m, deltat, normalize, ...])</code>	Autocorrelation of a 1-dimensional sequence on a log2-scale.
<code>correlate(a, v[, m, deltat, normalize, ...])</code>	Cross-correlation of two 1-dimensional sequences on a log2-scale.
<code>correlate_numpy(a, v[, deltat, normalize, ...])</code>	Convenience function that wraps around <code>numpy.correlate()</code> and returns the correlation in the same format as <code>correlate()</code> does.

2.1 Autocorrelation

`multipletau.autocorrelate(a, m=16, deltat=1, normalize=False, copy=True, dtype=None, compress='average', ret_sum=False)`

Autocorrelation of a 1-dimensional sequence on a log2-scale.

This computes the correlation similar to `numpy.correlate()` for positive k on a base 2 logarithmic scale.

`numpy.correlate(a, a, mode="full") [len(a)-1:]()`

$$z_k = \sum_n a_n a_{n+k}$$

Parameters

- **a** (*array-like*) – input sequence
- **m** (*even integer*) – defines the number of points on one level, must be an even integer
- **deltat** (*float*) – distance between bins
- **normalize** (*bool*) – normalize the result to the square of the average input signal and the factor $M - k$.
- **copy** (*bool*) – copy input array, set to False to save memory
- **dtype** (*object to be converted to a data type object*) – The data type of the returned array and of the accumulator for the multiple-tau computation.
- **compress** (*str*) – strategy for propagating values to the next register
 - "average" (default): average two measurements when pushing to the next level of the correlator.
 - "first": use only the first value when pushing to the next level of the correlator.

- “second”: use only the second value when pushing to the next level of the correlator.

Using only the first or the second values during propagation completely removes the systematic error at the cost of increasing the statistical error. See <https://doi.org/10.1063/1.3491098> for a discussion on the effect of averaging.

- **ret_sum** (*bool*) – return the exact sum $z_k = \sum_n a_n a_{n+k}$. In addition $M - k$ is returned as an array of length N .

Returns

- **autocorrelation** (*ndarray of shape (N,2)*) – the lag time (1st column) and the autocorrelation (2nd column).
- **count** (*ndarray of length N*) – only returned if *ret_sum* is True; the value of $M - k$ for each row in *autocorrelation*.

Notes

Changed in version 0.1.6: Compute the correlation for zero lag time.

The algorithm computes the correlation with the convention of the curve decaying to zero.

For experiments like e.g. fluorescence correlation spectroscopy, the signal can be normalized to $M - k$ by invoking *normalize=True*.

For normalizing according to the behavior of `numpy.correlate()`, use *normalize=False*.

For complex arrays, this method falls back to the method *correlate()*.

Examples

```
>>> from multipletau import autocorrelate
>>> autocorrelate(range(42), m=2, dtype=np.float_)
array([[ 0.00000000e+00,  2.38210000e+04],
       [ 1.00000000e+00,  2.29600000e+04],
       [ 2.00000000e+00,  2.21000000e+04],
       [ 4.00000000e+00,  2.03775000e+04],
       [ 8.00000000e+00,  1.50612000e+04]])
```

2.2 Cross-correlation

`multipletau.correlate(a, v, m=16, deltat=1, normalize=False, copy=True, dtype=None, compress='average', ret_sum=False)`

Cross-correlation of two 1-dimensional sequences on a log2-scale.

This computes the cross-correlation similar to `numpy.correlate()` for positive k on a base 2 logarithmic scale.

`numpy.correlate(a, v, mode="full") [len(a)-1:]()`

$$z_k = \sum_n a_n v_{n+k}$$

Note that only the correlation in the positive direction is computed. To obtain the correlation for negative lag times swap the input variables *a* and *v*.

Parameters

- **a** (*array-like*) – input sequences with equal length

- **v** (*array-like*) – input sequences with equal length
- **m** (*even integer*) – defines the number of points on one level, must be an even integer
- **deltat** (*float*) – distance between bins
- **normalize** (*bool*) – normalize the result to the square of the average input signal and the factor $M - k$.
- **copy** (*bool*) – copy input array, set to False to save memory
- **dtype** (*object to be converted to a data type object*) – The data type of the returned array and of the accumulator for the multiple-tau computation.
- **compress** (*str*) – strategy for propagating values to the next register
 - “average” (default): average two measurements when pushing to the next level of the correlator.
 - “first”: use only the first value when pushing to the next level of the correlator.
 - “second”: use only the second value when pushing to the next level of the correlator.

Using only the first or the second values during propagation completely removes the systematic error at the cost of increasing the statistical error. See <https://doi.org/10.1063/1.3491098> for a discussion on the effect of averaging.
- **ret_sum** (*bool*) – return the exact sum $z_k = \sum_n a_n v_{n+k}$. In addition $M - k$ is returned as an array of length N.

Returns

- **cross_correlation** (*ndarray of shape (N,2)*) – the lag time (1st column), the cross-correlation (2nd column).
- **count** (*ndarray of length N*) – only returned if *ret_sum* is True; the value of $M - k$ for each row in *autocorrelation*.

Notes

Changed in version 0.1.6: Compute the correlation for zero lag time and correctly normalize the correlation for a complex input sequence *v*.

The algorithm computes the correlation with the convention of the curve decaying to zero.

For experiments like e.g. fluorescence correlation spectroscopy, the signal can be normalized to $M - k$ by invoking *normalize=True*.

For normalizing according to the behavior of `numpy.correlate()`, use *normalize=False*.

Examples

```
>>> from multipletau import correlate
>>> correlate(range(42), range(1,43), m=2, dtype=np.float_)
array([[ 0.00000000e+00,  2.46820000e+04],
       [ 1.00000000e+00,  2.38210000e+04],
       [ 2.00000000e+00,  2.29600000e+04],
       [ 4.00000000e+00,  2.12325000e+04],
       [ 8.00000000e+00,  1.58508000e+04]])
```

2.3 Cross-correlation (NumPy)

`multipletau.correlate_numpy(a, v, deltat=1, normalize=False, dtype=None, copy=True)`

Convenience function that wraps around `numpy.correlate()` and returns the correlation in the same format as `correlate()` does.

Parameters

- **a** (*array-like*) – input sequences
- **v** (*array-like*) – input sequences
- **deltat** (*float*) – distance between bins
- **normalize** (*bool*) – normalize the result to the square of the average input signal and the factor $M - k$. The resulting curve follows the convention of decaying to zero for large lag times.
- **copy** (*bool*) – copy input array, set to False to save memory
- **dtype** (*object to be converted to a data type object*) – The data type of the returned array.

Returns `cross_correlation` – the lag time (column 1) and the cross-correlation (column 2).

Return type ndarray of shape (N,2)

Notes

Changed in version 0.1.6: Removed false normalization when `normalize==False`.

2.4 Constants

`multipletau.core.ZERO_CUTOFF = 1e-15`

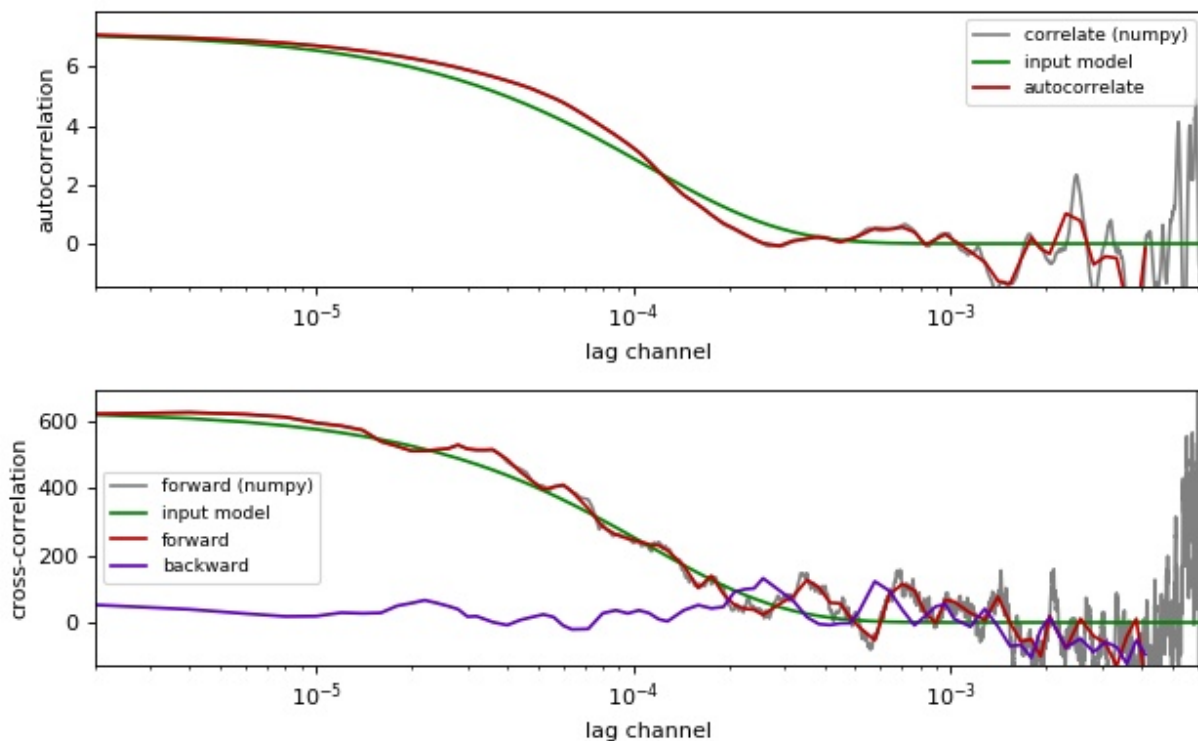
Defines the cutoff when the absolute mean trace divided by the median of the absolute value of the trace is treated as zero.

EXAMPLES

3.1 Comparison of correlation methods

This example illustrates the differences between the *multiplatau* correlation methods (*multiplatau.autocorrelate()*, *multiplatau.correlate()*) and *numpy.correlate()*.

This example requires *noise_generator.py* to be present in the current working directory.



compare_correlation_methods.py

```
1 from matplotlib import pylab as plt
2 import numpy as np
3
4 from multiplatau import autocorrelate, correlate, correlate_numpy
5
6 from noise_generator import noise_exponential, noise_cross_exponential
```

(continues on next page)

(continued from previous page)

```

7
8
9 # starting parameters
10 N = np.int(np.pi * 1e3)
11 countrate = 250. * 1e-3 # in Hz
12 taudiff = 55. # in us
13 deltat = 2e-6 # time discretization [s]
14 normalize = True
15
16 # time factor
17 taudiff *= deltat
18
19 # create noise for autocorrelation
20 data = noise_exponential(N, taudiff, deltat=deltat)
21 data -= np.average(data)
22 if normalize:
23     data += countrate
24 # perform autocorrelation (multipletau)
25 gac_mt = autocorrelate(data, deltat=deltat, normalize=normalize)
26 # numpy.correlate for comparison
27 gac_np = correlate_numpy(data, data, deltat=deltat,
28                          normalize=normalize)
29 # calculate model curve for autocorrelation
30 x = gac_np[:, 0]
31 amp = np.correlate(data - np.average(data), data - np.average(data),
32                   mode="valid")
33 if normalize:
34     amp /= len(data) * countrate**2
35 y = amp * np.exp(-x / taudiff)
36
37 # create noise for cross-correlation
38 a, v = noise_cross_exponential(N, taudiff, deltat=deltat)
39 a -= np.average(a)
40 v -= np.average(v)
41 if normalize:
42     a += countrate
43     v += countrate
44 gcc_forw_mt = correlate(a, v, deltat=deltat, normalize=normalize) # forward
45 gcc_back_mt = correlate(v, a, deltat=deltat, normalize=normalize) # backward
46 # numpy.correlate for comparison
47 gcc_forw_np = correlate_numpy(a, v, deltat=deltat, normalize=normalize)
48 # calculate the model curve for cross-correlation
49 xcc = gac_np[:, 0]
50 ampcc = np.correlate(a - np.average(a), v - np.average(v), mode="valid")
51 if normalize:
52     ampcc /= len(a) * countrate**2
53 ycc = ampcc * np.exp(-xcc / taudiff)
54
55 # plotting
56 fig = plt.figure(figsize=(8, 5))
57 fig.canvas.set_window_title('comparing multipletau')
58

```

(continues on next page)

(continued from previous page)

```

59 # autocorrelation
60 ax1 = fig.add_subplot(211)
61 ax1.plot(gac_np[:, 0], gac_np[:, 1], "-",
62         color="gray", label="correlate (numpy)")
63 ax1.plot(x, y, "g-", label="input model")
64 ax1.plot(gac_mt[:, 0], gac_mt[:, 1], "-",
65         color="#B60000", label="autocorrelate")
66 ax1.legend(loc=0, fontsize='small')
67 ax1.set_xlabel("lag channel")
68 ax1.set_ylabel("autocorrelation")
69 ax1.set_xscale('log')
70 ax1.set_xlim(x.min(), x.max())
71 ax1.set_ylim(-y.max()*0.2, y.max()*1.1)
72
73 # cross-correlation
74 ax2 = fig.add_subplot(212)
75 ax2.plot(gcc_forw_np[:, 0], gcc_forw_np[:, 1], "-",
76         color="gray", label="forward (numpy)")
77 ax2.plot(xcc, ycc, "g-", label="input model")
78 ax2.plot(gcc_forw_mt[:, 0], gcc_forw_mt[:, 1], "-",
79         color="#B60000", label="forward")
80 ax2.plot(gcc_back_mt[:, 0], gcc_back_mt[:, 1], "-",
81         color="#5D00B6", label="backward")
82 ax2.set_xlabel("lag channel")
83 ax2.set_ylabel("cross-correlation")
84 ax2.legend(loc=0, fontsize='small')
85 ax2.set_xscale('log')
86 ax2.set_xlim(x.min(), x.max())
87 ax2.set_ylim(-ycc.max()*0.2, ycc.max()*1.1)
88
89 plt.tight_layout()
90 plt.show()

```


PYTHON MODULE INDEX

m

`multipletau`, [1](#)

INDEX

A

`autocorrelate()` (*in module multipletau*), 3

C

`correlate()` (*in module multipletau*), 4

`correlate_numpy()` (*in module multipletau*), 6

M

module

 multipletau, 1

multipletau

 module, 1

Z

`ZERO_CUTOFF` (*in module multipletau.core*), 6