multipletau Documentation

Release 0.1.9

Paul Müller

Nov 16, 2017

Contents

CHAPTER 1

General

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

Multipe-τ 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

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__
'0.1.4'
```

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]])
```

CHAPTER 2

Methods

Summary:

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

2.1 Autocorrelation

`multipletau.`**`autocorrelate`**(*a*, *m=16*, *deltat=1*, *normalize=False*, *copy=True*, *dtype=None*)

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:]()
```

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

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.

Returns **autocorrelation** – the lag time (1st column) and the autocorrelation (2nd column).

Return type ndarray of shape (N,2)

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*)

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:]()
```

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

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

- **v** (*a,*) – 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.

Returns **cross_correlation** – the lag time (column 1) and the cross-correlation (column2).

Return type ndarray of shape (N,2)

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

- **v** (*a,*) – 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*.

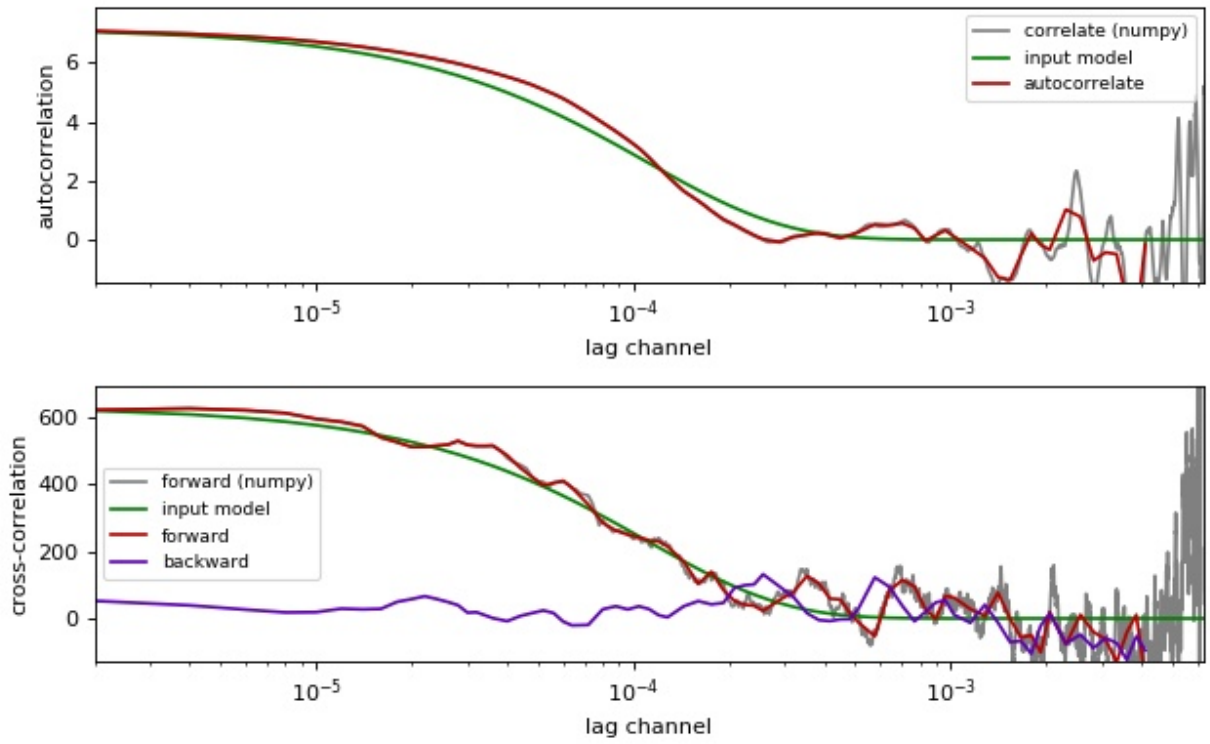
CHAPTER 3

Examples

3.1 Comparison of correlation methods

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

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



compare_correlation_methods.py

```
from matplotlib import pylab as plt
import numpy as np

from multipletau import autocorrelate, correlate, correlate_numpy

from noise_generator import noise_exponential, noise_cross_exponential


# starting parameters
N = np.int(np.pi * 1e3)
countrate = 250. * 1e-3  # in Hz
taudiff = 55.  # in us
deltat = 2e-6  # time discretization [s]
normalize = True

# time factor
taudiff *= deltat

# create noise for autocorrelation
data = noise_exponential(N, taudiff, deltat=deltat)
data -= np.average(data)
if normalize:
    data += countrate
# perform autocorrelation (multipletau)
gac_mt = autocorrelate(data, deltat=deltat, normalize=normalize)
# numpy.correlate for comparison
gac_np = correlate_numpy(data, data, deltat=deltat,
                         normalize=normalize)
# calculate model curve for autocorrelation
x = gac_np[:, 0]
amp = np.correlate(data - np.average(data), data - np.average(data),
                   mode="valid")
if normalize:
    amp /= len(data) * countrate**2
y = amp * np.exp(-x / taudiff)

# create noise for cross-correlation
a, v = noise_cross_exponential(N, taudiff, deltat=deltat)
a -= np.average(a)
v -= np.average(v)
if normalize:
    a += countrate
    v += countrate
gcc_forw_mt = correlate(a, v, deltat=deltat, normalize=normalize)  # forward
gcc_back_mt = correlate(v, a, deltat=deltat, normalize=normalize)  # backward
# numpy.correlate for comparison
gcc_forw_np = correlate_numpy(a, v, deltat=deltat, normalize=normalize)
# calculate the model curve for cross-correlation
xcc = gac_np[:, 0]
ampcc = np.correlate(a - np.average(a), v - np.average(v), mode="valid")
if normalize:
    ampcc /= len(a) * countrate**2
ycc = ampcc * np.exp(-xcc / taudiff)

# plotting
fig = plt.figure(figsize=(8, 5))
fig.canvas.set_window_title('comparing multipletau')
```

```
58
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()*.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()*.2, ycc.max()*1.1)
88
89  plt.tight_layout()
90  plt.show()
```

Python Module Index

m

Index