Elephant Cross Correlation Test

There may be an issue with the scaling of cross correlation at least for single pairs using `elephant.signal_processing.cross_correlation_function()` tested using the example below

In [1]:

```
%load_ext watermark
%matplotlib notebook

import numpy
import numpy as np
import matplotlib.pyplot as plt

import elephant
from elephant import signal_processing

import neo
from quantities import Hz, mV, sec, ms

%watermark -v -m -p numpy,scipy,neo,elephant
```

```
CPython 3.5.2
IPython 7.8.0

numpy 1.17.3
scipy 1.3.1
neo 0.7.2
elephant 0.6.3

compiler   : GCC 5.4.0 20160609
system     : Linux
release    : 4.4.0-165-generic
machine    : x86_64
processor  : x86_64
CPU cores  : 8
interpreter: 64bit
```

Example

The example problem converted from matlab (see http://www.mechanicalvibration.com/Using_cross_correlation_lin.html (http://www.mechanicalvibration.com/Using_cross_correlation_lin.html)).

In [2]:

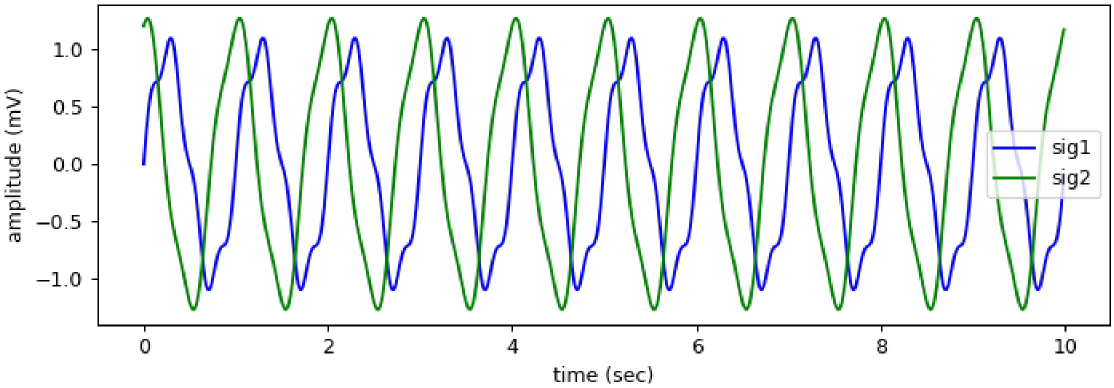
```
fs = 100.0
t = np.arange(0.0, 10.0, 1/fs)
f = 1
```

In [3]:

```
sig1 = 1.0 * np.sin(2.0*np.pi*f*t) + 0.15 * np.sin(2.0*np.pi*4.0*f*t)
sig2 = 1.2 * np.sin(2.0*np.pi*f*t + np.pi/2.0) +  0.15 * np.sin(2.0*np.pi*3.0*f*
t)
```

In [4]:

```
plt.figure(figsize=(8,3))
plt.plot(t, sig1,'b-', label='sig1')
plt.plot(t, sig2, 'g-', label='sig2')
plt.xlabel('time (sec)')
plt.ylabel('amplitude (mV)')
plt.legend()
plt.tight_layout()
```



Numpy Version

Performed cross correlation of these two signals using `numpy.correlate()` and, to match matlab used same normalization used in `matlab.xcorr()` ([https://www.mathworks.com/matlabcentral/answers/5275-algorithm-for-coeff-scaling-of-xcorr (https://www.mathworks.com/matlabcentral/answers/5275-algorithm-for-coeff-scaling-of-xcorr)](https://www.mathworks.com/matlabcentral/answers/5275-algorithm-for-coeff-scaling-of-xcorr)):

In [5]:

```
cxx = np.correlate(sig1, sig2, "full")/np.sqrt(np.sum(np.abs(sig1)**2)*np.sum(np
.abs(sig2)**2))
```

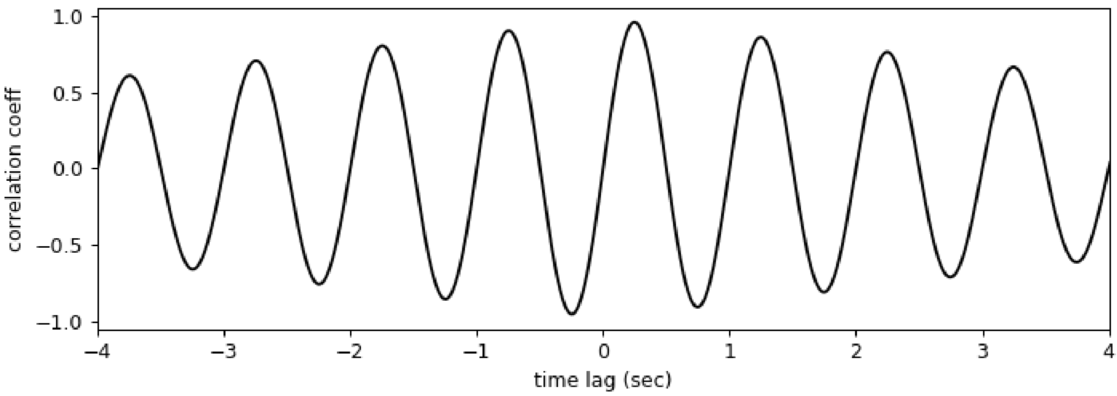
In [6]:

```
tx = np.arange(-t[-1],t[-1]+(0.5/fs),1/fs)
```

The cross correlation plot looks similar to original matlab example:

In [7]:

```
plt.figure(figsize=(8,3))
plt.plot(tx, cxx, 'k-')
plt.xlim(-4,4)
plt.xlabel('time lag (sec)')
plt.ylabel('correlation coeff')
plt.tight_layout()
```



Use peak of cross-correlation to find time shift between sig1 and sig2, which was 0.25 in original matlab example:

In [8]:

```
tshift = tx[np.argmax(cxx)]
tshift
```

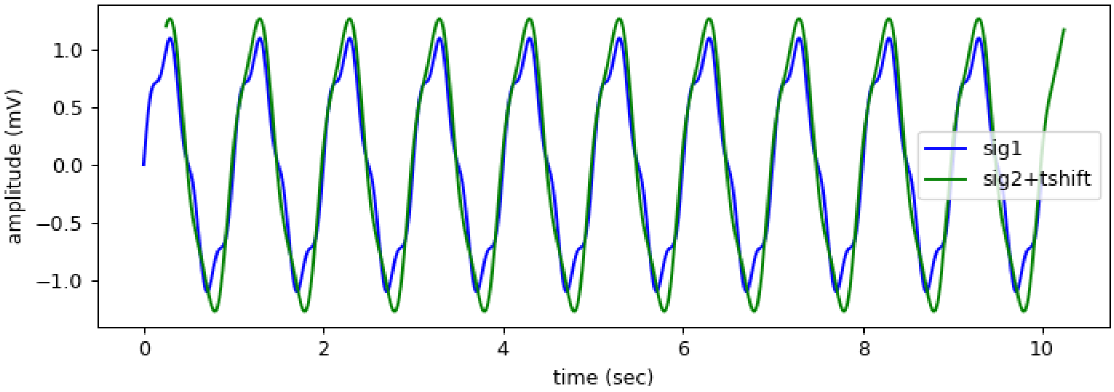
Out[8]:

```
0.2499999999997815
```

Now replot sig2 using this time shift to check signals align:

In [9]:

```
plt.figure(figsize=(8,3))
plt.plot(t, sig1,'b-', label='sig1')
plt.plot(t+tshift, sig2, 'g-', label='sig2+tshift')
plt.xlabel('time (sec)')
plt.ylabel('amplitude (mV)')
plt.legend()
plt.tight_layout()
```



Elephant Version

`elephant.signal_processing.cross_correlation_function()` does not give quite the same results as `numpy.correlate()` function

In [10]:

```
x = np.vstack([sig1,sig2]).T
x.shape
```

Out[10]:

```
(1000, 2)
```

In [11]:

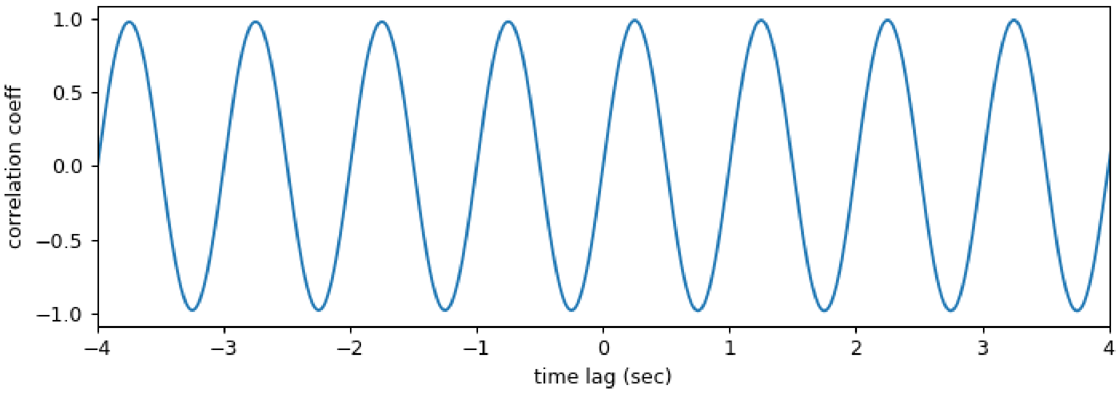
```
signal = neo.AnalogSignal(x, units='mV', t_start=0.*sec, sampling_rate=fs*Hz, dt
ype=float)
```

In [12]:

```
rho = elephant.signal_processing.cross_correlation_function(signal, [0,1])
```

In [13]:

```
plt.figure(figsize=(8,3))
plt.plot(rho.times, rho)
plt.xlabel('time lag (sec)')
plt.ylabel('correlation coeff')
plt.xlim(-4,4)
plt.tight_layout()
```



Differences...

Examined `cross_correlation_function()` code (https://github.com/NeuralEnsemble/elephant/blob/master/elephant/signal_processing.py (https://github.com/NeuralEnsemble/elephant/blob/master/elephant/signal_processing.py)). Essentially it performs an FFT convolution of normalized signals, where tau is the time lags and xcorr is correlation coefficient:

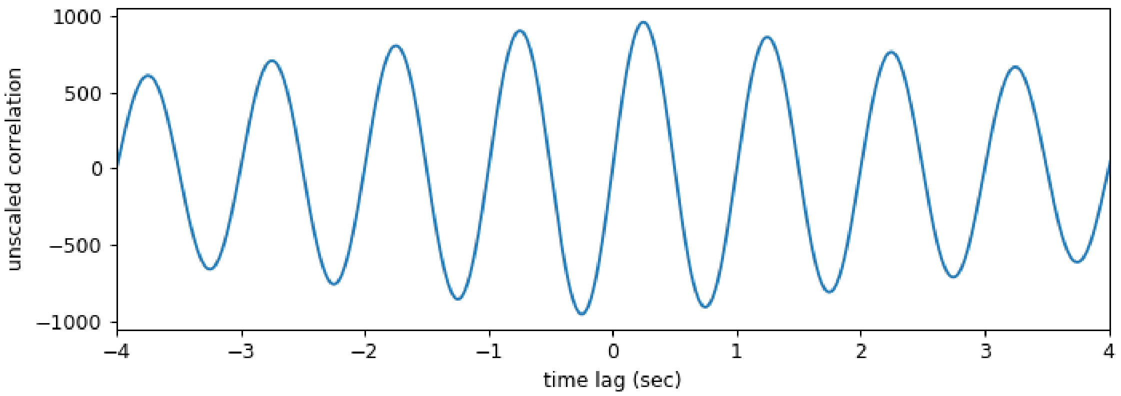
In [14]:

```
from scipy.signal import fftconvolve
from scipy.stats import zscore

zsig1 = zscore(sig1)
zsig2 = zscore(sig2)

nt = np.shape(zsig1)[0]

tau = (np.arange(nt) - nt//2)

xcorr = fftconvolve(zsig1, zsig2[::-1], mode='same')

# did not perform scaling of xcorr:
# xcorr = xcorr / npm.repmat((nt-abs(tau)), nch, 1).T
```

The unscaled "correlation coefficient" appears same as numpy correlate():

In [15]:

```
plt.figure(figsize=(8,3))
plt.plot(tau/fs, xcorr)
plt.xlabel('time lag (sec)')
plt.ylabel('unscaled correlation')
plt.xlim(-4,4)
plt.tight_layout()
```



The time shift matches numpy result too:

In [16]:

```
tshift = tau[np.argmax(xcorr)]/fs
tshift
```

Out[16]:

```
0.25
```

Conclusion

This suggests at least for single pairs there may be a problem in the scaling of xcorr. This is scaling step for a single channel case (one pair):

In [17]:

```
nch = 1
R = np.matlib.repmat((nt-abs(tau)), nch, 1).T
```

In [18]:

```
R.shape
```

Out[18]:

```
(1000, 1)
```

In [19]:

```
np.squeeze(R)
```

Out[19]:

```
array([ 500,  501,  502,  503,  504,  505,  506,  507,  508,  509,
510,
        511,  512,  513,  514,  515,  516,  517,  518,  519,  520,
521,
        522,  523,  524,  525,  526,  527,  528,  529,  530,  531,
532,
        533,  534,  535,  536,  537,  538,  539,  540,  541,  542,
543,
        544,  545,  546,  547,  548,  549,  550,  551,  552,  553,
554,
        555,  556,  557,  558,  559,  560,  561,  562,  563,  564,
565,
        566,  567,  568,  569,  570,  571,  572,  573,  574,  575,
576,
        577,  578,  579,  580,  581,  582,  583,  584,  585,  586,
587,
        588,  589,  590,  591,  592,  593,  594,  595,  596,  597,
598,
        599,  600,  601,  602,  603,  604,  605,  606,  607,  608,
609,
        610,  611,  612,  613,  614,  615,  616,  617,  618,  619,
620,
        621,  622,  623,  624,  625,  626,  627,  628,  629,  630,
631,
        632,  633,  634,  635,  636,  637,  638,  639,  640,  641,
642,
        643,  644,  645,  646,  647,  648,  649,  650,  651,  652,
653,
        654,  655,  656,  657,  658,  659,  660,  661,  662,  663,
664,
        665,  666,  667,  668,  669,  670,  671,  672,  673,  674,
675,
        676,  677,  678,  679,  680,  681,  682,  683,  684,  685,
686,
        687,  688,  689,  690,  691,  692,  693,  694,  695,  696,
697,
        698,  699,  700,  701,  702,  703,  704,  705,  706,  707,
708,
        709,  710,  711,  712,  713,  714,  715,  716,  717,  718,
719,
        720,  721,  722,  723,  724,  725,  726,  727,  728,  729,
730,
        731,  732,  733,  734,  735,  736,  737,  738,  739,  740,
741,
        742,  743,  744,  745,  746,  747,  748,  749,  750,  751,
752,
        753,  754,  755,  756,  757,  758,  759,  760,  761,  762,
763,
        764,  765,  766,  767,  768,  769,  770,  771,  772,  773,
774,
        775,  776,  777,  778,  779,  780,  781,  782,  783,  784,
785,
        786,  787,  788,  789,  790,  791,  792,  793,  794,  795,
796,
        797,  798,  799,  800,  801,  802,  803,  804,  805,  806,
807,
        808,  809,  810,  811,  812,  813,  814,  815,  816,  817,
818,
        819,  820,  821,  822,  823,  824,  825,  826,  827,  828,
```

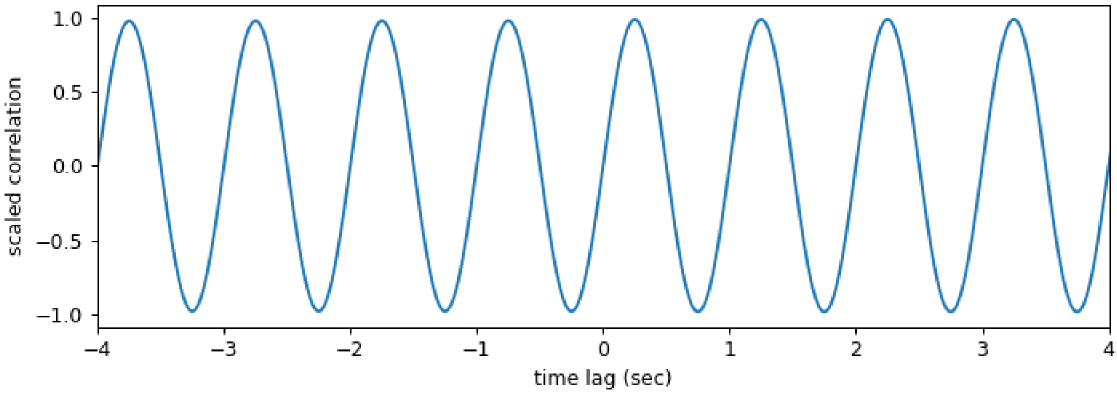
```
829,
         830,  831,  832,  833,  834,  835,  836,  837,  838,  839,
840,
         841,  842,  843,  844,  845,  846,  847,  848,  849,  850,
851,
         852,  853,  854,  855,  856,  857,  858,  859,  860,  861,
862,
         863,  864,  865,  866,  867,  868,  869,  870,  871,  872,
873,
         874,  875,  876,  877,  878,  879,  880,  881,  882,  883,
884,
         885,  886,  887,  888,  889,  890,  891,  892,  893,  894,
895,
         896,  897,  898,  899,  900,  901,  902,  903,  904,  905,
906,
         907,  908,  909,  910,  911,  912,  913,  914,  915,  916,
917,
         918,  919,  920,  921,  922,  923,  924,  925,  926,  927,
928,
         929,  930,  931,  932,  933,  934,  935,  936,  937,  938,
939,
         940,  941,  942,  943,  944,  945,  946,  947,  948,  949,
950,
         951,  952,  953,  954,  955,  956,  957,  958,  959,  960,
961,
         962,  963,  964,  965,  966,  967,  968,  969,  970,  971,
972,
         973,  974,  975,  976,  977,  978,  979,  980,  981,  982,
983,
         984,  985,  986,  987,  988,  989,  990,  991,  992,  993,
994,
         995,  996,  997,  998,  999, 1000,  999,  998,  997,  996,
995,
         994,  993,  992,  991,  990,  989,  988,  987,  986,  985,
984,
         983,  982,  981,  980,  979,  978,  977,  976,  975,  974,
973,
         972,  971,  970,  969,  968,  967,  966,  965,  964,  963,
962,
         961,  960,  959,  958,  957,  956,  955,  954,  953,  952,
951,
         950,  949,  948,  947,  946,  945,  944,  943,  942,  941,
940,
         939,  938,  937,  936,  935,  934,  933,  932,  931,  930,
929,
         928,  927,  926,  925,  924,  923,  922,  921,  920,  919,
918,
         917,  916,  915,  914,  913,  912,  911,  910,  909,  908,
907,
         906,  905,  904,  903,  902,  901,  900,  899,  898,  897,
896,
         895,  894,  893,  892,  891,  890,  889,  888,  887,  886,
885,
         884,  883,  882,  881,  880,  879,  878,  877,  876,  875,
874,
         873,  872,  871,  870,  869,  868,  867,  866,  865,  864,
863,
         862,  861,  860,  859,  858,  857,  856,  855,  854,  853,
852,
         851,  850,  849,  848,  847,  846,  845,  844,  843,  842,
841,
```

```
            840,  839,  838,  837,  836,  835,  834,  833,  832,  831,
    830,
            829,  828,  827,  826,  825,  824,  823,  822,  821,  820,
    819,
            818,  817,  816,  815,  814,  813,  812,  811,  810,  809,
    808,
            807,  806,  805,  804,  803,  802,  801,  800,  799,  798,
    797,
            796,  795,  794,  793,  792,  791,  790,  789,  788,  787,
    786,
            785,  784,  783,  782,  781,  780,  779,  778,  777,  776,
    775,
            774,  773,  772,  771,  770,  769,  768,  767,  766,  765,
    764,
            763,  762,  761,  760,  759,  758,  757,  756,  755,  754,
    753,
            752,  751,  750,  749,  748,  747,  746,  745,  744,  743,
    742,
            741,  740,  739,  738,  737,  736,  735,  734,  733,  732,
    731,
            730,  729,  728,  727,  726,  725,  724,  723,  722,  721,
    720,
            719,  718,  717,  716,  715,  714,  713,  712,  711,  710,
    709,
            708,  707,  706,  705,  704,  703,  702,  701,  700,  699,
    698,
            697,  696,  695,  694,  693,  692,  691,  690,  689,  688,
    687,
            686,  685,  684,  683,  682,  681,  680,  679,  678,  677,
    676,
            675,  674,  673,  672,  671,  670,  669,  668,  667,  666,
    665,
            664,  663,  662,  661,  660,  659,  658,  657,  656,  655,
    654,
            653,  652,  651,  650,  649,  648,  647,  646,  645,  644,
    643,
            642,  641,  640,  639,  638,  637,  636,  635,  634,  633,
    632,
            631,  630,  629,  628,  627,  626,  625,  624,  623,  622,
    621,
            620,  619,  618,  617,  616,  615,  614,  613,  612,  611,
    610,
            609,  608,  607,  606,  605,  604,  603,  602,  601,  600,
    599,
            598,  597,  596,  595,  594,  593,  592,  591,  590,  589,
    588,
            587,  586,  585,  584,  583,  582,  581,  580,  579,  578,
    577,
            576,  575,  574,  573,  572,  571,  570,  569,  568,  567,
    566,
            565,  564,  563,  562,  561,  560,  559,  558,  557,  556,
    555,
            554,  553,  552,  551,  550,  549,  548,  547,  546,  545,
    544,
            543,  542,  541,  540,  539,  538,  537,  536,  535,  534,
    533,
            532,  531,  530,  529,  528,  527,  526,  525,  524,  523,
    522,
            521,  520,  519,  518,  517,  516,  515,  514,  513,  512,
    511,
            510,  509,  508,  507,  506,  505,  504,  503,  502,  501])
```

So it appears that xcorr is scaled by abs(lag) which compensates for decay from peak correlation:

In [20]:

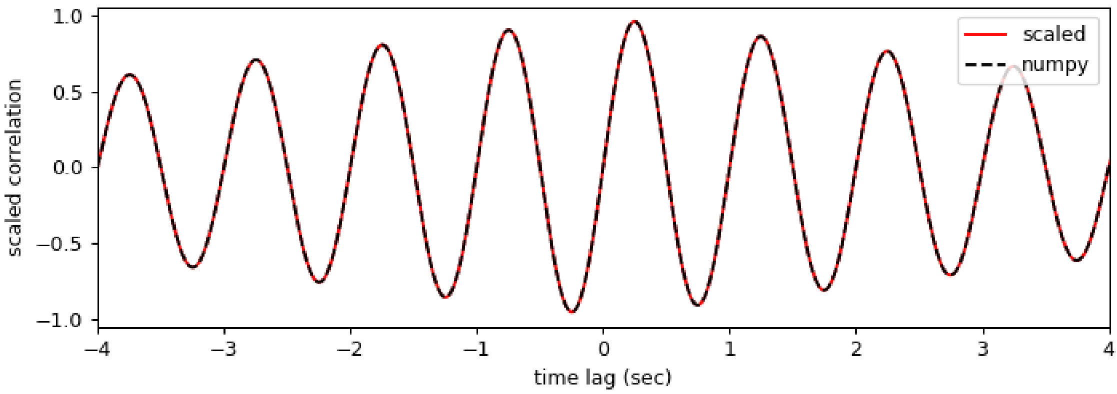
```
plt.figure(figsize=(8,3))
plt.plot(tau/fs, xcorr/np.squeeze(R))
plt.xlabel('time lag (sec)')
plt.ylabel('scaled correlation')
plt.xlim(-4,4)
plt.tight_layout()
```



Whereas dividing by sample length appears to scale correlation identically to numpy version:

In [21]:

```
plt.figure(figsize=(8,3))
plt.plot(tau/fs, xcorr/nt,'r-',label='scaled')
plt.plot(tx, cxx, 'k--', label='numpy')
plt.xlabel('time lag (sec)')
plt.ylabel('scaled correlation')
plt.xlim(-4,4)
plt.legend()
plt.tight_layout()
```



In [ ]: