

Correlation Library for MATLAB

Raul C. Mureşan
Version 1.2
March 4, 2014

Table of contents

INTRODUCTION.....	1
DESCRIPTION.....	1
MATLAB FUNCTIONS.....	1
<i>List of functions</i>	1
<i>Input data formatting</i>	2
<i>Function calls</i>	2
<i>Output data</i>	3
<i>Examples of usage</i>	4

Introduction

Correlation Library is a C++ and MATLAB library implemented by Raul C. Mureşan, which allows one to compute cross and scaled correlation. The code is free for non-commercial purposes. For details on the concept of scaled correlation, see the paper "Scaled Correlation Analysis", authors: Danko Nikolic, Wolf Singer, and Raul C. Muresan.

Description

The Correlation Library offers support for computing cross and scaled correlation between a pair of continuous valued, binary valued, or mixed signals.

For each method of computing the correlation and each combination of signals, there is one specialized function that can handle the respective operations.

The interface for MATLAB has been written in C++ and consists of 6 dll files (mexw64 on 64 bit platforms) that can be readily used in the MATLAB environment as functions.

MATLAB functions

List of functions

- **ComputeClassicCC_CC**: A function that computes the cross-correlation of two continuous valued signals (Continuous - Continuous). Implemented in file ComputeClassicCC_CC.dll (ComputeClassicCC_CC.mexw64 on 64 bit platforms).
- **ComputeClassicSTA**: A function that computes the cross-correlation of a vector of time stamps corresponding to events of a binary signal and a continuous valued signal (Binary – Continuous, also known in Neuroscience as Spike-Triggered Average – STA). Implemented in file ComputeClassicSTA.dll (ComputeClassicSTA.mexw64 on 64 bit platforms).
- .
- **ComputeClassicCC_BB**: A function that computes the cross-correlation for two vectors of time stamps corresponding to events of binary signals (Binary - Binary). Implemented in file ComputeClassicCC_BB.dll (ComputeClassicCC_BB.mexw64 on 64 bit platforms).
- **ComputeScaledCC_CC**: A function that computes the scaled correlation for two continuous valued signals (Continuous - Continuous). Implemented in file ComputeScaledCC_CC.dll (ComputeScaledCC_CC.mexw64 on 64 bit platforms).
- **ComputeScaledSTA**: A function that computes the scaled correlation for a vector of time stamps corresponding to events of a binary signal and a continuous signal (Binary - Continuous, also known in Neuroscience as Spike-Triggered Average – STA). Implemented in file ComputeScaledSTA.dll (ComputeScaledSTA.mexw64 on 64 bit platforms).
- **ComputeScaledCC_BB**: A function that computes the scaled correlation for two vectors of time stamps corresponding to events of binary signals (Binary - Binary). Implemented in file ComputeScaledCC_BB.dll (ComputeScaledCC_BB.mexw64 on 64 bit platforms).

Input data formatting

All functions respect the same standard for accepting the input data. For a binary input data, a vector of time stamps is expected as input while for a continuous valued input, a digitized vector of samples is expected as input.

When multiple measurements (trials) are available for the data, the functions can handle multiple trials. The length of the trial, in original sampling units, has to be specified to the function that is called. The function determines how many trials are available, computes the correlogram on each trial and finally averages the correlograms into one single correlogram.

When multiple trials are available, the input should be structured as follows:

```
|.....Trial 1.....||.....Trial 2.....||.....Trial 3.....||.....Trial 4.....||.....Trial 5.....|
```

- For continuous signals, there must be as many samples in each trial as the previously specified trial length. In any case, the method only considers as many samples as the length of the sample vector provided as input.
- For binary signals, the number of available trials is decided by taking into account the time stamp with the largest value. The method then searches, for each trial, the presence of valid time stamps. Time stamps for consecutive trials have to have increasing values. The trial to which a time stamp belongs is decided by taking into account the length of a trial in sampling units.

If the user cannot structure the trials as a contiguous series of data points, then the best option is to pass a single trial to the function, get the correlogram for each trial, and then the user should compute the average correlogram.

Note:

For the case of scaled correlation on multiple trials, the user can use the sum of coefficients of correlation for each bin of the correlogram and the count of coefficients to efficiently compute the average, smooth, scaled correlogram (see section Output Data, Method 2). Averaging the correlation coefficients directly gives a better and smoother estimate of the final scaled correlogram than averaging the individual, per-trial, scaled correlograms.

Function calls

For the two types of correlations the functions expose an interface to compute the cross correlation and the scaled correlation respectively. Depending on the type of the input data, the methods accept different parameter types. However, they have a general format:

- ComputeClassic...(CorrelationWindow,TrialLength,VectorA,VectorB,NormalizeCorrelogram)
- ComputeScaled...(ScaleSegmentSize,CorrelationWindow,TrialLength,VectorA,VectorB,UseFisherZTransform)
 - CorrelationWindow specifies the size of the correlation. For example, for a correlogram with lags from -100 to +100, the value of this parameter should be set to a value of 100.

- ScaleSegmentSize represents, for the case of scaled correlation, the scale on which the correlation will be computed. For example if we are interested in synchrony in the gamma range (> 30 Hz) we could set it to 33 ms.
- TrialLength is the length of a single trial in sampling units. If the input vectors have a larger size than TrialLength, for continuous signals, or if the last time stamp is larger than TrialLength-1, for binary signals, then the function will automatically identify more trials and compute the final correlation by averaging over trials.
- VectorA is the first input signal. For continuous variables, it should be a vector of samples, for binary variables it should contain integer numbers with the time stamps when events occurred (e.g. neuronal spike times). The input vector data should be supplied along the row.
- VectorB is the second input signal. For continuous variables, it should be a vector of samples, for binary variables it should contain integer numbers with the time stamps when events occurred (e.g. neuronal spike times). The input vector data should be supplied along the row.
- NormalizeCorrelogram will determine the function to normalize the cross correlogram to the rate of the first binary signal for the case of binary-binary or binary-continuous pairs of data, or to the number of samples in the first signal for continuous-continuous pairs of input signals. Set to a value of 1 to normalize, or to 0 to leave the correlation not normalized.

For the case of classic cross correlation on continuous signal pairs this parameter is a flag and can have multiple values:

- 0 – the correlogram is unnormalized and biased;
- 1 – the correlogram is normalized and biased;
- 2 – the correlogram is unnormalized and unbiased;
- 3 – the correlogram is normalized and unbiased.
- UseFisherZTransform instructs the function that computes scaled correlation to average the correlation coefficients for each bin by using the Fisher Z transform. Sometimes, when the distributions of coefficients are highly skewed, such a transform can improve the estimation of the average. Set to a value of 1 to use the Z Fisher transform, or to 0 to use the normal arithmetic average.

Output data

The functions that compute the cross correlation return always a single vector of doubles, with a size of $2 * \text{CorrelationWindow} + 1$. The vector stores the cross correlogram. In case there was an error or the correlation could not be defined because of empty input arrays, the vector is filled with Not a Number (NaN) MATLAB constants. The user should check and treat these cases appropriately.

The functions that compute the scaled correlation return a matrix having 3 rows and $2 * \text{CorrelationWindow} + 1$ columns. The first row is represented by the scaled correlogram. In case there was an error or the correlation could not be defined because of empty input arrays or lack of variance, the vector is filled, for the corresponding bin, with Not a Number (NaN) MATLAB constants. It is frequently the case that some bins of the scaled correlogram, on pairs of binary-binary data, are NaN, because the correlation could not be defined for the respective bin (for details see the Scaled Correlation Analysis). The user should check and treat these cases appropriately. It is important that when more scaled correlograms are averaged, the NaN values are discarded; the counts of values that will be taken into the average should not include the NaNs. It is not correct to replace NaNs with 0 and average the correlograms because the estimated average will be biased towards 0. To avoid this problem, the functions return two more rows, containing the sum of correlation coefficients for each bin and the counts of valid (\neq NaN) correlation coefficients for the respective bin. The user should sum the sums of correlation coefficients and the counts for each bin and then divide the first to the second to get the correct estimate of the average scaled correlogram. The second row of the return matrix contains the sums of correlation coefficients. The third row contains the counts of

correlation coefficients that were summed up. The methods for computing the average scaled correlogram are shown below:

Method 1: average the scaled correlograms. Not as accurate as Method 2. Use Method 2 whenever possible to compute the average scaled correlogram.

```
%Average scaled correlograms variant:

%Set the buffers to 0
Sums = zeros(1,2*iCorrelationWindow+1);
Counts = zeros(1,2*iCorrelationWindow+1);

%For each trial compute a scaled correlogram and sum them up with care
for trial=1:TrialNumber ;
    ResultMatrix = ComputeScaled...
    for i=1:2*CorrelationWindow+2 ;
        if isnan(ResultMatrix(1,i)) == 0 %the first row of the result is the scaled correlogram
            Sums(i) = Sums(i) + ResultMatrix(1,i);
            Counts(i) = Counts(i) + 1; %only increment counts if the bin is not NaN
        end;
    end;
end;

%Compute the correct average of scaled correlograms
ScaledCorrelogram = Sums ./ Counts;
```

Method 2: use the sums of correlation coefficients and their counts. This method is much better since it computes the correct average scaled correlogram. Averaging scaled correlograms yields less accurate results because it produces an average of averages (each scaled correlogram is already an average of correlation coefficients for each bin) which is not equal to the true average of all correlation coefficients. Use this second method whenever you need to compute the average scaled correlogram.

```
%Average the correlation coefficients variant:

%Set the buffers to 0
Sums = zeros(1,2*iCorrelationWindow+1);
Counts = zeros(1,2*iCorrelationWindow+1);

%For each trial extract the sums of coefficients and their counts
for trial=1:TrialNumber ;
    ResultMatrix = ComputeScaled...
    for i=1:2*CorrelationWindow+2 ;
        if isnan(ResultMatrix(2,i)) == 0 %the second row of the result = sum of coefficients
            Sums(i) = Sums(i) + ResultMatrix(2,i); %sums of correlation coefficients
            Counts(i) = Counts(i) + ResultMatrix(3,i); %counts of correlation coefficients
        end;
    end;
end;

%Compute the correct average scaled correlogram (best estimation)
ScaledCorrelogram = Sums ./ Counts;
```

Examples of usage

Cross-correlation on a pair of continuous-continuous signals:

```
%Classic correlation analysis: Cross Correlation for Continuous - Continuous signals
%Example of using the ComputeClassicCC_CC.dll library
%Author: Raul C. Muresan, 25.06.2007
%ATTENTION: The samples should be given in vectors in row format (values along the row, not along
the column)

clear all;
```

```

%Some example values for the parameters
iTrialLength = 2000;           %Suppose we have a sampling frequency of 1 kHz. We choose a trial length
of 2000 ms.
iCorrelationWindow = 100;     %We are interested in a cross correlogram of -100..100 ms.
iTrialNumber = 5;             %Suppose we have 5 trials
fNoiseAmplitude = 1.91;       %Some noise that is added to the first signal to produce a second
correlated signal

%Produce the first LFP
daLFPsSamplesA = zeros(1,iTrialLength*iTrialNumber);
for i=1:iTrialLength*iTrialNumber;
    daLFPsSamplesA(i) = sin(2*3.14/50*1000*i);
end
daNoise = 2*fNoiseAmplitude*(0.5*ones(size(daLFPsSamplesA)) - rand(size(daLFPsSamplesA)));
daLFPsSamplesA = daLFPsSamplesA + daNoise;

%Produce the second LFP
daLFPsSamplesB = zeros(1,iTrialLength*iTrialNumber);
for i=1:iTrialLength*iTrialNumber;
    daLFPsSamplesB(i) = sin(2*3.14/50*1000*i);
end
daNoise = 2*fNoiseAmplitude*(0.5*ones(size(daLFPsSamplesB)) - rand(size(daLFPsSamplesB)));
daLFPsSamplesB = daLFPsSamplesB + daNoise;

%Compute the cross correlation on continuous signals
%The function expects 5 parameters:
%1. CorrelationWindow - Integer Number; Scalar - The size of the cross correlation window (eg. 80
for a cross correlation with lags of -80..+80); use the same units as the sampling units
%2. TrialLength - Integer Number; Scalar - The size of the trial in sampling units
%3. SamplesA - Vector of Doubles: Samples - The vector holding the samples of the first channel
%4. SamplesB - Vector of Doubles: Samples - The vector holding the samples of the second channel
%5. NormalizeFlag - Integer Number; Scalar - Flag that decides how to normalize the correlogram: 0 -
unnormalized, biased; 1 - normalized, biased; 2 - unnormalized, unbiased; 3 - normalized, unbiased;
normalization removes the dependency on the length of the signals and gives values in cross-product
units of the continuous variables; unbiased removes the dependency on the finite length of signal
windows
%ATTENTION: the function returns NaN (Not a Number) values if the input buffers are empty

CrossCorrelogram =
ComputeClassicCC_CC(iCorrelationWindow,iTrialLength,daLFPsSamplesA,daLFPsSamplesB,0);

%Plot the classic cross-correlogram
x = -100:1:100;
bar(x,CrossCorrelogram,'r');

```

Cross-correlation on a pair of binary-continuous signals (Spike-Triggered Average – STA):

```

%Classic correlation analysis: Spike-Triggered Average (STA)
%Example of using the ComputeClassicSTA.dll library
%Author: Raul C. Muresan, 14.05.2007
%ATTENTION: If you supply the time stamps and the samples for a single trial, make sure that the
time stamps are relative to the beginning of the trial
%The time stamps and samples should be given in vectors in row format (values along the row, not
along the column)

clear all;

%Some example values for the parameters
iTrialLength = 2000;           %Suppose we have a sampling frequency of 1 kHz. We choose a trial length
of 2000 ms.
iCorrelationWindow = 100;     %We are interested in a cross correlogram of -100..100 ms.
iTrialNumber = 20;           %Suppose we have 20 trials
fFiringRate = 10;            %Suppose an average firing rate of 10 Hz => 20 Spikes per trial

%Produce some random time stamps (spikes of the neuron)
iaTimeStampsNeuronA = sort(round(iTrialLength * iTrialNumber * rand(1,iTrialNumber * fFiringRate *
iTrialLength / 1000)), 'ascend');

%Produce an LFP
daLFPsSamplesB = zeros(1,iTrialLength*iTrialNumber);
for i=1:iTrialLength*iTrialNumber;
    daLFPsSamplesB(i) = sin(2*3.14/50*1000*i);
end

```

```

%Compute the spike triggered average
%The function expects 5 parameters:
%1. CorrelationWindow - Integer Number; Scalar - The size of the cross correlation window (eg. 80
for a cross correlation with lags of -80..+80); use the same units as the sampling of your time
stamps
%2. TrialLength - Integer Number; Scalar - The size of the trial in sampling units
%3. TimeStampsA - Vector of Integer Numbers: Time Stamps - The vector holding the time stamps of the
neuron
%4. SamplesB - Vector of Double Numbers: Samples - The vector holding the samples of the continuous
signal
%5. Normalize - Integer Number; Scalar - Set to 1 if you want to normalize the correlogram or to 0
otherwise
%ATTENTION: the function returns NaN (Not a Number) values if there is no matching time stamp of
events in the binary signal with the continuous samples (that start at time t=0!!)

STA = ComputeClassicSTA(iCorrelationWindow,iTrialLength,iaTimeStampsNeuronA,daLFPSamplesB,1);

%Plot the classic STA correlogram
x = -100:1:100;
bar(x,STA,'r');

```

Cross-correlation on a pair of binary-binary signals:

```

%Classic correlation analysis: Cross Correlation for Binary - Binary signals
%Example of using the ComputeClassicCC_BB.dll library
%Author: Raul C. Muresan, 25.06.2007
%ATTENTION: The time stamps should be given in vectors in row format (values along the row, not
along the column)

clear all;

%Some example values for the parameters
iTrialLength = 2000;           %Suppose we have a sampling frequency of 1 kHz. We choose a trial length
of 2000 ms.
iCorrelationWindow = 100;     %We are interested in a cross correlogram of -100..100 ms.
iTrialNumber = 20;           %Suppose we have 20 trials
fFiringRate = 10;            %Suppose an average firing rate of 10 Hz => 20 Spikes per trial
iJitter = 5;                  %Produce two example spike trains such that the second one is a jittered
version of the first with this amount of uniform jitter

%Produce some random time stamps in the first neuron
iaTimeStampsNeuronA = sort(round(iTrialLength * iTrialNumber * rand(1,iTrialNumber * fFiringRate *
iTrialLength / 1000)),'ascend');

%Produce another random spike stamps by jittering the one before; the larger the jitter the smaller
the correlation
iaTimeStampsNeuronB = round(iaTimeStampsNeuronA + 2*iJitter*(0.5 -
rand(size(iaTimeStampsNeuronA))));

%Compute the cross correlation on spikes
%The function expects 5 parameters:
% 1. CorrelationWindow - Integer Number; Scalar - The size of the cross
correlation window (eg. 80 for a cross correlation with lags of -80..+80); use the same units as the
sampling of your time stamps
% 2. TrialLength - Integer Number; Scalar - The size of the trial in
sampling units
% 3. TimeStampsA - Vector of Integer Numbers: Time Stamps - The vector holding the
time stamps of the first neuron
% 4. TimeStampsB - Vector of Integer Numbers: Time Stamps - The vector holding the
time stamps of the second neuron
% 5. Normalize - Integer Number; Scalar - Set to 1 if you want to
normalize the correlogram or to 0 otherwise
% ATTENTION: the function returns NaN (Not a Number) values if the input buffers are empty

CrossCorrelogram =
ComputeClassicCC_BB(iCorrelationWindow,iTrialLength,iaTimeStampsNeuronA,iaTimeStampsNeuronB,0);

%Plot the classic cross-correlogram
x = -100:1:100;
bar(x,CrossCorrelogram,'r');

```



```

ReturnMatrix =
ComputeScaledCC_CC(iScaleSegmentSize,iCorrelationWindow,iTrialLength,daLFPsSamplesA,daLFPsSamplesB,0);

% Get the scaled cross-correlogram
daScaledCorrelogram = ReturnMatrix(1,:);

%Plot the scaled cross-correlogram
figure(1);
x = -iCorrelationWindow:1:iCorrelationWindow;
bar(x,daScaledCorrelogram,'r');

%
*****
*****
*****
%Option II: the trials are not one after another in the LFP vector, so we go trial by
trial %In this case, instead of averaging the scaled correlograms for each trial, we should sum up the
coefficients of correlation for each bin and then divide by their total count (see below)
% Averaging the scaled correlations is also possible but yields a bit more noisier results than the
method with the sums and counts of correlation coefficients

daCoefficientSums = zeros(1,2*iCorrelationWindow+1);
daCoefficientCounts = zeros(1,2*iCorrelationWindow+1);

%For each trial, do:
for trial=1:iTrialNumber

    %Compute the indexes where trials start and finish
    idxFrom = (trial-1)*iTrialLength + 1;
    idxTo = trial*iTrialLength;

    %Extract the trial from the two signals
    daTrialA = daLFPsSamplesA(idxFrom:idxTo);
    daTrialB = daLFPsSamplesB(idxFrom:idxTo);

    %Compute the scaled correlation on the trial
    %The function expects 6 parameters:
    %1. ScaleSegmentSize - Integer Number; Scalar - The size of the segment of interest (the scale
of the scaled correlation in sampling frequency units)
    %2. CorrelationWindow - Integer Number; Scalar - The half size of the cross correlation window
(eg. 80 for a cross correlation on the scale of -80..+80); use again the same units as the
sampling of your time stamps
    %3. TrialLength - Integer Number; Scalar - The size of the trial in units of your time stamps
    %4. SamplesA - Vector of Double Numbers: Samples - The vector holding the samples of the first
channel
    %5. SamplesB - Vector of Double Numbers: Samples - The vector holding the samples of the second
channel
    %6. UseFisherZTransform - Integer Number; Scalar - Set to 1 to use the Fisher Z transform to
average the coefficients of correlation; set to 0 for normal computation
    % ATTENTION: the function returns NaN (Not a Number) in a bin for which correlation could not
be defined (missing data, etc...)
    %Return values:
    %R = matrix(3,2*CorrW+1) - A return matrix with 3 rows and 2*piCorrelationWindow+1 columns. The
matrix contains:
    %- R(1,...) - The Scaled Cross Correlogram - Vector of Doubles A vector containing the cross
correlogram, having 2 * piCorrelationWindow + 1 elements; lag 0 is at position
piCorrelationWindow + 1 in Matlab
    %Returns NaN (Not a Number) values bins for which correlation could not be defined (missing
data, etc...)
    %- R(2,...) - The sum of the correlation coefficients for a given bin; contains NaN (Not a
Number) values bins for which correlation could not be defined (missing data, etc...).
    %- R(3,...) - The count of valid correlation coefficients for a given bin, that are used to
compute the scaled correlogram;
    %=> R(1,i) = R(2,i) / R(3,i);
    ReturnMatrix =
    ComputeScaledCC_CC(iScaleSegmentSize,iCorrelationWindow,iTrialLength,daTrialA,daTrialB,0);

    %Retrieve sum of coefficients and their counts and treat the Not a Number cases
    Sums = ReturnMatrix(2,:);
    Sums(isnan(Sums)) = 0; %Do not put NaNs to 0 when you average scaled correlograms (i.e.
ReturnMatrix(1,:)). Here it is safe because we have the real count of valid coefficients and the
average will be correct.
    Counts = ReturnMatrix(3,:);

```

```

    %Sum up coefficients and counts
    daCoefficientSums = daCoefficientSums + Sums;
    daCoefficientCounts = daCoefficientCounts + Counts;

end

%Compute the scaled correlogram by dividing the sum of correlation coefficients to their counts
daScaledCorrelogram = daCoefficientSums ./ daCoefficientCounts;

%Plot the scaled cross-correlogram
figure(2);
x = -iCorrelationWindow:1:iCorrelationWindow;
bar(x,daScaledCorrelogram,'r');

```

Scaled-correlation on a pair of binary-continuous signals (Spike-Triggered Average – STA):

```

%Scaled correlation analysis: Spike-Triggered Average (STA)
%Example of using the ComputeScaledSTA.dll library
%Author: Raul C. Muresan, 17.05.2007
%ATTENTION: If you supply the time stamps and the samples for a single trial, make sure that the
time stamps are relative to the beginning of the trial
%The time stamps and samples should be given in vectors in row format (values along the row, not
along the column)
%The sampling frequency of the continuous and binary signals must match

clear all;

%Some example values for the parameters
iTrialLength = 2000;           %Suppose we have a sampling frequency of 1 kHz. We choose a trial length
of 2000 ms.
iScaleWindow = 40;            %The size of the scale segment for the scaled correlation analysis
iCorrelationWindow = 100;     %We are interested in a scaled correlogram of -100..100 ms.
iTrialNumber = 20;           %Suppose we have 20 trials
fFiringRate = 10;            %Suppose an average firing rate of 10 Hz => 20 Spikes per trial

%Produce some random time stamps in the neuron
iaTimeStampsNeuronA = sort(round(iTrialLength * iTrialNumber * rand(1,iTrialNumber * fFiringRate *
iTrialLength / 1000)),'ascend');

%Produce an LFP
daLFPSamplesB = zeros(1,iTrialLength*iTrialNumber);
for i=1:iTrialLength*iTrialNumber;
    daLFPSamplesB(i) = sin(2*3.14/50*1000*i);
end

% We have two options to compute the scaled spike-triggered average.
% I. If the trials follow one after another in the spike and LFP vector then we can call only once
the function ComputeScaledSTA and it will return the scaled correlogram averaged over trials.
% II. If we do not have all the trials in the same spike and LFP vector then we would like to
compute the correlation separately for each trial and eventually average over them.

%
*****
*****
*****
%Option I: the trials are in the spike and LFP vector one after another in a continuous
sequence %Option II: the trials are in the spike and LFP vector one after another in a continuous
sequence
%Compute the spike triggered average
%The function expects 6 parameters:
%1. ScaleWindow - Integer Number; Scalar - The size of the scale segment window used in scaled
correlation analysis
%2. CorrelationWindow - Integer Number; Scalar - The size of the correlation window (eg. 80 for a
scaled correlation with lags of -80..+80); use the same units as the sampling of your time stamps
and samples
%3. TrialLength - Integer Number; Scalar - The size of the trial in sampling units
%4. TimeStampsA - Vector of Integer Numbers: Time Stamps - The vector holding the time stamps of
events in the binary signal
%5. SamplesB - Vector of Double Numbers: Samples - The vector holding the samples of the continuous
signal
%6. UseFisherZTransform - Integer Number; Scalar - Set to 1 if you want to use the Fisher's Z
Transform when the correlation coefficients are averaged
%ATTENTION: the function returns NaN (Not a Number) values if there is no matching time stamp of an
event in the binary signal for a given scale segment

```

```

%Return values:
%R = matrix(3,2*CorrW+1) - A return matrix with 3 rows and 2*piCorrelationWindow+1 columns. The
matrix contains:
% - R(1,...) - The Scaled Cross Correlogram - Vector of Doubles A vector containing the cross
correlogram, having 2 * piCorrelationWindow + 1 elements; lag 0 is at position piCorrelationWindow +
1 in Matlab
% Returns NaN (Not a Number) values bins for which correlation could not be defined (missing data,
etc...)
% - R(2,...) - The sum of the correlation coefficients for a given bin; contains NaN (Not a Number)
values bins for which correlation could not be defined (missing data, etc...).
% - R(3,...) - The count of valid correlation coefficients for a given bin, that are used to compute
the scaled correlogram;
% => R(1,i) = R(2,i) / R(3,i);

ReturnMatrix =
ComputeScaledSTA(iScaleWindow,iCorrelationWindow,iTrialLength,iaTimeStampsNeuronA,daLFPSamplesB,0);

% Get the scaled STA
daSTA = ReturnMatrix(1,:);

%Plot the scaled STA correlogram
figure(1);
x = -100:1:100;
bar(x,daSTA,'r');

%
*****
*****
*****
%Option II: We want to go trial by trial and compute the correlogram on each
%
% In this case, instead of averaging the scaled correlograms for each trial, we should sum up the
coefficients of correlation for each bin and then divide by their total count (see below)
% Averaging the scaled correlations is also possible but yields a bit more noisier results than the
method with the sums and counts of correlation coefficients

daCoefficientSums = zeros(1,2*iCorrelationWindow+1);
daCoefficientCounts = zeros(1,2*iCorrelationWindow+1);

%For each trial, do:
for trial=1:iTrialNumber

    %Compute the indexes where trial for the continuous variable starts and finishes
    idxContFrom = (trial-1)*iTrialLength + 1;
    idxContTo = trial*iTrialLength;

    %Extract the trial from continuous signal
    daTrialContinuous = daLFPSamplesB(idxContFrom:idxContTo);

    %Compute the indexes where the trial in the time stamp vector (time stamps of events from the
binary signal) starts and ends
    idxBinaryFrom = find(iaTimeStampsNeuronA >= (trial-1)*iTrialLength,1,'first');
    idxBinaryTo = find(iaTimeStampsNeuronA < trial*iTrialLength,1,'last');

    %Extract the trial from the time stamp vector
    daTrialBinary = iaTimeStampsNeuronA(idxBinaryFrom:idxBinaryTo);

    %Align the time stamps to the beginning of the trial
    daTrialBinary = daTrialBinary - (trial-1)*iTrialLength;

    %Compute the spike triggered average on the trial
    %The function expects 6 parameters:
    % 1. ScaleWindow - Integer Number; Scalar - The size of the scale
segment window used in scaled correlation analysis
    % 2. CorrelationWindow - Integer Number; Scalar - The size of the
correlation window (eg. 80 for a scaled correlation with lags of -80..+80); use the same units
as the sampling of your time stamps and samples
    % 3. TrialLength - Integer Number; Scalar - The size of the trial
in sampling units
    % 4. TimeStampsA - Vector of Integer Numbers: Time Stamps - The vector holding the
time stamps of events in the binary signal
    % 5. SamplesB - Vector of Double Numbers: Samples - The vector holding the
samples of the continuous valued signal
    % 6. UseFisherZTransform - Integer Number; Scalar - Set to 1 if you want
to use the Fisher's Z Transform when the correlation coefficients are averaged

```

```

% ATTENTION: the function returns NaN (Not a Number) values if there is no matching time stamp
for a given scale segment
%Return values:
% R = matrix(3,2*CorrW+1) - A return matrix with 3 rows and 2*piCorrelationWindow+1 columns.
The matrix contains:
%
% - R(1,...) - The Scaled Cross Correlogram - Vector of Doubles A
vector containing the cross correlogram, having 2 * piCorrelationWindow + 1 elements; lag 0 is
at position piCorrelationWindow + 1 in Matlab
%
% Returns NaN (Not a Number) values bins for which correlation
could not be defined (missing data, etc...)
%
% - R(2,...) - The sum of the correlation coefficients for a given
bin; contains NaN (Not a Number) values bins for which correlation could not be defined (missing
data, etc...).
%
% - R(3,...) - The count of valid correlation coefficients for a
given bin, that are used to compute the scaled correlogram;
%
% => R(1,i) = R(2,i) / R(3,i);
ReturnMatrix =
ComputeScaledSTA(iScaleWindow,iCorrelationWindow,iTrialLength,daTrialBinary,daTrialContinuous,0)
;

%Retrieve sum of coefficients and their counts and treat the Not a Number cases
Sums = ReturnMatrix(2,:);
Sums(isnan(Sums)) = 0; %Do not put NaNs to 0 when you average Scaled Correlograms (i.e.
ReturnMatrix(1,:)). Here it is safe because we have the real count of valid coefficients and the
average will be correct.
Counts = ReturnMatrix(3,:);

%Sum up coefficients and counts
daCoefficientSums = daCoefficientSums + Sums;
daCoefficientCounts = daCoefficientCounts + Counts;
end

%Compute the scaled correlogram by dividing the sum of correlation coefficients to their counts
daSTA = daCoefficientSums ./ daCoefficientCounts;

%Plot the scaled cross-correlogram
figure(2);
x = -iCorrelationWindow:1:iCorrelationWindow;
bar(x,daSTA,'r');

```

Scaled-correlation on a pair of binary-binary signals:

```

%Scaled correlation analysis: Scaled Correlation for Binary - Binary signals
%Example of using the ComputeScaledCC_BB.dll library
%Author: Raul C. Muresan, 25.06.2007
%ATTENTION: The time stamps should be given in vectors in row format (values along the row, not
along the column)

clear all;

%Some example values for the parameters
iTrialLength = 2000; %Suppose we have a sampling frequency of 1 kHz. We choose a trial length
of 2000 ms.
iCorrelationWindow = 100; %We are interested in a cross correlogram of -100..100 ms.
iScaleSegmentSize = 40; %40 ms for example, to extract synchrony in the gamma frequency range
iTrialNumber = 20; %Suppose we have 20 trials
fFiringRate = 10; %Suppose an average firing rate of 10 Hz => 20 Spikes per trial
iJitter = 5; %Produce two example spike trains such that the second one is a jittered
version of the first with this amount of uniform jitter

%Produce some random time stamps in the first neuron
iaTimeStampsNeuronA = sort(round(iTrialLength * iTrialNumber * rand(1,iTrialNumber * fFiringRate *
iTrialLength / 1000)), 'ascend');

%Produce another random spike stamps by jittering the one before; the larger the jitter the smaller
the correlation
iaTimeStampsNeuronB = round(iaTimeStampsNeuronA + 2*iJitter*(0.5 -
rand(size(iaTimeStampsNeuronA))));

% We have two options to compute the scaled correlogram.
% I. If the trials follow one after another in the LFP vector then we can call only once the
function ComputeScaledCC_CC and it will return the scaled correlogram averaged over trials.
% II. If we do not have all the trials in the same LFP vector, we would like to compute the
correlation separately for each trial and then average.

```

```

%
*****
*****
*****
%%%%%%%%%% Option I: the trials are in the LFP vector one after another in a continuous sequence
%%%%%%%%%%
%Compute the scaled correlation on spikes
%The function expects 6 parameters:
% 1. ScaleSegmentSize - Integer Number; Scalar - The size of the segment of
interest (the scale of the scaled correlation in sampling frequency units)
% 2. CorrelationWindow - Integer Number; Scalar - The half size of the cross
correlation window (eg. 80 for a cross correlation on the scale of -80..+80); use again the same
units as the sampling of your time stamps
% 3. TrialLength - Integer Number; Scalar - The size of the trial in
units of your time stamps
% 4. TimeStampsA - Vector of Integer Numbers: Time Stamps - The vector holding the
time stamps of the first neuron
% 5. TimeStampsB - Vector of Integer Numbers: Time Stamps - The vector holding the
time stamps of the second neuron
% 6. UseFisherZTransform - Integer Number; Scalar - Set to 1 to use the Fisher
Z transform to average the coefficients of correlation; set to 0 for normal computation
% ATTENTION: the function returns NaN (Not a Number) in a bin for which there was no coincidence
at all
%Return values:
% R = matrix(3,2*CorrW+1) - A return matrix with 3 rows and 2*piCorrelationWindow+1 columns. The
matrix contains:
% - R(1,...) - The Scaled Cross Correlogram - Vector of Doubles A
vector containing the cross correlogram, having 2 * piCorrelationWindow + 1 elements; lag 0 is at
position piCorrelationWindow + 1 in Matlab
% Returns NaN (Not a Number) values bins for which correlation could
not be defined (missing data, etc...)
% - R(2,...) - The sum of the correlation coefficients for a given
bin; contains NaN (Not a Number) values bins for which correlation could not be defined (missing
data, etc...).
% - R(3,...) - The count of valid correlation coefficients for a given
bin, that are used to compute the scaled correlogram;
% => R(1,i) = R(2,i) / R(3,i);

ReturnMatrix =
ComputeScaledCC_BB(iScaleSegmentSize,iCorrelationWindow,iTrialLength,iaTimeStampsNeuronA,iaTimeStampsNeuronB,0);
daScaledCorrelogram = ReturnMatrix(1,:);

%Plot the scaled cross-correlogram
figure(1);
x = -iCorrelationWindow:1:iCorrelationWindow;
bar(x,daScaledCorrelogram,'r');

%
*****
*****
*****
%%%%%%%%%% Option II: the trials are not one after another in the LFP vector, so we go trial by
trial %%%%%%%%%%%
% In this case, instead of averaging the scaled correlograms for each trial, we should sum up the
coefficients of correlation for each bin and then divide by their total count (see below)
% Averaging the scaled correlations is also possible but yields a bit more noisier results than the
method with the sums and counts of correlation coefficients

daCoefficientSums = zeros(1,2*iCorrelationWindow+1);
daCoefficientCounts = zeros(1,2*iCorrelationWindow+1);

%For each trial, do:
for trial=1:iTrialNumber
%Compute the indexes where the trial in the first binary signal (timestamp vector) starts and
ends
idxBinaryAFrom = find(iaTimeStampsNeuronA >= (trial-1)*iTrialLength,1,'first');
idxBinaryATo = find(iaTimeStampsNeuronA < trial*iTrialLength,1,'last');

%Extract the trial from the first timestamp vector
daTrialBinaryA = iaTimeStampsNeuronA(idxBinaryAFrom:idxBinaryATo);

%Compute the indexes where the trial in the second binary signal (timestamp vector) starts and
ends
idxBinaryBFrom = find(iaTimeStampsNeuronB >= (trial-1)*iTrialLength,1,'first');
idxBinaryBTo = find(iaTimeStampsNeuronB < trial*iTrialLength-1,1,'last');

```

```

%Extract the trial from the second timestamp vector
daTrialBinaryB = iaTimeStampsNeuronB(idxBinaryBFrom:idxBinaryBTo);

%Align the time stamps to the beginning of the trial
daTrialBinaryA = daTrialBinaryA - (trial-1)*iTrialLength;
daTrialBinaryB = daTrialBinaryB - (trial-1)*iTrialLength;

%Compute the spike triggered average on the trial
%Compute the scaled correlation on spikes
%The function expects 6 parameters:
% 1. ScaleSegmentSize - Integer Number; Scalar - The size of the
segment of interest (the scale of the scaled correlation in sampling frequency units)
% 2. CorrelationWindow - Integer Number; Scalar - The half size of the
cross correlation window (eg. 80 for a cross correlation on the scale of -80..+80); use again
the same units as the sampling of your time stamps
% 3. TrialLength - Integer Number; Scalar - The size of the trial
in units of your time stamps
% 4. TimeStampsA - Vector of Integer Numbers: Time Stamps - The vector holding the
time stamps of the first neuron
% 5. TimeStampsB - Vector of Integer Numbers: Time Stamps - The vector holding the
time stamps of the second neuron
% 6. UseFisherZTransform - Integer Number; Scalar - Set to 1 to use the
Fisher Z transform to average the coefficients of correlation; set to 0 for normal computation
% ATTENTION: the function returns NaN (Not a Number) in a bin for which there was no
coincidence at all
%Return values:
% R = matrix(3,2*CorrW+1) - A return matrix with 3 rows and 2*piCorrelationWindow+1 columns.
The matrix contains:
% - R(1,...) - The Scaled Cross Correlogram - Vector of Doubles A
vector containing the cross correlogram, having 2 * piCorrelationWindow + 1 elements; lag 0 is
at position piCorrelationWindow + 1 in Matlab
% Returns NaN (Not a Number) values bins for which correlation
could not be defined (missing data, etc...)
% - R(2,...) - The sum of the correlation coefficients for a given
bin; contains NaN (Not a Number) values bins for which correlation could not be defined (missing
data, etc...).
% - R(3,...) - The count of valid correlation coefficients for a
given bin, that are used to compute the scaled correlogram;
% => R(1,i) = R(2,i) / R(3,i);
ReturnMatrix =
ComputeScaledCC_BB(iScaleSegmentSize,iCorrelationWindow,iTrialLength,daTrialBinaryA,daTrialBinaryB,0);

%Retrieve sum of coefficients and their counts and treat the Not a Number cases
Sums = ReturnMatrix(2,:);
Sums(isnan(Sums)) = 0; %Do not put NaNs to 0 when you average Scaled Correlograms (i.e.
ReturnMatrix(1,:)). Here it is safe because we have the real count of valid coefficients and the
average will be correct.
Counts = ReturnMatrix(3,:);

%Sum up coefficients and counts
daCoefficientSums = daCoefficientSums + Sums;
daCoefficientCounts = daCoefficientCounts + Counts;
end

%Compute the scaled correlogram by dividing the sum of correlation coefficients to their counts
daScaledCorrelogram = daCoefficientSums ./ daCoefficientCounts;

%Plot the scaled correlogram
figure(2);
x = -iCorrelationWindow:1:iCorrelationWindow;
bar(x,daScaledCorrelogram,'r');

```

Release note:

The code is free for non commercial purposes. You may use it freely, with the sole restriction that you may not claim that you wrote it. I do not warrant that the code is 100% bug free. Use at your own risk!