Teaching Modules in Computational Neuroscience 1 (2012)
The German Neuroinformatics Node – www.g-node.org

Course Module

Analysis of the phase relation of spikes to the LFP

Dr. Michael Denker

Institute of Neuroscience and Medicine (INM-6), Forschungszentrum Juelich, Germany [m.denker@fz-juelich.de]

Synopsis

In neuronal recordings from the brain, spiking activity is often measured in parallel to mass signals such as the local field potential (LFP, the low-pass filtered electrode signal). It is assumed that biophysically LFPs primarily result from the synaptic activity in neurons of a large volume near the electrode, and that its oscillatory components originate in the synchronized activation of synaptic inputs. A common observation is that spikes occur at a preferred phase of the LFP oscillation cycle. Knowledge of the degree of locking between spikes and LFP helps to disambiguate neuronal activity of neurons that is entrained to the population signal from that which is not.

In this module, we focus to familiarize students with both long-established and more recent methods that quantify phase and amplitude measurements from LFP signals. In particular, we apply these techniques to parallel recordings of LFP signals and spike sequences. The methods include spike-triggered averages (STA), the normalized STA power spectrum (spike-field coherence), phase and amplitude extraction, and phase synchronization measures. The exercises are intended to acquaint students with some of these measures using simulated data, and provide hands-on training using a selected set of electrophysiological recordings.

Supplemental Material

With this course module we provide an Introductory Lecture "*Analysis of the phase relation of spikes to the LFP*" and additional data files for practical analysis. Data Courtesy of *Dr. Alexa Riehle, Institut de Neurosciences de la Timone (INT), UMR 7289, CNRS - Aix Marseille Univ., Marseille, France.*

The supplemental material for this teaching module is available online at the following URL: https://portal.g-node.org/dataanalysis-course-2012/doku.php

Requirements

Practical work on this course module requires that you run Matlab Version 7 or higher, the Matlab Signal Processing Toolbox and the Matlab Statistics Toolbox. Parts of the exercises require the chronux toolbox (available at http://chronux.org/) developed by the Mitra Lab at Cold Spring Harbor Laboratory (see also: "Observed Brain Dynamics", Partha Mitra and Hemant Bokil, Oxford University Press, New York, 2008). Supplemental data files are required for data analysis.

Introduction

This exercise is composed of two main tasks. The first task is intended to provide hands-on experience regarding three analysis techniques for spike-LFP relationships introduced in the accompanying lecture. The techniques are partly implemented by the students, and tested on simple artificial data. A series of exercises shows some limits of each of these techniques under extreme circumstances. The second task guides students through the analysis of a biological data set. The focus rests on the detection of phase locking between spikes and LFP in a transient fashion, introducing a simple surrogate technique to construct the null hypothesis.

Students are not expected to finish the complete list of exercises: most tasks are formulated in an open fashion that leaves room for students to explore the various analysis scenarios suggested in as much depth as they prefer. It is highly suggested that before anything, all students perform the first exercise 1.1, as it introduces the different analysis techniques. After completing 1.1, students may choose to focus more on the rest of Task 1 or on Task 2. The remaining exercises in Task 1 are largely independent of each other and could be completed in any order. In contrast, the exercises of Task 2 should be completed in order.

The following functions are provided to accelerate the exercises:

- *create_sine*: create a sine wave time series
- *create_poiss*: create a spike train modeled as a Poisson process
- *create_locked*: create a spike train that is phase-locked to the LFP with a given precision
- *convert2bin*: convert a spike train given by spike time stamps into a binned spike train
- *cs_mean*: circular mean of a vector
- *cs_std*: circular std of a vector
- *cs_R*: normalized vector strength
- *cs_test_uniform*: circular Rayleigh test for uniformity (corrected for small samples)
- *vonMise*: a normalized von Mise distribution
- *create_isi_surrogates*: create a spike train by shuffling the inter-spike intervals of an existing one

From the publicly available *chronux* toolbox the following functions are used:

- *sta: calculates a mean-corrected spike-triggered average*
- *coherencycpt: calculates the coherence between a continuous signal and a point process represented by time stamps*
- *coherencycpb: the coherence between a continuous signal and a point process represented by a binary sequence*

Task 1: Assessment of the sensitivity and robustness of different spike-LFP measures

In this first task, we will become acquainted with different measures that relate spikes to the LFP. In particular, we will study the spike-triggered average (STA), the spike-field coherence (SFC), and spike-LFP phase locking (PL) by means of artificially created data. In this data we are able to simulate different scenarios of how a spike train may phase lock to an LFP oscillation. To this end, spike trains are modeled either as Poisson (or Gamma) processes or as spikes that are locked to the LFP, whereas LFPs are modeled as sine waves.

1.1) In this section, we implement the three methods STA, SFC and PL and test them on a simple surrogate data.

1.1a) Define a sampling rate (e.g., 1000 Hz), the sampling period (1/sampling rate), and a time length *T* for all simulations (e.g., T=5 s). We first create a Poisson spike train (represented by a vector that contains the spike times in ms) and a sinusoidal LFP. We use reasonable parameters, for instance you might choose a spike rate of 10 Hz and an oscillation at 10 Hz, over a length of *T*. You will find the functions *create_poiss()* and *create_sine()* useful for this task. In the following exercises, you may want to experiment with these parameters.

1.1b) Calculate the spike-triggered average of the data created in 1.1a. There are three options of doing this.

The straight-forward approach is to cut the LFP in a window of +-*s* ms around time points of spiking, and then average the cut time segments (each segment thus has length *2s+1*). If you chose 10 Hz as the LFP oscillation frequency, a good choice is *s*=200 ms since this choice includes about 4 periods. Be sure not to include spikes that occur right on the border (within s ms) of the available data segment!

A second alternative is to convert the spike train into a binned data structure (e.g., using *convert2bin()*). Here, spikes are represented by a binary vector, each of its entries is a time bin (e.g., binned on 1 ms) and a value of 1 denotes a spike whereas 0 denotes no spike. We may then calculate the cross-correlation of the binned process and the LFP signal. An efficient way involves using the Matlab function *xcov()* (passing as parameters the width of the cut segments s as lags, and 'none' as normalization). Be careful of the order of arguments when using *xcov()*: pass the LFP signal as first parameter, and the spike train as second parameter (otherwise, the resulting STA will be flipped around the y-axis). We normalize the result by the number of spikes.

A third option involves using the *sta()* function of the *chronux* toolbox (in contrast to the above methods, it provides an STA that has its mean subtracted).

To calculate the confidence of the observed STA, we create 1000 STAs constructed from spike trains in which spikes are randomly shuffled in time (since we are dealing with stationary Poisson spike trains here, a simple shuffling procedure is adequate). This shuffling is best achieved on the binned (0-1) spike train, where we randomly permute the entries of the vector (the Matlab function *randperm()* might be helpful here). Once all 1000 STAs are obtained, at each time lag, extract the 25th smallest and 25th largest value in the surrogate distribution of values of the STA at that lag (results in 5% confidence, two-tailed). Plotting these two values at each lag give an estimate of the size of the STA when potential spike-LFP correlations are destroyed. These confidence bands may be used in subsequent analysis steps. In a a real situation, more sophisticated surrogates need to be applied.

1.1c) Calculate the spike-field coherence of the signal. We will use the function *coherencycpb()* (for spikes given as binned data) / *coherencycpt()* (for spikes given as time stamps) of the *chronux* toolbox. Be mindful of the fact that this function takes data as column vectors (each column represents one trial). This function will return the calculated spike-field coherence in a frequency-resolved manner, as well as the phase of coherence, the individual cross-spectra that enter the coherence, and the confidence level at each frequency (at a chosen significance level alpha). *Technical note*: When calling these functions, you should receive all return values `[C,phi,S12,S1,S2,f,zerosp,confC,phistd]` to avoid errors. To obtain a frequency axis and confidence limits at an alpha level of alpha, create a structure

```
cparam.err=[1 alpha];
cparam.Fs=samplingrate;
```

and pass this structure to *coherencycpb()* or *coherencycpt()*.

1.1d) Finally, we calculate the spike-triggered phase histogram using a direct decomposition of the signal into phase and amplitude. To this end, calculate the analytic signal $z(t)$ via the Hilbert transform $H[.]$ of the LFP series using the *hilbert()* Matlab function (despite its name, this function directly calculates the analytic signal, not the Hilbert transformation). Using the Matlab functions *angle* and *abs*, we extract the phase $\phi(t)$ and amplitude $A(t)$, respectively, of the oscillation of the returned analytic signal $z(t)=x(t)+i\, H[x(t)]=A(t)\, exp(i\, \phi(t))$.

We may now backtrack to obtain the Hilbert transform as the imaginary part of the analytic signal (*imag()*). Visualize the simulated LFP signal, its Hilbert transform, the analytic signal and the phase as a function of time. Note: *angle()* returns phase angles between $-\pi$ and π. What is the phase 0 on the original signal? What is phase 0 on the Hilbert signal? In the following we stick to this definition of phase.

Next, we calculate the phase and amplitude at the spike times. Plot the phase and amplitude distributions. Using *cs_mean()*, *cs_std()*, *cs_R()*, and *cs_test_uniform()*, calculate the mean phase, its standard deviation sigma, the vector strength R, and test whether the phase distribution is uniform.

Side note: If you want to be very precise in the calculation of the spike-triggered phase when a spike occurs between two sample time points t_i and t_{i+1} for which the phase $\phi(t)$ of $x(t)$ was calculated, you might want to circularly interpolate the phase between these two neighboring bins:

$\phi(t)=arg\ (\ exp(\ i\ *(\ \phi(t_i)\ +\ (t-t_i)/(t_{i+1}-t_i)\ *\ (\phi(t_{i+1})-\phi(t_i))\ )\ )\ )$

1.2) In the following sections, we test how well the previously established methods deal with various types of artificial data. How well can we detect spike-LFP locking with decreasing precision *p* of the phase locking? Here and in the following, *p* defines the window on the phase cycle in which a locked spike may be located: $-p..+p$. Thus, $p=\pi$ is equivalent to no locking, $p=0$ represents perfect locking.

Create a locked spike train with various degrees of locking precision to our regular (sine) LFP. To this end, you may use the function *create_locked()*, which takes the LFP input signal as an argument and returns an array of *N* locked spikes at a predefined range of phases of each oscillatory cycle, ϕ in *[WindowStart..WindowStop]* (e.g., p=-0.1..+0.1). Note: This simple routine will not be able to place spikes if the allowed window *WindowStart..WindowStop* is to small compared to the sampling period (i.e. sampling rate) of the LFP! Test the three previously developed tools and visualize the data. How sensitive are they under changing the parameter *p* of the locked spike train (i.e, the width of the window in which the locked spike may occur)?

1.3) Assume that only part of the spikes are locked. For example, you may imagine that spikes originate in the activity of several neurons (recorded multi-unit activity), and only some of the contributing neurons are locked. How well can we detect locking if only part of the spikes are locked? We create compound spike trains in which only a fraction *f* of the spikes is perfectly locked to the oscillation (created with *create_locked()*, using parameter *p*=0.1 (see 1.2)), the remaining fraction of *1-f* of spikes obeys a Poisson process (created with *create_poiss()*). To accomplish this, you could simulate both spike trains, and randomly pick the corresponding fractions of spikes from each process. Evaluate how sensitive each measure can single out the existence of locking in the population using the test for uniformity, *cs_test_uniform()*.

1.4) Using SFC is related to the STA via its Fourier transform. Because of its normalization with the average power of any frequency, the SFC detects spikes even if they are locked to frequencies that are shadowed by large amplitude oscillatory components in the signal. However, the SFC still weights spikes according to the instantaneous amplitude of the respective LFP component at which the spikes occur. In contrast, PL disambiguates amplitudes from phases on a spike-by-spike basis.

To see this effect, construct a data set composed of two segments: The first half has on average a low amplitude LFP and spikes locked with a precision of *p*=0.1 (see 1.2), the second half has a high average amplitude LFP and spikes are modeled as Poisson. Hint: To create the LFP and ensure a smooth phase throughout, let us model the LFP as a sine with linarly increasing amplitude. For example, you may want to create a sine wave with (small) amplitude A_1 and then linearly ramp this amplitude up to a higher value A_2 as time increases:

```
temp=create_sine(timelength,samplingrate,lfp_freq,A1,0);
lfp_mixamp=temp.*((0:timelength/samplingperiod)*...
  (A2/A1-1)/timelength*samplingperiod + 1);
```

As you calculate the SFC (coherence, coherence phase, and cross spectra) and PL measures (vector strength, mean phase, test for uniformity), vary the amplitude ratio A_1/A_2 of the LFP between the first and second half of the data segment. You will notice that the SFC measure is sensitive to this variational change, whereas the values for the vector strength of the phase distribution obtained for PL remain relatively constant. How sensitive is this analysis to the various parameters?

Extra-1.5) Assume a gamma spike train of rate *f* and order *N* by creating a Poisson spike train of rate *N*f* and keeping only every *N*th spike. Does the regularity of the process induce locking?

Task 2: Analysis of spike-LFP phase locking in recordings from motor cortex

In this second task we will work with data recorded from motor cortex of a behaving monkey in a time estimation task. We will look at one specific experimental condition during which the LFP activity was recorded in parallel to one spiking neuron. The data comprises 1200 ms during the delay period of the task - the period between a preparatory stimulus and the signal indicating the movement start. We will try to detect whether a recorded neuron shows genuine, transient locking to the LFP.

2.1) Load the data files *spikes.mat* and *lfp.mat*. The former contains a cell array of *N* dimensions *spike_cell*, each containing the spike times within a particular trial. The latter contains an *NxM* matrix *lfp_matrix*, where each row contains the LFP time series in a given trial, and a 1xM vector *time* that corresponds to the time stamps in ms of each column in the LFP matrix. The variable *sf* contains the sampling frequency of each trace in Hz. Plot the spike data (spike raster) and the LFP traces, and compare them. For the LFP, it is helpful for visualization to normalize amplitudes using a z-transform: subtract the mean from each waveform, and divide by the standard deviation. Describe the data by eye: Are there non-stationarities? Are there regularities? Which oscillations can you detect in the LFP, and do they occur transiently?

2.2) Calculate the inter-spike interval distribution of the spike data, i.e., the distribution of time differences between consecutive spikes $d=t_{i+1}-t_i$. Compare the ISI distribution to one expected for a regular, i.e., Poisson spike train. The waiting time distribution of such a process is given as $p(t)=exp(-t\lambda)$, where λ is the rate of the process. By roughly comparing theoretical vs. measured ISI distributions by eye (at various λ close to the spike rate observed in the data), what conclusions may we draw considering we want to analyze phase locking in this data set? Where do you see potential problems in the analysis of spikes and LFP based on the ISI?

2.3) As in task 1.1d, we now calculate the overall degree of phase locking to the data using the PL approach. However, in this case, the signal contains a mixture of various frequency components. Thus, we need to pre-filter the LFP traces. Coming back to the traces visualized in 2.1, we notice a clear oscillatory structure. Construct a power spectrum of the data, for example using the Matlab function *pwelch()*. Supply this function with the sampling frequency in order to obtain a frequency axis. Do this separately for each trial, and average the individual spectra. What is the preferred frequency of oscillation? If you are interested, you might want to check the temporal dynamics of the oscillation using a spectrogram (e.g., using *spectrogram.m*)

We use a Butterworth filter to filter the data. To this end, use the *butter()* function of Matlab to obtain a set of filter coefficients. The order of the filter (number of coefficients) should not be chosen above 10. Implement a bandpass filter between *a* Hz and *b* Hz based on the above power spectral analysis, using

```
[a b]*2/sf
```

as argument for the pass band to the *butter()* function. Butterworth filters are part of a class of filters (IIR) that does not preserve phase information. Therefore, we must be careful in applying an appropriate filter technique that corrects for the phase distortions of the filter, i.e., a filter algorithm that is phase neutral. A good way to achieve such neutrality this is to use the Matlab function *filtfilt()* instead of *filter().* The former implements a zero-phase filtering technique by applying the filter a second time in a time-reversed manner. Both filter routines take the coefficients returned by *butter()* as argument.

Plot the resulting *filtfilt()* filtered LFP traces. Compare these to the unfiltered LFPs and to LFPs filtered using the *filter()* function and estimate the phase lag introduced by filter(). Try various pass filters based on the power spectrum: what would be a good choice?

2.4) We now calculate the phases for each trial of the filtered LFP, and determine the spike-triggered phases in analogy to 1.1d. Calculate the histogram of spike phases (pooled across trials) and normalize the result by the number of spikes (such that the histogram has unit area). Plot the resulting phase histogram, using a phase axis from 0 to 2π binned at, e.g., 25 bins. Repeat the analysis with the unfiltered band and compare the results.

To estimate the parameters of the histogram (filtered LFP), we will fit a von Mises distribution to the data. The functional form of the von-Mises distribution is given by *vonMise()*. The fitting is performed using the Matlab function *nlinfit()*: It requires a handle to the fit function (`@vonMise`), the binned phase axis, the calculated phase histogram, and starting values for the parameters *κ* and *a* (these two are passed to *vonMise()* in a two element vector A). Using the parameters found by *nlinfit()*, you can use *nlpredci()* to obtain the best fit estimate. Is the distribution uniform (test for uniformity, *cs_test_uniform()*)?

2.5) As we have seen in 2.2, there may be a certain degree of regularity in the data that might explain any observed locking. To test for this hypothesis (in a very ad-hoc fashion), we utilize a surrogate approach using inter-spike interval shuffles. To this end, create 1000 surrogates. For each surrogate, we shuffle the inter-spike intervals of each trial, assigning a random starting position for the first spike in each trial. This procedure is performed by the function *create_isi_surrogate()*, which creates one set of shuffled spikes from the input data cell *spike_cell*. From the surrogate data, calculate the 1000 vector strengths *R*. Plot the distribution of R obtained from all surrogates. Compare the value of R measured from the actual data to this distribution - does it exceed the 95% limit (950th highest value of R obtained from the surrogates)?

2.6) Compare the cumulative distribution (CDF) of R obtained via surrogates in 2.5 to the one obtained for a pure Poisson spike train. The surrogate cumulative distribution is given as *p(x)={*fraction of surrogates with *R*>=*x}*. Theoretically, for a Poisson process we know that the quantity $2R^2N$ is distributed as X^2 distribution with 2 degrees of freedom, where *R* is the vector sum and *N* is the number of samples (i.e., spikes) involved in the computation. Thus, the cumulative distribution for Poisson trains can be obtained using the *chi2cdf()* function and $2R^2N$ as argument. Compare the two CDFs from the surrogate and theory: Which method is more conservative? Based on this ad-hoc analysis, is there evidence that regularity may cause spurious locking?

Extra-2.7) In a next step, we test whether the degree of spike-LFP locking changes dynamically within the experiment. Calculate the phase locking using spikes across trials and in overlapping sliding windows of 200 ms. From each time window, extract the mean phase phi and the vector strength R. Plot the time course of these two variables. At which times can you detect locking beyond chance (using the test for uniformity, *cs_test_uniform()*)? Compare the time-resolved analysis with an SFC analysis using the *cohgramcpt()* function of the *chronux* toolbox. Would we have detected the locking using this related analysis technique?

Acknowledgements

We thank Dr. Alexa Riehle for providing the data sets used in the exercises, and the developers of the *chronux* toolbox for providing their code to the scientific community.