

Neural Signal Processing: Tutorial 1

Keith P. Purpura, PhD¹ and Hemant Bokil, PhD²

¹Department of Neurology and Neuroscience, Weill Cornell Medical College
New York, New York

²Cold Spring Harbor Laboratory
Cold Spring Harbor, New York

Introduction

In this chapter, we will work through a number of examples of analysis that are inspired in part by a few of the problems introduced in “Spectral Analysis for Neural Signals.” Our purpose here is to introduce and demonstrate ways to apply the Chronux toolbox to these problems. The methods presented here exemplify both univariate analysis (techniques restricted to signals elaborated over a single time course) and bivariate analysis, in which the goal is to investigate relationships between two time series. Problems involving more than two time series, or a time series combined with functions of spatial coordinates, are problems for multivariate analysis. The chapters “Multivariate Neural Data Sets: Image Time Series, Allen Brain Atlas” and “Optical Imaging Analysis for Neural Signal Processing: A Tutorial” deal explicitly with these techniques and the use of the Chronux toolbox to solve these problems.

“Spectral Analysis for Neural Signals” introduces the spectral analysis of single-unit recordings (spikes) and continuous processes, for example, local field potentials (LFPs). As shown in that chapter, the multitaper approach allows the researcher to compute and render graphically several descriptions of the dynamics present in most electrophysiological data. Neural activity from the lateral intraparietal area (LIP) of the alert monkey was used to calculate LFP spectrograms and spike-LFP coherograms and, most importantly, measures of the reliability of these estimates. Although the computations required to produce these descriptions are easily attainable using today’s technology, the steps required to achieve meaningful and reliable estimates of neural dynamics need to be carefully orchestrated. Chronux provides a comprehensive set of tools that organize these steps with a set of succinct and transparent MATLAB scripts. Chronux analysis software also clears up much of the confusion surrounding which of the parameters that control these calculations are crucial, and what values these parameters should take, given the nature of the data and the goals of the analysis.

Chronux is downloadable from www.chronux.org. This software package can process both univariate and multivariate time series data, and these signals can be either continuous (e.g., LFP) or point process data (e.g., spikes). Chronux can handle a number of signal modalities, including electrophysiological and optical recording data. The Chronux release includes a spike-sorting toolbox and extensive online and within-MATLAB help documentation. Chronux also includes scripts that can translate files generated by NeuroExplorer (Nex Technologies, Littleton, MA) (.NEX), and the time-stamped (.PLX) and

streamed (.DDT) data records collected with Plexon (Dallas, TX) equipment.

We will proceed through components of a standard electrophysiology analysis protocol in order to illustrate some of the tools available in Chronux. Figure 1 charts the basic steps required for handling most electrophysiological data. We will assume that an editing procedure has been used to sort the data into continuous signals (LFPs or EEGs) and spikes. We will also advance to a stage that follows both the spike sorting and data conditioning steps (detrending and removing artifacts, including 60 Hz line noise). We will return to the detrending and 60 Hz line noise problems later in the chapter.

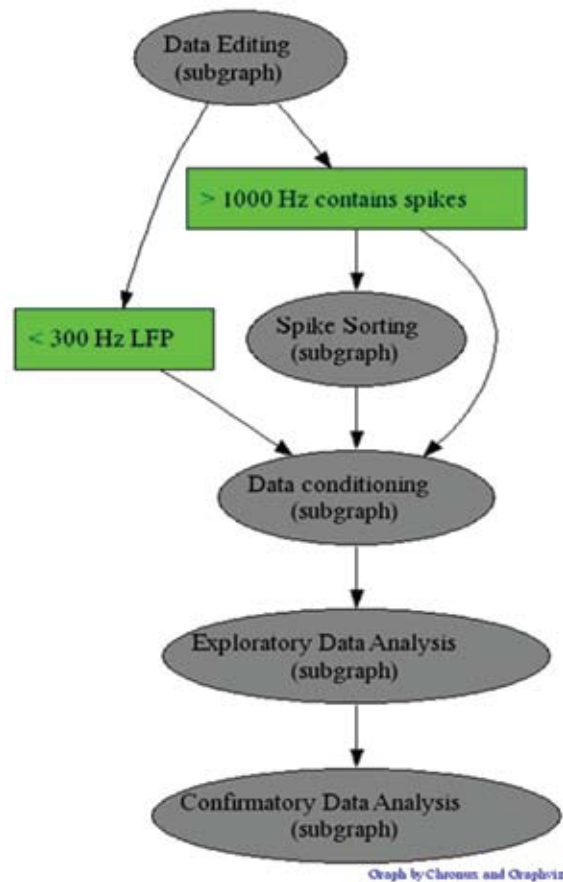


Figure 1. Electrophysiological data analysis protocol.

Typically, a first step to take in exploratory data analysis is to construct a summary of neural activity that is aligned with the appearance of a stimulus or some behaviorally relevant event. Recall that in the example described in “Spectral Analysis for Neural Signals,” the monkey is challenged with a delay period during which it must remember the location of a visual target that was cued at the beginning of a trial. Each

NOTES

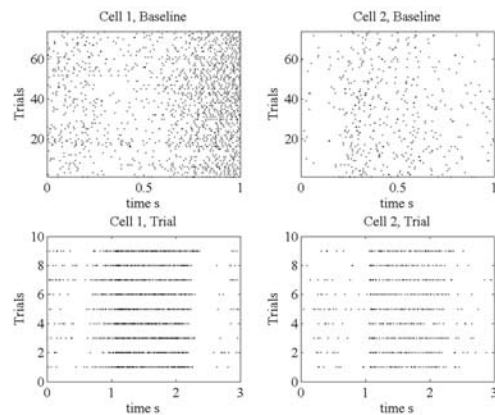


Figure 2. Raster plots for 2 isolated spikes recorded in monkey LIP. Each dot represents the time of occurrence of a spike. Each row details the spike times from a different trial in the experiment. Top row: baseline period (first second of each trial) for all trials. Bottom row: complete trials, including the baseline period (0-1 s) and delay and response periods (1-3 s) for a subset of trials associated with a single target.

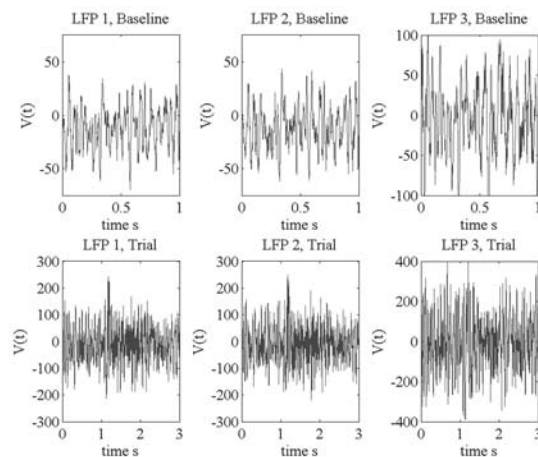


Figure 3. Local field potentials (LFPs) recorded concomitantly with the spikes shown in Fig. 1. The LFPs are averaged over the trials elaborated in the spike raster plots in Fig. 1. Top row: baseline period; bottom row: complete trials. Voltage values for the LFPs are in units of microvolts.

3 s trial is composed of a 1 s baseline period followed by a 2 s period containing the delay and response periods. The neural signals, contained in the tutorial data file *DynNeuroLIP.mat*, include three LFP signals and two point-process time series (i.e., two channels of spike times). Nine trials associated with one target direction are included, as are 72 trials of the baseline period combined across eight possible target positions. We will first produce an estimate of the firing rate associated with the delay period and then proceed to look for interactions between the two spikes and for any temporal structure in the LFPs.

A script that will take us through these steps can be launched by typing

```
>> lip_master_script
```

at the command prompt. The first figure generated by the script (Fig. 2) details the spike times (as raster plots) of the two single units for all the baseline periods (top row) and for the subset of trials (bottom row) associated with one particular target. Note that the number of spikes increases roughly 1 s after the start of the trial. This increase indicates that something is indeed happening at the start of the delay period and suggests that these neurons may play a role in establishing a working memory trace of one target's location. The tutorial script will also produce results as in Figure 3, where we see the three LFP signals that were recorded alongside the spikes. These signals likewise demonstrate a change in activity at the start of the delay period. As we proceed through the tutorial, we will see how Chronux can be used to further characterize the neural activity in these recordings.

Regression

Figure 4 illustrates one characterization that is often used for depicting spike data. The top subplot of the figure illustrates a standard frequency histogram, using a bin size of 104 ms, for a single trial of spike response. The rate is calculated by dividing the spike count in each bin by the bin width. The bottom subplot of the figure shows a smooth estimate of the firing rate generated by applying a local regression algorithm, *locfit*. In order to plot the regression fit and produce 95% confidence bounds for the rate

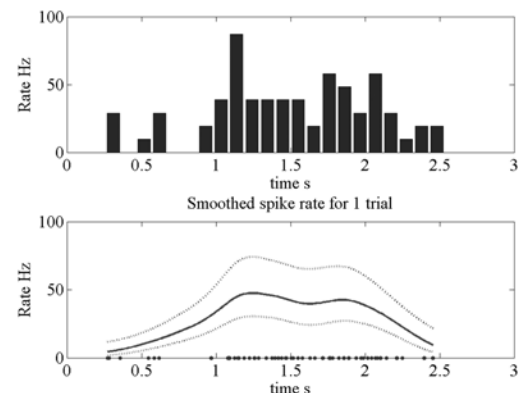


Figure 4. Spike rate estimates. Top row: frequency histogram constructed from a single trial for one isolated spike. Bin size = 104 ms. Bottom row: output from Chronux script *locfit*. The solid line depicts an estimate of the spike rate. The dashed lines indicate the 95% confidence interval for this estimate. The dots along the time axis represent the spike times. The nearest neighbor variable bandwidth parameter, *nn*, is set to 0.7 for *locfit*.

estimate, our tutorial script has run *locfit* using the following syntax:

```
>> fit=locfit(data,'family','rate')
```

followed by

```
>> lfplot(fit)
```

and

```
>> lfband(fit)
```

(Fig. 4, dashed lines in bottom subplot). In this case, we have opted to fit our single-trial spike train to a rate function by setting the *family* parameter of *locfit* to *rate*. Alternatively, we could have smoothed the spike data by choosing to fit it to a density function in which the smoothing is meant to determine the probability of firing a spike as a function of time. We generated density estimates by setting the *family* parameter in *locfit* to *density* instead of *rate*.

Note the dots that appear along the time axis of the bottom subplot: These are the spike times for the trial under consideration. *Locfit* will fit a linear, quadratic, or other user-specified function to some subset of the spikes, using each spike time in turn as the center point for the least-squares fit. The number of spikes in each subset can be stipulated in one of two ways: (1) as a fixed “bandwidth”, i.e., time interval. For example, if the parameter $h=1$ (and the spike times are given in units of seconds), then each local fit to the data will include 1 s of the trial; or (2) with $h=0$, and nn (nearest neighbor parameter) set to some fraction such as 0.3, in which case the time interval surrounding each spike will expand (or shrink) until 30% of the total number of spikes in the time series is included.

```
>> fit=locfit(data,'family','rate','h',1)
```

will produce a fit using a fixed bandwidth of 1 s. The bottom subplot of Figure 4 was produced with a nearest neighbor variable bandwidth,

```
>> fit=locfit(data,'family','rate','nn',0.7)
```

where nn was set to 0.7. If we change this value to a smaller fraction, say 0.3, then the smoothing will be done more locally, thereby revealing more of the temporal fluctuations in the spike rate (Fig. 5).

The data input to *locfit* can include multiple trials (following the data format rules outlined in the Appendix to this chapter). As seen in Figure 6 (which our tutorial script will also generate), the resulting fit

to our two spike trains appears smoother than the fit to the single trial, even though the nearest neighbor parameter (nn) is still 0.3. Although the regression is always done on a single time series, in this case, all the spike times from all the trials for each single unit are collapsed into one vector. Note how a smoother estimate arises in Figure 6 than in Figure 5 owing to the greater continuity across the samples (spike times) of the underlying rate function. Jackknife confidence limits can be computed for the multiple trials case by holding out each trial in turn from the regression fit and then calculating the mean and standard error from the ensemble of drop-one fits.

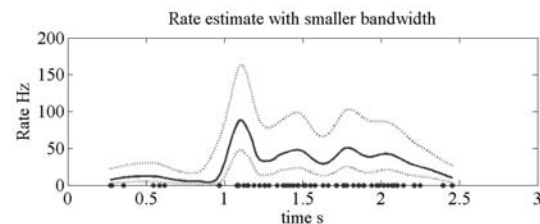


Figure 5. Spike rate estimate using *locfit* in Chronux. Here the nearest neighbor variable bandwidth parameter, nn , is set to 0.3.

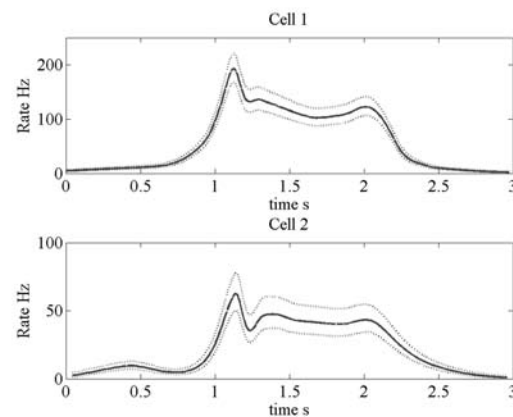


Figure 6. Spike rate estimates using *locfit*. Estimates are constructed using all spike times from all trials shown in the bottom row of Figure 2. The nn parameter of *locfit* is set to 0.3.

Spectra

We now begin our frequency-domain exploration of the dynamics of the LFPs and spikes in our data set. Our tutorial script will now generate Figure 7, which illustrates a multitaper spectrum calculated from the continuous voltage record in one LFP channel (sampling rate = 1 KHz) for a single trial. Only the delay period of the trial is included in the data array. The tutorial script *lip_master_script.m* includes the

NOTES

following three lines, which can also be run from the command prompt:

```
>>params.Fs=1000;
>>[S,f]=mthspectrumc(data,params);
>> plot_vector(S,f);
```

The first line sets the sampling rate and, therefore, the frequency resolution and range of the spectrum. Many Chronux functions use a structure, *params*, that contains a number of fields for assigning values to the parameters governing the Fourier analysis routines (see Appendix for more about the fields for *params*). The spectrum *S* and frequency range *f* used in the calculation are the outputs of *mthspectrumc*. A Chronux script (the third line in this code segment) can be used to perform special plotting. The default setting for *plot_vector* produces a plot with a log transform of *S* as a function of a linear frequency range. To plot the spectrum on a linear-linear set of axes, use

```
>> plot_vector(S,f,'n')
```

In Figure 7, the spectrum is plotted over a default range: from 0 Hz to the Nyquist limit for the sampling rate, 500 Hz. The output range of *S* and *f* is restricted by setting another field in the *params* structure, *params.fpass*. Figure 8 presents the LFP spectrum from the single trial but now with

```
>>params.fpass=[0 100]
```

and then, as before

```
>>params.Fs=1000;
>>[S,f]=mthspectrumc(data,params);
>> plot_vector(S,f);
```

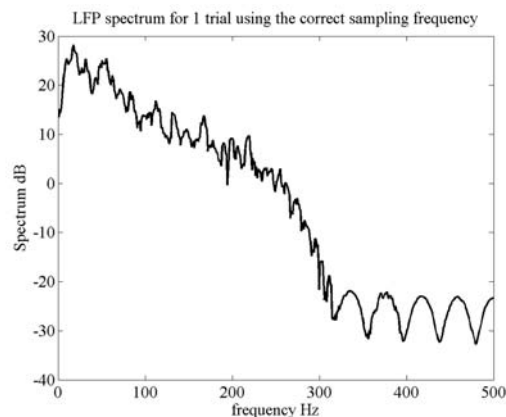


Figure 7. Multitaper spectrum for a single trial LFP; data selected from the delay period. The *y*-axis of the spectrum is in units of $\text{dB}=10 \cdot \log_{10}(S)$. *params.Fs*=1000, *params.tapers*=[3 5], *params.fpass*=[0 *params.Fs*/2], *params.pad*=0.

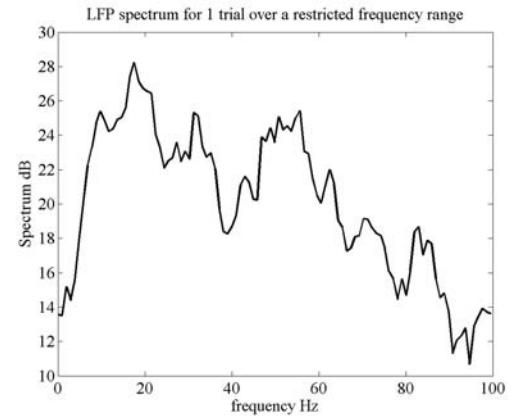


Figure 8. Multitaper spectrum for a single trial LFP. Data selected from the delay period. *params.Fs*=1000, *params.tapers*=[3 5], *params.fpass*=[0 100], *params.pad*=0.

The tutorial script will generate other examples of band-limited spectra after you choose lower limits and upper limits for the frequency range.

The spacing in the frequency grids used by the fast Fourier transforms (FFTs) called by Chronux can be adjusted through another field in the structure *params*. If *params.pad* = -1, then no zeros will be appended to the time series, and the frequency grid will be set by the defaults imposed by MATLAB. With *params.pad* = 0, the time series will be padded with zeros so that its total length is 512 sample points. For *params.pad* = 1, 2, ... the zero padding produces time series that are 1024, 2048, ... samples in length, respectively. As one can see by executing the next code segment,

```
>> params.pad=1;
>>[S,f]=mthspectrumc(data,params);
>> plot_vector(S,f,'y')
>> params.pad=3;
>>[S,f]=mthspectrumc(data,params);
>> plot_vector(S,f,'m')
```

the spectrum generated with a padding factor of 3 (Fig. 9, red) is computed on a much denser grid than the spectrum computed with a padding factor of 1 (Fig. 9, blue plot).

One advantage of taking the multitaper approach to spectral analysis is the ability to control the degree of smoothing in the spectrum. This is accomplished by adjusting the time-bandwidth product of the data windowing, which in turn is established by the choice of the number of Slepian tapers to use. The tutorial script *lip_master_script.m* again calculates the spectrum of the single trial LFP signal, but now with two different degrees of smoothing. As before, the number of tapers to use is set by a field in the

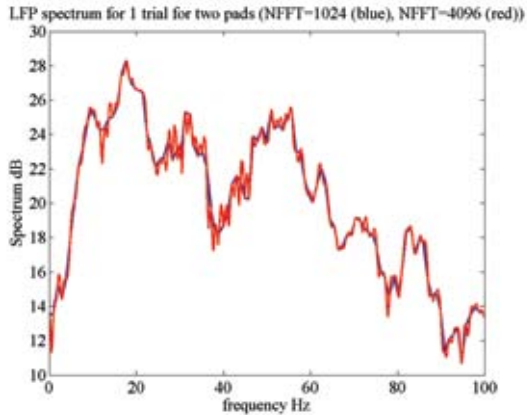


Figure 9. Multitaper spectrum for a single trial LFP; data selected from the delay period. $params.Fs=1000$, $params.tapers=[3\ 5]$, $params.fpass=[0\ 100]$, $params.pad=1$ (blue), $params.pad=3$ (red).

structure $params$: $params.tapers=[TW\ K]$, where TW is the time-bandwidth product and K is the number of tapers. For example, if

```
>> params.tapers=[3 5]
```

then the time-bandwidth product is $TW = 3$ and the number of tapers used is 5. The rule $K = 2 * TW - 1$ sets the highest number of tapers that can be used while preserving the good time-frequency concentration of the data windowing available from the Slepian taper sequences. Fewer tapers than the limit of five can be employed, and Chronux will produce a flag when the number of tapers requested is inconsistent with the TW . T is the length (typically in seconds) of our data segment. One can also think of this value as being established by the [number of samples in data segment] \times $1/Fs$ (inverse of the sampling rate). W is the half-bandwidth of the multitaper filter, and if we do not change T , we can demonstrate changes in smoothing as a function of changes in the half-bandwidth, as shown in Figure 10. The tutorial script will prompt the user to try other degrees of spectral smoothing by entering new values for the time-bandwidth product.

To compute the spectrum over a number of trials and return an average thereof, we set the $trialave$ field in the structure $params$ to 1. The tutorial script will carry out the following steps:

```
>> params.trialave=1;
>> [S,f]=mtspectrumc(data,params);
>> plot_vector(S,f)
```

If $trialave = 0$, and the structured array $data$ have any number of trials, the output S will be a matrix where each column is the spectrum computed from one trial's neural signal.

Chronux will also calculate and plot error bars for multitaper spectra. Two different types of confidence interval estimates are available. If we set the field err in $params$ to

```
>> params.err=[1 p], with  $p=0.05$ , and then execute
>> [S,f,Serr]=mtspectrumc(data,params);
>> plot_vector(S,f,[],Serr);
```

Chronux will plot the spectrum bracketed by the theoretical 95% confidence limits for that estimate. The array $Serr$ contains the $(1 - p)\%$ limits, with the lower limit in the first row and the upper limit in the second row of the array. In this case, the confidence bounds are based on the parametric distribution for the variance of a random variable, i.e., the chi-square, with two degrees of freedom. If instead, we set the field err in $params$ to

```
>> params.err=[2 p], with  $p=0.05$ ,
```

the 95% confidence bounds will be derived from a jackknife estimate of the standard error for the sample spectra. Thus, if we run the lines of code given above for the theoretical confidence interval, and continue with

```
>> hold
>> p=0.05;
>> params.err=[2 p];
>> [S,f,Serr]=mtspectrumc(data,params);
>> plot(f,10*log10(Serr(1,:)),'r');
>> plot(f,10*log10(Serr(2,:)),'r');
```

a figure similar to that seen in Figure 11 should be rendered by the tutorial script.

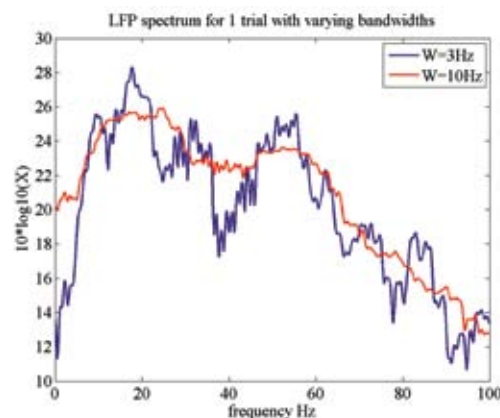


Figure 10. Multitaper spectrum for a single trial LFP; data selected from the delay period (1 s duration, so $T = 1$). $params.Fs=1000$, $params.tapers=[3\ 5]$ (blue), $params.tapers=[10\ 19]$ (red), $params.fpass=[0\ 100]$, $params.pad=2$

NOTES

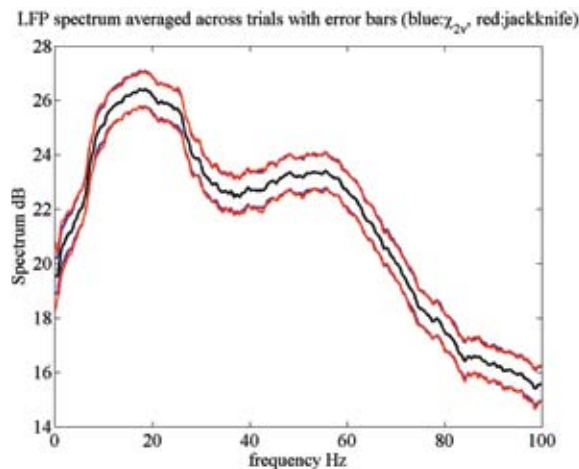


Figure 11. Multitaper spectrum for LFP using all trials associated with one target; data selected from the delay period. `params.Fs=1000`, `params.tapers=[10 19]`, `params.fpass=[0 100]`, `params.pad=2`, `params.trialave=1` (average spectrum shown in black), `params.err=[1 .05]` (blue), `params.err=[2 .05]` (red).

Note that for these data, the jackknife confidence interval (in red) is in good agreement with the so-called theoretical interval (in blue).

As discussed in detail in other chapters, multitaper spectra can be calculated for point process data. As described in the Appendix herein, Chronux contains a whole set of analogous scripts for point processes that match those for continuous data. However, the suffixes of the script names carry a *pt* or *pb*, for point times and point binned, respectively, instead of a *c*, for continuous. For example, the script `mtspectrumpt.m` will compute the multitaper spectrum for data represented as a series of spike times. The following section of MATLAB code will extract a data segment of interest from the trials, set the appropriate params fields, compute the spectrum for the spike data, and plot the results:

```
data=dsp1t; % data from 1st cell
delay_times=[1 2]; % start and end time
                    % of delay period

data=extractdatap
(data,delay_times,1); % extracts spikes within
                    % delay period

params.Fs=1000; % inverse of the spacing
                % between points on the
                % grid used for computing
                % Slepian functions

params.fpass=[0 100]; % range of frequencies
                    % of interest

params.tapers=[10 19]; % tapers
```

```
params.trialave=1; % average over trials
p=0.05; % p value for errors
params.err=[1 p]; % chi2 errors
[S,f,R,Serr]=mtspectrumpt
(data,params);
```

The output should be similar to that presented in Figure 12. The tutorial script should be able to produce this figure. One thing to note here is the high number of tapers, 19, used for computing the spectrum. Owing to the inherent complexity of even a single spike's power spectrum, extensive smoothing often helps represent spike spectra. The output variable *R* is something unique to spike spectra: It is the high-frequency estimate of the spike rate derived from the spectrum. This estimate is either made on a trial-by-trial basis or based on the average, depending on the setting of the parameter `params.trialave`. In Figure 12, the mean rate estimates appear as dotted horizontal lines.

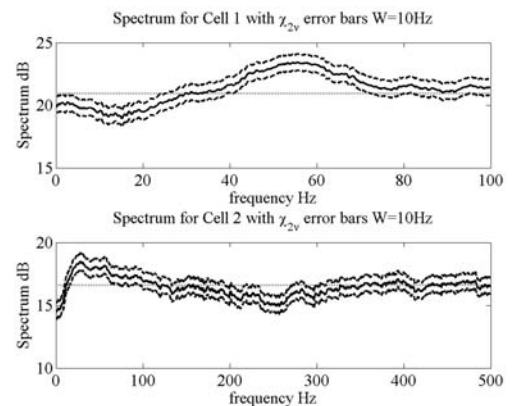


Figure 12. Multitaper spectrum for two spikes recorded in area LIP; delay period activity only. Top: Cell 1, `params.Fs=1000`, `params.tapers=[10 19]`, `params.fpass=[0 100]`, `params.pad=0`, `params.trialave=1` (average, heavy line), `params.err=[1 .05]` (dashed lines), mean rate estimate (dotted horizontal line). Bottom: Cell 2, `params.Fs=1000`, `params.tapers=[10 19]`, `params.fpass=[0 500]`, `params.pad=0`, `params.trialave=1` (average, heavy line), `params.err=[1 .05]` (dashed lines), mean rate estimate (dotted horizontal line).

Spectrograms

This section will illustrate how Chronux controls the calculation of time-frequency representations of neural data. Chronux can generate spectrograms for continuous data (like EEGs and LFPs) as well as point process activity (spike times and binned spike counts). An important component of the Chronux spectrogram is the sliding window, which sets the width of the data window (usually specified in seconds) and how much the window should slide along the time axis between samples. Within each

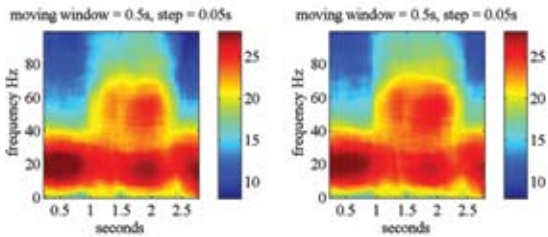


Figure 13. Time-frequency spectrograms for two LFP channels. Activity from all trials, over the entire trial (3 s) used for the analysis. *Movingwin*=[.5 .05], *params.Fs*=1000, *params.tapers*=[5 9], *params.fpass*=[0 100], *params.pad*=0, *params.trialave*=1, *params.err*=0.

window, multitaper spectral analysis of the data proceeds as it does when calculating standard spectra. However, one must remember that the spectral analysis is restricted to the temporal window for the data. Thus, the number of tapers used for the spectrogram should reflect the time-bandwidth product of the window, not the dimensions of the entire data segment of interest. Extensive temporal overlapping between successive windows will tend to produce smoother spectrograms. The following code fragment from the tutorial script (*lip_master_script.m*) will help generate spectrograms for two of the LFP channels in our data set (Fig. 13):

```

movingwin=[0.5 0.05];           % set the moving
                                % window dimensions
params.Fs=1000;                 % sampling frequency
params.fpass=[0 100];          % frequencies of
                                % interest
params.tapers=[5 9];           % tapers
params.trialave=1;             % average over trials
params.err=0;                  % no error
                                % computation

data=dllfp1t;                  % data from channel 1
[S1,t,f]=mtspecgramc
(data,movingwin,params);      % compute
                                % spectrogram

subplot(121)
plot_matrix(S1,t,f);
xlabel('');
caxis([8 28]); colorbar;

data=dllfp2t;                  % data from channel 2
[S2,t,f]=mtspecgramc
(data,movingwin,params);      % compute
                                % spectrogram

subplot(122);
plot_matrix(S2,t,f);
xlabel('');
caxis([8 28]); colorbar;

```

Note the use of the special Chronux plotting routine *plot_matrix*. Here the window is set to 500 ms in duration with a slide of 50 ms along the time axis between successive windows.

The same sets of parameters used for continuous LFP signals can be employed for calculating the spike spectrograms (Fig. 14). However, one useful modification to make when plotting spike spectrograms is to normalize the spike power S by the mean firing rate R . For example,

```

data=dsp1t;                    % data from 1st cell
[S,t,f,R]=mtspecgrampt
(data,movingwin,params);      % compute
                                % spectrogram

figure;
subplot(211);

                                % plot spectrogram
                                % normalized by rate

plot_matrix(S./repmat(R,
[1 size(S,2)]),t,f);xlabel('');
caxis([-5 6]);colorbar;

data=dsp2t;                    % data from 2nd cell
[S,t,f,R]=mtspecgrampt
(data,movingwin,params);      % compute
                                % spectrogram

subplot(212);

                                % plot spectrogram
                                % normalized by rate

plot_matrix(S./repmat(R,
[1 size(S,2)]),t,f);
caxis([-5 6]);colorbar;

```

The normalized spectrograms demonstrate how the spike power fluctuates across the trials with respect to the mean rate. Here one can readily observe that, while there is an increase in gamma-band power in the spike discharge with respect to the mean rate during the delay period (Fig. 14, yellow-orange colors, top subplot), the power in the lower-frequency fluctuations in the spike discharge is suppressed with respect to the mean rate (blue colors).

Coherence

As an example of the use of Chronux software for evaluating the strength of correlations between different neural signals, we will calculate the spike-field coherence for pairs drawn from the three LFP channels and two spike channels in our monkey parietal lobe data set. As discussed in “Spectral Analysis for Neural Signals,” spike-field coherence is a frequency-domain representation of the similarity of dynamics between a spike train and the voltage fluctuations produced by activity in the spiking neuron’s local neural environ-

NOTES

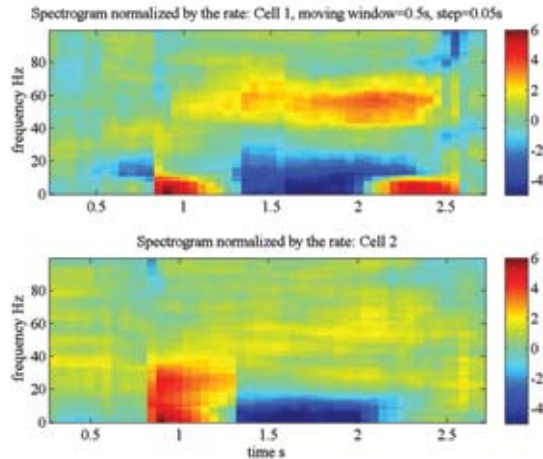


Figure 14. Time-frequency spike spectrograms for two spikes recorded in LIP. Activity from all trials, over the entire trial (3 s) used for the analysis. Spectrograms are normalized by the mean rates of the two single units. *Movingwin*=[.5 .05], *params.Fs*=1000, *params.tapers*=[5 9], *params.fpass*=[0 100], *params.pad*=0, *params.trialave*=1, *params.err*=0.

ment. As before, we start by setting the values of the parameters carried by the structure *params*:

```

params.Fs=1000;           % sampling frequency,
                          % same for LFP and spike
params.fpass=[0 100];    % frequency range
                          % of interest
params.tapers=[10 19];  % emphasize smoothing
                          % for the spikes
params.trialave=1;       % average over trials
params.err=[1 0.05];     % population error bars

delay_times=[1 2];      % define the delay period
                          % (between 1 and 2
                          % seconds)

datasp=extractdatapt
(dsp1t,delay_times,1);  % extract the spike data
                          % from the delay period

datafjp=extractdatac
(dlfp1t,params.Fs,
delay_times);          % extract the LFP data
                          % from the delay period

[C,phi,S12,S1,S2,f,
zerosp,confC,phistd]=
coherencycpt
(datafjp,datasp,params); % compute the coherence

```

Note that the script for computing the coherency is *coherencycpt*, a function that handles the hybrid case mixing continuous and point process data. For the outputs, we have the following:

- *C*, the magnitude of the coherency, a complex quantity (ranges from 0 to 1);

- *phi*, the phase of the coherency;
- *S1* and *S2*, the spectra for the spikes and LFP signals, respectively;
- *f*, the frequency grid used for the calculations;
- *zerosp*, 1 for trials for which there was at least one spike, 0 for trials with no spikes;
- *confC*, confidence level for *C* at $(1 - p)\%$ if *params.err*=[1 *p*] or *params.err*=[2 *p*]; and
- *phistd*, theoretical or jackknife standard deviation, depending on the *params.err* selection

These are used to calculate the confidence intervals for the phase of the coherency.

This code segment, which is called by the tutorial script, should generate a graphic similar to Figure 15. The top row of the figure shows the spike-field coherence for spike 1 against the three LFP channels. The bottom row has the spike-field coherence estimates for spike 2 against the three LFP channels. The figure depicts the confidence level for the coherence estimates as a horizontal dotted line running through all the plots; coherence values above this level are significant. We see from this figure that the spikes and LFPs in the monkey parietal cortex showed an enhanced coherence during the delay period for the frequency range from ~25 Hz to more than 100 Hz for all the matchups, except LFP3, with both spikes. For the coherence measures involving LFP3, the coherence is not significant for very fast fluctuations (>90 Hz).

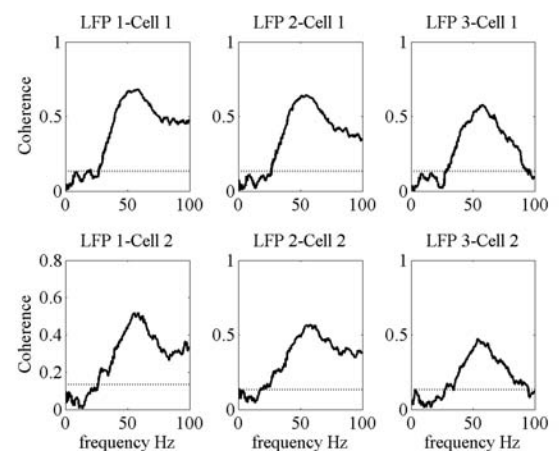


Figure 15. Spike-field coherence. Top row: coherence estimates between cell (spike) 1 with LFP channel 1 (left), LFP channel 2 (middle), and LFP channel 3 (right). Bottom row: coherence estimates between cell (spike) 2 with LFP channel 1 (left), LFP channel 2 (middle), and LFP channel 3 (right). Significance level for the coherence estimates: horizontal dotted line running through all plots.

Using Chronux, we can also estimate the coherence between two spike trains. The setup of the parameters for the calculation is very similar to that required for the hybrid script:

```
>> params.err=[2 p];
>> [C,phi,S12,S1,S2,f,zerosp,confC,phistd,
Cerr]=coherencypt(dataspl,datasp2,params);
```

Here, *phistd* is the jackknifed standard deviation of the phase, and *Cerr* is the $(1 - p)\%$ confidence interval for the coherence. Figure 16 shows a plot of the spike-spike coherence, comparing the delay period activity (in blue) with the coherence during the baseline (in red).

Denosing

In Figure 17, we expand the Data Conditioning subgraph of the electrophysiology analysis protocol first introduced in Figure 1. The branch for the LFP data carries us through two stages of processing:

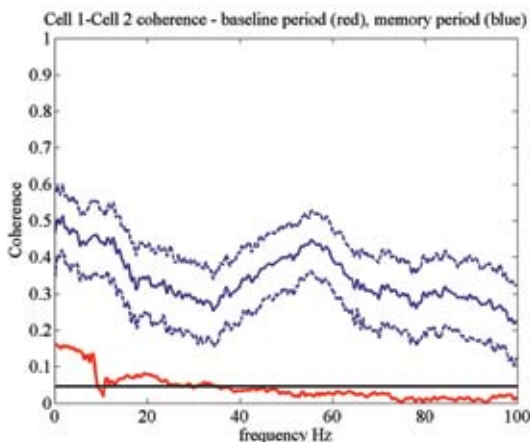


Figure 16. Coherence between spikes of cell 1 and cell 2. Blue traces: data restricted to the delay period of each trial (solid line, average coherence; dashed lines, 95% jackknife confidence interval for this estimate of the coherence). Red trace: data restricted to the baseline period of each trial. Horizontal line: significance level for the coherence estimates. *params.Fs=1000, params.tapers=[10 19], params.fpass=[0 100], params.pad=0, params.trialave=1, params.err=[2 .05]*.

local detrending and the testing and removal of 60 Hz line noise. Electrophysiological recordings, both in the research laboratory and in clinical settings, are prone to contamination. 60 Hz line noise (50 Hz in Europe), slow drifts in baseline voltage, electrical transients, ECG, and breathing movements all contribute different types of distortion to the recorded signal. Methods for removing particular waveforms, such as ECG and large electrical transients, have good solutions that are treated elsewhere (Perasan B,

“Spectral Analysis for Neural Signals”; Mitra and Pesaran, 1999; Sornborger et al., 2005; Mitra and Bokil, 2008). We will focus here on slow fluctuations in electrophysiological signals and line noise.

If we add a sinusoidal voltage fluctuation to one of

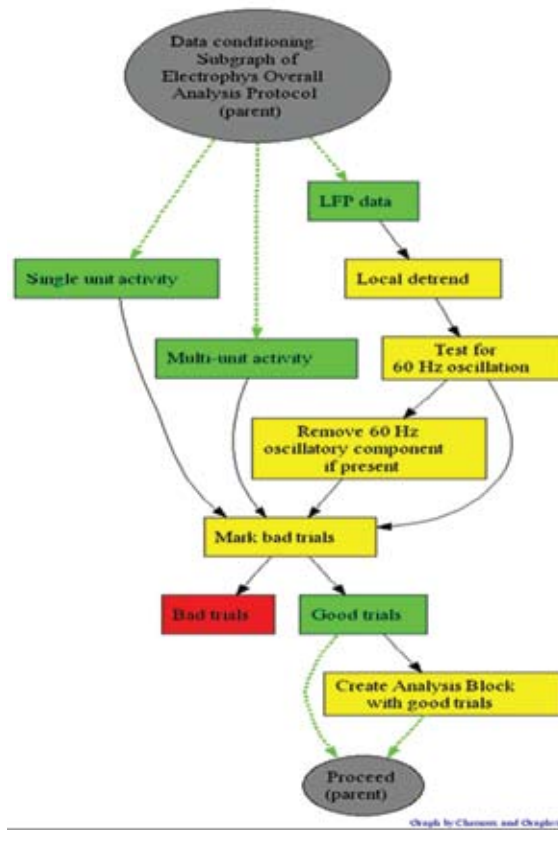


Figure 17. Data conditioning component of electrophysiological analysis protocol.

our LIP LFP recordings, the result will look something like Figure 18 (top left). Such a slow fluctuation could be entirely the result of changes in the electrostatic charges built up around the recording environment. Therefore, it is noise and we should try to remove it. With Chronux, we use a local linear regression to detrend neural signals. The script *locdetrend* utilizes a moving window controlled by *params* to select time samples from the signal. The best fitting line, in a least-squares sense, for each sample is weighted and combined to estimate the slow fluctuation, which is then removed from the data signal.

For Figure 18 (top middle),

```
>> dLFP=locdetrend(LFP,[.1 .05]).
The dimensions of the sampling window are 100 ms
```

NOTES

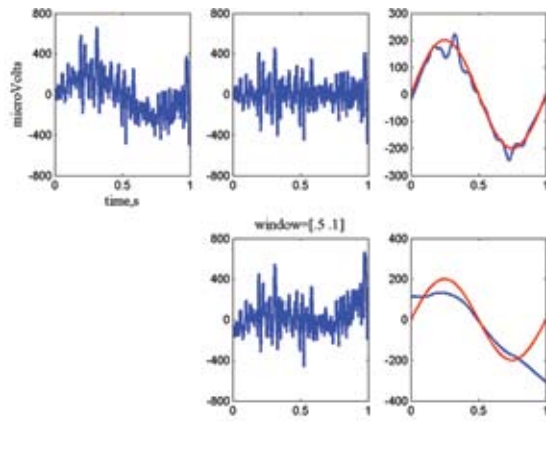


Figure 18. Application of *locdetrend*.

in duration, with a window shift of 50 ms between samples. Note that the detrended signal has much less low-frequency fluctuation than the original signal (Fig. 18, top left). In Figure 18 (top right), we see that the estimate of the slow fluctuation (blue) does a pretty good job of capturing the actual signal (red) that was added to the LFP data. However, if the sampling window parameters are not well matched to changes in the signal, the detrending will not be successful.

For Figure 18 (bottom, center),

```
>> dLFP=locdetrend(LFP,[.5 .1]).
```

Window duration = 500 ms, half the sample length with a window shift of 100 ms. Here the estimate of the slow fluctuation (blue) does a poor job of capturing the sinusoid (Fig. 18, red, bottom right).

Chronux accomplishes the removal of 60 Hz line noise by applying Thomson's regression method for detecting sinusoids in signals (Thomson, 1982). This method does not require that the data signal have a uniform (white) power spectrum. The Chronux script *rmlinesc* can either remove a sinusoid of chosen frequency or automatically remove any harmonics whose power exceeds a criterion in the F-distribution (the F-test). Figure 19 demonstrates the application of the F-test option of *rmlinesc*.

```
>>no60LFP=rmlinesc(LFP,params).
```

Here the LFP (Fig. 19, top left) has been contaminated with the addition of a 60 Hz sinusoid. The multitaper spectrum of this signal is shown in Figure 19 (top right panel). Note the prominent 60 Hz element in this spectrum (broadened but well defined by the application of the multitaper technique). The spec-

trum of *no60LFP* is shown in the figure's bottom left panel; the time series with the 60 Hz noise removed, the vector returned by *rmlinesc*, is shown in the bottom right panel.

Appendix: Chronux Scripts

While not an exhaustive list of what is available in Chronux, the scripts enumerated here (discussed in this chapter) are often some of the most useful for trying first during the early phase of exploratory data analysis. This section describes the means for setting some of the more important parameters for controlling multitaper spectral calculations, as well as the basic rules for formatting input data.

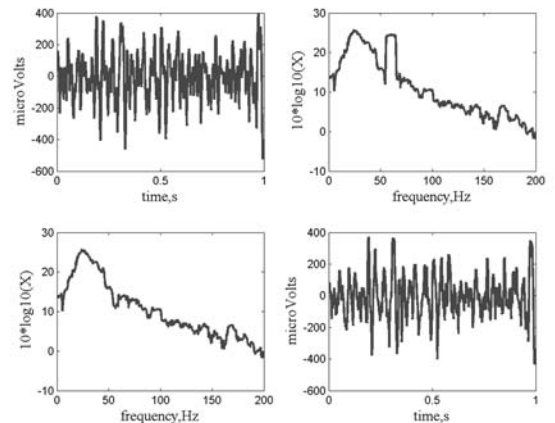


Figure 19. Application of *rmlinesc*.

Denosing

- (1) Slow variations (e.g., movements of a patient for EEG data)

locdetrend.m: Loess method

- (2) 50/60 Hz line noise

rmlinesc.m

rmlinesmovingwinc.m

Spectra and coherences (continuous processes)

- (1) Fourier transforms using multiple tapers

mtfftc.m

- (2) Spectrum

mtspectrumc.m

- (3) Spectrogram

mtspecgramc.m

- (4) Coherency

mtcoherencyc.m

- (5) Coherogram

mtcohgramc.m

Analogous scripts are available for analyzing time

series data organized in alternative formats. Point-process time series data can be analyzed using *mtfftpt.m*, *mtspectrumpt.m*, etc. Binned spike count data can be analyzed with *mtfftb.m*, *mtspectrumb.m*, etc. An additional set of scripts is available for calculating the coherence between a continuous series and a point-process time series (*coherencycpt.m*, *coherogramcpt.m*, etc.), and for the coherence between a continuous and binned spike counts (*coherencycpb.m*, *coherogramcpb.m*, etc.).

In a typical function call, such as

$[S, f, Serr] = mtspectrumc(data, params)$, a structure *params* is passed to the script. This structure sets values for a number of important parameters used by this and many other algorithms in Chronux.

params.Fs Sampling frequency (e.g., if data are sampled at 1 kHz, use 1000).

params.tapers Number of tapers to use in spectral analysis specified by either passing a matrix of precalculated Slepian tapers (using the *dpss* function in MATLAB) or calculating the time-frequency bandwidth and the number of tapers as $[NW K]$, where K is the number of tapers. Default values are *params.tapers*=[3 5].

params.pad Amount of zero-padding for the FFT routines utilized in the multi-taper spectral analysis algorithms. If *pad* = -1, no padding; if *pad* = 0, the FFT is padded to 512 points; if *pad* = 1, the FFT is padded to 1024 points, *pad* = 2, padding is 2048 points, etc. For a spectrum with a dense frequency grid, use more padding.

params.fpass Frequency range of interest. As a default, $[0 Fs/2]$ will allow from DC up to the Nyquist limit of the sampling rate.

params.err Controls error computation. For *err*=[1 *p*], so-called theoretical error bars at significance level *p* are generated and placed in the output *Serr*; *err*=[2 *p*] for jackknife error bars; *err*=[0 *p*] or *err*=0 for no error bars (make sure that *Serr* is not requested in the output in this case).

params.trialavg If 1, average over trials/channels; if set to 0 (default), no averaging.

Local regression and likelihood

(1) Regression and likelihood

locfit.m

(2) Plotting the fit

lfplot.m

(3) Plotting local confidence bands

lfband.m

(4) Plotting global confidence bands

scb.m

Data format

(1) Continuous/binning spike count data

Matrices with dimensions: time (rows) \times trials/channels (columns)

Example: 1000×10 matrix is interpreted as 1000 time-point samples for 10 trials from 1 channel or 1 trial from 10 channels. If multiple trials and channels are used, then add more columns to the matrix for each additional trial and channel.

(2) Spike times

Structured array with dimension equal to the number of trials/channels.

Example: *data*(1).times=[0.3 0.35 0.42 0.6]
data(2).times=[0.2 0.22 0.35]

Chronux interprets *data* as two trials/channels: four spikes collected at the times (in seconds) listed in the bracket for the first trial/channel, and three spikes collected in the second trial/channel.

Supported third-party data formats

NeuroExplorer (.NEX)

Plexon (both .PLX and .DDT file formats)

References

Mitra PP, Bokil H (2008) Observed Brain Dynamics. New York: Oxford UP.

Mitra PP, Pesaran B (1999) Analysis of dynamic brain imaging data. *Biophys J* 76:691-708.

Sornborger A, Yokoo T, Delorme A, Sailstad C, Sirovich L (2005) Extraction of the average and differential dynamical response in stimulus-locked experimental data. *J Neurosci Methods* 141:223-229.

Thomson DJ (1982) Spectrum estimation and harmonic analysis. *Proc IEEE* 70:1055-1096.