Physics Z-5156, Fall 2004

Chapter 4: The Spike-Sorting Problem in Neurobiology

Copy the question files and suggested program templates from /home/5156/assignments/neuro, for example using the following commands:

```
  cd
  cp -r /home/5156/assignments/neuro .
  cd neuro
  ln -s /home/5156/data/b01c1b.bin .
```

start at your home directory

copy the directory to all levels

begin working in this directory

create a symbolic link to the raw data

For this project, I would like a short report with appropriate accompanying figures, graphs, and tables. The main point of the project is to answer as well as you can two questions, as explained in more detail below: (1) how many distinct neurons has the probe picked up; and (2) for each such neuron, what is the sequence of firing times? Once you have come to your conclusions, I would like you to compare your results to those of one or more classmates. You should comment on any differences. I don’t expect certainty and will not be grading on correctness; there may not be time in two weeks to accomplish a detailed statistical analysis establishing that your result is the unique correct answer. Spike sorting remains an open problem in statistical biology and the subject of much current research.

Please write in plain text (a file you make with vi or emacs), troff, TeX, or Latex. Please do not use HTML, SGML, or Microsoft “quoted printable” formats, as they are difficult to read. I cannot read Microsoft Word at all, but Postscript generated by Word is acceptable. When you submit your results, please make clear with a README file which files in the directory contain your report and programs. When you are done, issue the make submit command to tell me you have finished. Do not change anything in the directory after running make submit.

Some biological and electrochemical background

Dr. Kendall Morris, USF Department of Physiology and Biophysics, is interested in understanding the mechanism by which the vertebrate brain stem controls respiration (breathing). Understanding this relatively simple part of the brain is a first step to interpreting the complex spatio-temporal patterns characteristic of higher functions.

Using the cat as a model, Dr. Morris implants 72 electrodes in the brain stem. Since the electrodes are extracellular, each picks up signals from several neurons at once. The very first task is to disentangle the signals, replacing the time series of voltage measurements on each electrode with one time series for each neuron. The next step would be to look for statistically significant correlations among neurons and between neurons and the respiratory cycle. The emerging patterns can then be compared to physical theories of collective behavior, such as neural networks (which are related to spin glasses) and dynamical systems. These models come from the field of statistical mechanics, and many of the researchers in the field have backgrounds in physics and statistics.

A long axon connects a neuron’s cell body to a distant synapse. Experiments in the first half of the twentieth century using intracellular probes in the giant-squid axon established a simple picture of how neurons conduct electrical signals. Normally, active molecular pumps maintain a

---


4. The Spike-Sorting Problem in Neurobiology

deficit of Na\(^+\) inside the cell relative to the exterior, resulting in a net negative potential inside of approximately 80mV. The cell membrane can be modeled as a capacitive-resistive-battery network, supporting the propagation of a depolarizing pulse. As such a pulse approaches a segment of membrane, reducing the negative polarization, a threshold is reached at which sodium channels open, briefly resulting in a large net positive charge inside the axon. Other molecular mechanisms (Ca\(^{++}\), K\(^+\), organic ions, and Cl\(^-\) all play important roles) then quickly repolarize the segment, sometimes overshooting the rest potential, and usually a refractory period follows during which the membrane slowly recovers its resting state and cannot fire. The traveling pulse is called an *action potential*, and normally (absent experimental probes such as drugs and applied voltages) is an all-or-nothing, invariant response triggered wherever a depolarization exceeds some threshold.

![Figure 1. Schematic representation of a typical action potential (adapted from Nichols, Martin, Wallace, op. cit.).](image)

We take one important feature from this model: for a given type of neuron, every spike is approximately the same (in duration, amplitude, and shape) as every other spike, so the only way for a neuron to distinguish the strength of a stimulus is to fire at a greater or lesser rate. (This may not be strictly true of certain types of motor bursters; I do not know if we have them in our data set. Dispersion may also change the shape with distance, but this does not alter the conclusion: information in a single neuron can normally be encoded only in the frequency of action potentials.)

Unfortunately, few axons are so large as those of the giant squid or take so well to having glass tubes poked in them. The signals in Dr. Morris’s data, taken outside the cells, in the largely conducting saline intracellular fluid, are thus strongly attenuated and possibly dispersed echoes of the intracellular spike. The researcher needs to amplify the signals significantly. (The difficulty of these experiments is another reason that so many of the people doing them have a physics background.) If we assume that every spike is intrinsically the same as every other, we can interpret different measured amplitudes and shapes in the data stream as originating from different neurons different distances from the electrode. This provides the basis for spike sorting.

**The data**

The file b01c1b.bin holds a short (one-minute) excerpt from one electrode. The data were sampled at a rate of 24 KHz from an A.C.-coupled analog amplifier, meaning that what we see is effectively the derivative of the voltage. The amplifier stage uses an analog notch filter to cut
4. The Spike-Sorting Problem in Neurobiology

out 60-Hz noise from the electronics, but persistent problems with ground loops (we are, after all, trying to measure voltage differences inside a saline solution) have left sharp components at all odd harmonics of 60 Hz. Further noise may come from the amplifier and from neurons too distant to resolve. There is also a strong electro-cardiogram at around 4 Hz.

When I received the file b01c1b.bin, I first had to determine the format. Commands like `od -f b01c1b.bin | more` established the format was unlikely to be floating point, so I tried `od -x` and `od -t xL`, the former looking at two-byte integer shorts, the latter at four-byte longs, in hexadecimal. Interpreted as shorts, the data seemed to change slowly (note that negative numbers roll over backward, putting the first hexadecimal digit in the range 8–F, so that $-2$ for example would be FFFFE). The best news was that there didn’t appear to be a header, a block of information preceding the data that might be of unknown length and undocumented format.

I then had to decide whether to work with the data in binary or ASCII form. The former is much more efficient, but the latter lets one see what one’s doing. In either case, our graphing programs eventually require ASCII.

Once you have written the little conversion program—you might call it `b2a`, for binary-to-ASCII—you should start looking at the data. The time series runs to 1.4 million words (shorts), which is not so large that we can’t look at it all at once:

```
  b2a < *bin | axis -a | xplot
```

The `-a` switch tells `axis` automatically to supply abscissas to the single column of ordinates in the output of `b2a`. I then recommend looking at more manageable pieces of the data; this is where it’s useful to make a working copy of the ASCII representation:

```
  % mkdir /scratch/yourname
  % b2a < *bin > /scratch/yourname/data.ascii
  % ln -s !$. write big file on scratch disk
  % head -2200 < *ascii | tail -300 | axis -a | xplot
  look at points 1901–2200
```

I started looking at these numbers on a Sun, where the byte order is reversed. With `od -x`, it was evident that the least significant byte (two hexadecimal digits) was changing far more slowly than the most significant. I then had to write a little ten-line program to reverse the byte order. Sun Spares are called “big-endian,” because the most significant byte comes first, while Intels are “little-endian.” Compare the output of `od -tx2` with that of `od -tx1`.

---

4 I started looking at these numbers on a Sun, where the byte order is reversed. With `od -x`, it was evident that the least significant byte (two hexadecimal digits) was changing far more slowly than the most significant. I then had to write a little ten-line program to reverse the byte order. Sun Spares are called “big-endian,” because the most significant byte comes first, while Intels are “little-endian.” Compare the output of `od -tx2` with that of `od -tx1`. 
The segment of this particular example contains two obvious spikes and a representative sample of noise. The lower amplitude of the second spike and the short interval (2.5 msec) after the previous both suggest, but do not prove, that the spikes came from different neurons. Alternatively, we could be looking at a burster, or the differences could be due simply to noise. We could try to rule out the latter possibility on statistical grounds.

Filtering

One kind of filtering we should do before anything else: subtract the mean value from the data. Because the amplifier is AC coupled, this mean value should be small.

We’d eventually like to look at autocorrelations of the subseries for each neuron we find; see *Numerical Recipes*, chapter 13. If we did so right now, we’d find a series of sharp spikes, sometimes a good omen, but not here: they’re at the odd harmonics greater than the fundamental of 60 Hz line noise and completely drown out anything of possible interest.

Short-answer question 1 (answer this in the file q1). I inferred a fundamental somewhat different from 60 Hz, which could be due to fluctuation in the electric utility or to a small difference from 24 KHz in the digitization rate. Using your power-spectrum software from last week, estimate the number that should be used instead of 60 Hz. Quote an appropriate number of significant figures. I found the peaks with a little gawk script, then (by hand, but I could have used a script) interpolated the true peak positions using the highest point and one to either side.

I wrote a notch filter that cuts out a triangular piece of frequency space centered at all the odd multiples of “60” Hz, making the base of the triangular notch a command-line argument. The procedure is simple: FFT the data, multiply by a function $H(f)$ that is 1.0 except at the triangular notches, then FFT back. I also implemented overlapping segments of variable length, as discussed last week in the context of spectral estimation. When you implement this, pick the notches relatively narrow, or you will change the heights and shapes of your spikes. (To make sure you’re not doing so, plot selections from the raw and filtered data together). Since spikes are quite narrow in time, they are very spread out in frequency, meaning that they’re unusually sensitive to any kind of Fourier filtering.

Getting rid of the harmonic noise is a boon for the autocorrelation, but it seems not to do much for figure 2 and may not be necessary for spike sorting. I’ll defer further consideration of filtering.

Triggering

To sort spikes, we first must find them. Using the software you wrote in the first week, make a histogram of the measurements (or of their absolute values; either is informative). You could start with the raw data (after subtracting the mean), with the harmonic-notch-filtered data, or with some other filtered set (such as the wavelet-filtered data I describe later). If your filtering has not damaged the signal, it shouldn’t make much of a difference (but I could be wrong here).

There are obviously fewer measurements the further we go from zero. We might expect pure noise to have a Gaussian distribution; any systematic deviation in the tails indicates a signal we have some hope of finding. To estimate whether or not the histogram looks Gaussian, I instructed axis to print on a logarithmic scale; noise would then be an inverted parabola. Instead, I found figure 3:
The tails, particularly on the negative end, are unmistakable. So we look for all signals more negative than some multiple $\alpha$ of the standard deviation (which we compute using last week’s `dist` script). The one-line `gawk` program “`gawk \'$1<\$2\*cutoff\{print NR\}`” accomplishes the selection. (NR in `gawk` is the record, i.e., line number. Counting starts at 1, unlike in C.) You can set $\alpha$ graphically from figure 3 or by asking how many events you would expect that many standard deviations out, assuming it were all noise.\(^5\) Pick an $\alpha$ such that the number expected is enormously fewer than the number observed: then almost all the observed peaks are probably real. One needs to do a little better than the `gawk` line above, since it will trigger several times for each negative spike: I wrote a short little `gawk` script that does essentially the same thing but considers only local minima.

In fact, we should be able to extract spikes even if they’re in the noise, because we can select not just a single time but the coincidence, say, of a positive peak followed by a negative followed by a positive, all in a time typical of a spike (see figure 2). I leave this for you to pursue.

Once I’ve triggered, I need to be able to pull out each spike. I did this in two steps: first, I used the `gawk` script described above to select the times (NR’s) at which there is a local minimum more negative than the cutoff, $-\alpha$ times the standard deviation, putting the results in a file. Then I wrote a C program to go through that file and extract the context of each spike, which I defined (heuristically) as 50 points, starting 19 time steps before the trigger. I wrote the program in C rather than `gawk` because it’s a little tricky to handle overlapping snippets. The program writes each 50-point snippet in a separate file sitting in a directory. I provide a “multiplot” shell script that lets me superimpose an arbitrary number of snippets, the result of which appears in figure 4. There’s a single off-scale peak appearing twice (because it’s in overlapping snippets), which I’ve suppressed.

\(^5\) Useful references: Taylor, *An Introduction to Error Analysis*, University Science Books, 1982 (the train-wreck book); Keeping, *Introduction to Statistical Inference*, van Nostrand, 1962; rpt. Dover, 1995. The standard math library, `-lm`, contains an `erf()` function; several libraries also contain `erf_c()` functions. Since calling conventions may differ, be careful of the order in which you link the libraries.
I may have been too conservative in my \( \alpha \): all the triggers appear to have found spikes, which means I’ve probably missed some.

**Sorting**

The next step is to find some set of criteria on the basis of which to sort. One approach (employed by some commercial spike-sorting programs) is to pick eight or so heuristic parameters, such as depth of the negative peak and heights of the flanking positives. Each spike is then reduced to a single point in an eight-dimensional space. The hope is to find clusters in this space. For visualization purposes, it’s best to take a two-dimensional projection of the space. The simplest such projections are along pairs of axes; for example, a program might simply plot each spike as a dot whose \( x \) component would give the minimum voltage and whose \( y \) component would give the time lag to the peak on the right. More generally, however, the two axes could be linear combinations of the original coordinates. After storing the eight (or however many) parameters in an \( N \times 8 \) matrix, \( N \) the number of spikes, practitioners use a principal-component analysis to determine the directions in the multidimensional space (eight-dimensional in the example above) that account for as much as possible of the variation among spikes.\(^6\)

**Wavelet transforms**

Instead of heuristic measures or principal components based on statistics, I chose two wavelet components of the spikes. A wavelet transform, like the Fourier transform, describes a signal in a different form: in both cases, if I start out with a time series of \( N \) points, the transform also consists of \( N \) points. (In the Fourier case, these may be complex, but for the wavelets, they’re real.) In both cases, I can transform back to recover my original set of \( N \) points. Consider

\(^6\) H. Hotelling, *J. Educ. Psych.* 24 (1933) 417. Didactic treatments may be found in textbooks on multivariate statistics or on clustering, as in M.R. Anderberg, *Cluster Analysis for Applications*, Wiley, 1973 §5.1.3.3. The idea is simply that covariance-matrix eigenvectors with successively smaller eigenvalues correspond to uncorrelated linear combinations of the original variables with successively smaller variances. This relates to singular-value decomposition, for which see *Numerical Recipes*, chapters 2 and 15.
the original data and their discrete Fourier transform. Each Fourier component represents the complex amplitude of waviness at a given frequency; any Fourier component transformed back by itself is a sine wave, infinite in extent but extremely localized in frequency. Conversely, the original representation of the data can be thought of as a set of amplitudes multiplying delta functions, which are extremely localized. The wavelet transform lies in between, somewhat localized in space and somewhat localized in frequency, but not extremely localized in either sense. I refer you to Numerical Recipes for an introduction.\footnote{For a thorough reference, see Meyer, Ondelettes, transl. Ryan, Wavelets: algorithms and applications, SIAM, 1993.}

The wavelet algorithm is fractal, picking out important features at different time scales. Roughly speaking, where a given Fourier component tells me how much sine-wave of a given frequency there is in the signal, a given wavelet component tells me how much of a hump there is of a given width starting in a given sector of my data. From this intuitive description, it ought to be a good way of sorting out spikes.

To sort spikes, I used the Daubechies-4 wavelets described in Numerical Recipes, adapting their \texttt{wt1} and \texttt{daub4} routines by re-writing them in C (instead of Fortran-translated-badly-to-C). For each 50-point snippet, the wavelet transform returned 50 wavelet components. I wrote a few scripts to determine for each snippet which five components were most important (largest). I then chose the two indices that occurred most often in the top five lists, plotting these two wavelet components against each other. This yielded figure 5a. While not in any way optimized, the obvious clustering suggests that I’ve succeeded in sorting two neural signals.

\begin{figure}[h!]
\centering
\includegraphics[width=\textwidth]{figure5}
\caption{(a) A scatter plot of the two principal Daubechies-4 components for 208 identified spikes; (b) a random selection from the two clusters, with those from the lower left plotted in a solid line and those from the upper right dotted.}
\end{figure}

Note in figure 5b several features (overall depth, a gap, location of right shoulder) that distinguish the two identified clusters. If we had more time for this project, I’d recommend doing statistical analyses to ensure that the apparent clustering in figure 5 could not be accounted for by sampling error (discrete measurement times) or by some other accident.\footnote{The CDAT-16 digitizer employs 64-times oversampling, significantly reducing the effect of sampling error or aliasing.}

Shortly after the first version of this course was offered, Letelier and Weber found that a Daubechies-8 transform did a better job of spike sorting than traditional principal-component methods.\footnote{J.C. Letelier and P.P. Weber, “Spike sorting based on discrete wavelet transform coefficients,” \textit{J. Neurosci. Meth.} 101 (2000), 93.}
Digression on wavelet filtering

To get a feeling for how much better suited to neural spikes wavelets are compared to Fourier components, I applied the Daubechies-4 wavelet transform to the entire data set, then threw away (set to 0) the 90% of wavelet components that were smallest before transforming back. Noise was visibly reduced without affecting the lineshapes of the spikes at all. I certainly would not want to try the same thing with a Fourier transform: all I’d have left would be noise. I did compare the Daubechies filtering to a Fourier high-pass filter that cut out everything below 500 Hz and let everything above 1 KHz pass. As already noted, localized spikes have significant weight at every frequency, so the high-pass filter severely degraded them.

Autocorrelation

Now that you’ve identified the neurons in the time series, you can start looking for temporal patterns. For example, is one neuron firing in a particularly ordered way? Correlation functions—see Numerical Recipes for definitions—also provide a check on the sorting procedure, since one expects a neuron to be somewhat better correlated with itself at a different time than with its neighbors (although neighbors are probably also correlated). You could compare the autocorrelation functions of the different sequences you’ve identified to the autocorrelation of the whole data set, to correlations between neurons, and to autocorrelations for parts of the data (say, the first and second halves). You can now replace the actual waveforms with Kronecker delta functions at their trigger points (i.e., 1 “at” each spike, 0 everywhere else).

Conclusions

I’ve indicated several places where I could have done a more thorough job; in particular, it is likely that I’ve missed (failed to trigger on) some spikes, and there may very well be more than the two neurons I’ve identified (not counting a solitary outlier). I’ve suggested more ideas than anyone will have time to explore in two weeks. I hope each student will pick a different set; it will be interesting to compare results.