Spike Sorting

This example was modified from: Gabbiani & Cox, Mathematics for Neuroscientists

SpikeSort.m: Sorting spikes from noisy multi-unit extracellular recordings.

MATLAB concepts covered:
1. thresholding to find peaks in data
2. using 'diff' to find events in continuous data
3. PCA/SVD to reduce dimensionality of data
4. 3D plotting
5. ISI histograms

In the previous exercise, we were dealing with the spike rates produced by a single identified neuron as recorded with an extracellular electrode. Now we're going to take a step backwards in the data analysis process. The raw data recorded from an extracellular electrode usually contains "action potentials" (a.k.a. "spikes") from multiple different neurons plus noise. So before we can measure spike rates, we need to identify spikes from single neurons. We do this by comparing the waveforms of the individual spikes: the spike from a given neuron that is a fixed difference from the electrode will have a relatively stereotypical waveform; since different neurons will have different spatial relationships with the electrode, their waveforms will differ. Our task here is to identify clusters of shapes (waveforms). To do this we first need to separate the continuous voltage trace into discrete events (= spikes) and align them. As you'll see below, each spike will consist of 601 data points, which consist of the digitized voltage over a period of 12 ms. This means that each spike has 601 dimensions: we need 601 numbers to describe it. But there exist many elegant methods for reducing the number of dimensions that we need to describe things. One of these is known as "principle components analysis" ("PCA" for short). Essentially what this does is search for linear combinations of our original dimensions that maximize the explained variance in the data. In this exercise, we will use "singular value decomposition" (SVD) to perform PCA on our spike waveforms. The linear algebra behind SVD is actually quite hairy and way beyond the scope of this course. In class, I'll give a brief demonstration in two dimensions that should give you some intuitive notion of how PCA works. But our ignorance of the mathematics behind SVD does not necessarily hold us hostage, since we can generate arbitrary data (where we know what the right answer is) add varying amounts of noise and then use this fake data to see whether the mysterious algorithm ('svd' in this case) is behaving in the desired way.

Load in some data recorded using an extracellular electrode:

```
load SpikeSortData;
```

This file contains 3 variables:

> 'Time' is the time-base, from 0 to 3000 ms in 0.02 ms steps.
> 'Vtotal' is the digitized voltage trace of all spikes.
> 'Vspikes' is the data from the individual neurons, with each column representing the voltage trace of the spikes generated by one neuron.

Ex. 1: In one figure, make separate plots of the three individual neurons and the total trace.

The data are very condensed, so use the zoom tool to expand parts of the trace. You should be able to see action potentials ('spikes') of three different sizes. In this case we know that there are three, because we generated them ourselves. We will use this knowledge below to see how well our spike-sorting algorithm has performed.

Ex. 2: Compare the sum of the individual spikes with the trace in Vtotal.

What do you notice about the variability in spike heights comparing the two? Why might this be?

In order to sort and analyze the individual spikes, the first thing we need to do is break up the "continuous" voltage trace into discrete events containing single spikes. Looking at the raw voltage trace, we see that spike events are generally much bigger than the noise. We also notice that there are some very large events that are clearly bigger than any of the spikes we generated. What is going on here? These represent the superposition of two spikes, which is rare in real recordings.

Ex. 3: Identify contiguous blocks of data that belong to the single spikes. Plot this on top of the raw voltage trace (Vtotal).

How can we break these up into discrete events? If you zoom in on this new trace, you'll see that all of the green dots within a single spike are neighbors, and that there are gaps between the spikes. Another way to see what's going on is to actually plot our indices. From a distance, this looks like a smooth line, but if we zoom in, we'll see that it is jagged, with smooth, nearly horizontal lines broken by sudden vertical jumps.

Ex. 4: Identify the indices of the beginning of each thresholded spike.

We now have a list of start-points for candidate spikes. Now we need to plow through this list of events and grab some actual data (remember that thus far we have been working with indices!) from both sides of the index. From looking at the raw traces, we see that our spikes are generally less than 10 ms in width, so, to be safe, let's grab 3 ms before and 9 ms after each event marker.

Ex. 5: Generate an n x 601 array called 'SpikeData' that will contain all of the spikes; each spike's data in one row. Exclude events whose peak is greater than 4 mV. Plot the data.

Once we've got our spikes aligned in one big array, we can use singular value decomposition to help with the sorting. Basically, you can think of each spike as a vector of 601 dimensions. That is, each spike is uniquely defined by 601 numbers consisting of the voltage value at each of 601 time points. But perhaps we can effectively describe our spikes with fewer dimensions. We'll use SVD to find new coordinates that maximize the variability accounted for in our spikes. We first need to transpose the array for subsequent analyses. Now each spike is a single column and time runs along the rows. The details of how SVD works are well beyond the scope of this course. For now, just use it and see what it does!

```
SpikeData = SpikeData';
SpikeNum = size(SpikeData,2);
SpikeData = SpikeData - repmat(mean(SpikeData,2),1,SpikeNum);
[Y,Sig,X] = svd(SpikeData);
sig = diag(Sig);
figure; semilogy(sig(sig>1),'kx-')      % plot the significant singular values
xlabel('index','fontsize',14); ylabel('singular value','fontsize',14)l
```

This is interesting. The singular value, in this case, gives us a measure of how much of the variance each new vector is accounting for. We can see that the first 3 or 4 account for a lot, and subsequent ones not so much. In order to see where each spike falls in our new space, we plot each spike according to its projection on the first three principle components. To do this, we just multiply our SpikeData matrix by the vector defining each of the first three principle components. This is matrix multiplication, not element multiplication:

`SpikeData'` is 98 x 601

`Y(:,1)` is 601 x 1 and corresponds to the coordinates of the 1st principle component

so the matrix product is 98 x 1

Essentially, the 601-vector defining the 1st principle component is multiplied by the time series for each trace to get one value (the magnitude of the projection of that trace onto the 1st principle component) for each spike. Spikes that have a similar projection onto the 1st component will be plotted near each other.

Ex. 6: Calculate the projections of all neurons onto the 1st three principle components and plot them in three dimensions.

Use the 'Rotate 3D' tool on the figure to see where each cell falls in this new space. You can see pretty clearly that the data fall into clusters (with a few stragglers). In most modern "Spike-sorting GUIs",one can use the mouse to draw circles around clusters of points in the PCA-space and thus define the range of values that will be accepted as a spike belonging to one neuron. In this case, we know which spikes belong to which neurons, because we generated them ourselves. We can thus go back in and calculate the projections for each spike from each known neuron and then color code them.

Ex. 7: One reality check for spike sorting is to look at the interspike intervals. How might this help? What does this tell you about the way the spikes were generated?