An Exact Jitter Method using Dynamic Programming

Matthew Harrison
Division of Applied Mathematics
Brown University
Providence, RI 02912 USA
Matthew_Harrison@Brown.EDU

Stuart Geman
Division of Applied Mathematics
Brown University
Providence, RI 02912 USA
geman@dam.brown.edu

March 8, 2004

APPTS Report #04-3, Division of Applied Math, Brown University
http://www.dam.brown.edu/ptg/REPORTS/04-3.pdf

1 Introduction

We are given a neural spike train with spike times $t_1, \ldots, t_n$ and we want to discover something about the resolution over which these spike times were generated. A heuristic approach is to fix a statistic of the spike train, say $f(t_1, \ldots, t_n)$, and then Monte Carlo jitter each of the spike times by some small random amount, computing the statistic over and over again for each jitter. Presumably, if the jitter amount is smaller than the meaningful resolution of the spike train, then the original value of the statistic will not be an outlier in the distribution of jittered statistics. The jitter heuristic is designed to create a lot of "similar" spike trains in the sense that it preserves the "rate of spiking" measured over windows larger than the jitter amount. Failing to account for a non-constant firing rate has been the downfall of many statistical methods which look at the time resolution of spike trains.

A close analysis of the jitter method reveals several drawbacks. Perhaps the easiest to see is that the jittered spike trains may not obey known physiological constraints, such as the absolute refractory period. Furthermore, spikes may swap positions, which is intuitively unsettling at the very least. Han Amarasingham's thesis [1] contains a detailed discussion of various jitter methods, including an exact (not Monte Carlo) method that includes physiological constraints in an agnostic way. Unfortunately, this exact method is computationally intensive and may not be appropriate for online physiological experiments where a jitter method is applied to data from one trial in order to generate stimuli for the next trial.

Here we describe a fast and exact jitter method that allows for some predefined physiological constraints. We feel that it is a reasonable compromise between the well formulated and agnostic methods in [1] and the obviously deficient original jitter heuristic. We will first describe a Monte Carlo formulation of our method. Then we will show that it can be solved exactly and quickly using dynamic programming. That our method is exact and is not a Monte Carlo method is important for reducing the variance of our estimates. Reduced variance translates into reduced numbers of trials in physiological experiments.

2 Monte Carlo Constrained Jitter

The original jitter method effectively jitters all spikes simultaneously, irrespective of each other. Of course, we can think about this as jittering the first spike, then jittering the second spike, and so on. The main thing is that the amount of jitter for the second spike did not depend on the amount of jitter for the first spike. This is easy enough to modify, though.

First, we jitter the initial spike according to some initial distribution. Maybe this initial distribution is uniform over some small time window, but it need not be. Then, given the jittered position of the first spike, we jitter the second spike using a distribution that captures some known physiological constraints. These constraints can be hard: we might want to prevent the second spike from preceding the first, or we might want to prevent it from falling into the absolute refractory period of the first spike. They can be soft: we might want to have only a small probability that the second spike falls into the relative refractory period of the first spike, or we might want the jitter probability to obey a given distribution of interspike intervals. They can even vary from spike to spike: we might want to preserve bursting, so if the second spike closely followed the first in the original spike train, then we can have the jitter probability preserve this closeness.

Once we have jittered the second spike, then we move on. Given the jittered position of the second spike, we jitter the third spike according to some distribution. This continues until all the spikes are jittered. The important thing for our exact method detailed below is that the jittered position of a spike only depends on the jittered position of the previous spike. This can be relaxed to depend on the jittered position of the previous m spikes, but m will need to be very small for the algorithm to be fast.

This method for jittering spikes is more general than the original jitter method. It allows us to incorporate certain structure into the jittered train. Suppose we repeated this process over and over to create many jittered versions of the same spike train. If the original spike train was produced by some process operating at a resolution much coarser than our jitter, except for the physiological constraints that we included, then the underlying assumption is that jittered spike trains will show up with about the same relative frequency that we might expect if we were to run the "identical" physiology experiment over and over again, whatever that means. This is the null hypothesis.

This particular null hypothesis is not mathematically well-defined, but it captures the intuition that we are interested in. Mathematically well-defined or not, the jitter process (which is well-defined) effectively creates a null-hypothesis which we can test using a permutation test. The Monte Carlo version of this test is simple. Create many jittered spike trains. Compute the same statistic on all of them (we haven't discussed choice of statistics, yet) and create the distribution of these statistics. The p-value of the test is the tail probability of the distribution at the value of the statistic for the original spike train. That is, the p-value is how much the original train is an outlier.

2.1 Choosing the statistic

In principle the statistic is any function f of the entire spike train. For our exact method to work, the function must be of an appropriate form. In particular, we will assume that

$$f(t_1, \ldots, t_n) = \sum_{k=1}^{n} f_k(t_k),$$

that is, f evaluated on a spike train with many spikes is just the sum of a sequence of functions f_k each evaluated on individual spikes.

Within this additive constraint, we are free to choose f. We believe that a particularly powerful f for rejecting the null-hypothesis will involve some sort of synchrony measure between the jittered spike and a fixed, comparison spike train. For example, let $\tilde{t}_1, \ldots, \tilde{t}_{\tilde{n}}$, be the spike times of another spike train, perhaps recorded at the same time as the original spike train. A common measure of synchrony is

$$f_k(t) = g(t) = \mathbb{1}\left\{\min_{1 \le j \le \tilde{n}} |t - \tilde{t}_j| \le \Delta\right\},$$

where $\mathbb{1}\{A\}$ is the indicator function of the event A. The particular application will dictate the specifics of the test statistic.

3 Exact Constrained Jitter

As we run more and more Monte Carlo simulations, the empirical distribution and, thereby, the p-value converge to some limiting values. We have formulated this jitter method so that these limiting values are easily and quickly obtainable. The key insight is that the sequence of spike times in a jittered train are the realizations of a Markov chain. Perhaps this was apparent from our original description of the method, but we will go through some of the details.

We have an observed spike train with times $t_1, \ldots, t_n$. These are fixed and known. Each of these spikes can be jittered by a certain amount. Let us say that spike k can be jittered to any one of the times $S_k = \{s_{k1}, \ldots, s_{kM_k}\}$. For example, we might have $S_k = \{t_k - 10, t_k - 9, \ldots, t_k + 10\}$ for a ± 10 ms jitter. The S_k must be finite (we do not allow continuously valued jitter), but this is not a problem in practice. Note that the method really only makes sense if $t_k \in S_k$.

The S_k will be the sequence of state spaces for our Markov chain. In many cases, $S_k = t_k \oplus \{\delta_1, \ldots, \delta_M\}$, for all k, in which case we can make the state space the same for each time step in the Markov chain, but this is not really important.

A realization from the Markov chain is $X_1, \ldots, X_n$, with $X_k \in S_k$, corresponding to the jittered spike times. We begin the jitter process with some initial distribution μ for X_1 over S_1. Once X_1 is selected, then we use the known (that is, made up according to reasonable physiological constraints like refractory period) probability transition matrix $\mathbb{P}_1(x_2|x_1)$ to generate $X_2 \in S_2$ given the value of $X_1 \in S_1$. Once X_2 is chosen we continue on using our sequence of transition probability matrices $\mathbb{P}_k$, $k = 1, \ldots, n-1$. Notice that the $\mathbb{P}_k$ can be designed from the original spike times and can thus preserve certain structure like bursting.

For each jittered spike X_k we compute some deterministic statistic $Y_k = f_k(X_k)$. The statistic of the entire jittered spike train is

$$Z = \sum_{k=1}^{n} Y_k = \sum_{k=1}^{n} f_k(X_k) = f(X_1, \ldots, X_n).$$

In the next section we describe a dynamic programming approach for computing the distribution of Z. Once this distribution has been computed, the p-value is again just the tail probability at the original value of the statistic on the unjittered spike train. Our initial simulation experiments indicate that this dynamic programming algorithm

is sufficiently fast (< 30 ms in Matlab over typical parameter ranges) for use in online, adaptive neurophysiological experiments.

3.1 The distribution of sums of functions of a Markov chain

Let $X_k \in S_k$, $k = 1, \ldots, n$, be a finite state, first order, time inhomogeneous Markov chain with initial distribution μ for X_1 and with probability transition matrices

$$\mathbb{P}_k(s|s') = \text{Prob}\left\{X_{k+1} = s | X_k = s'\right\}.$$

Let $Y_k = f_k(X_k)$ for deterministic functions $f_k : S_k \to A_k \subset \mathbb{R}$. Notice that $|A_k| \leq |S_k| < \infty$. Define $Z = \sum_{k=1}^n Y_k$ and let p be the distribution of Z over its support $B \subset \mathbb{R}$. Since there are at most $\prod_{k=1}^n |A_k|$ outcomes for the sequence $(Y_1, \ldots, Y_n)$, we know that $|B| \leq \prod_{k=1}^n |A_k| \leq \prod_{k=1}^n |S_k| < \infty$. If the A_k are appropriately chosen, then $|B|$ will typically be significantly smaller than this bound.

We are given the distribution μ, the transition matrices $\mathbb{P}_k$ and the functions f_k and we want to compute p, the distribution of Z. Define

$$W_k(s, b) = \text{Prob}\left\{X_k = s, \sum_{j=1}^k Y_j = b\right\},$$

$$B_k = \left\{b : \sum_{s \in S_k} W_k(s, b) > 0\right\}.$$

B_k is the support of W_k. We can iteratively compute W_k and B_k, $k = 1, \ldots, n$, using a dynamic programming procedure. Notice that

$$p(z) = \text{Prob}\left\{\sum_{k=1}^n Y_k = z\right\} = \sum_{s \in S_n} \text{Prob}\left\{X_n = s, \sum_{k=1}^n Y_k = z\right\} = \sum_{s \in S_n} W_n(s, z)$$

and has support $B = B_n$. So computing W_n and B_n will solve our problem.

First notice that

$$W_1(s, b) = \text{Prob}\left\{X_1 = s, Y_1 = b\right\} = \text{Prob}\left\{X_1 = s\right\} \mathbb{1}\left\{f_1(s) = b\right\} = \mu(s) \mathbb{1}\left\{f_1(s) = b\right\},$$

$$B_1 = \left\{b \in A_1 : \sum_{s \in S_1} W_1(s, b) > 0\right\}.$$

Both of these are easy to compute.

Now, given W_k and B_k we can compute W_{k+1} and B_{k+1} as follows. First, notice that $B_{k+1} \subset B_k \oplus A_{k+1}$, the set of all possible values resulting from a sum of one element from B_k and one from A_{k+1}. For each $b \in B_k \oplus A_{k+1}$ and $s \in S_{k+1}$ we can compute

$$\begin{aligned}
W_{k+1}(s, b) &= \text{Prob}\left\{X_{k+1} = s, \sum_{j=1}^{k+1} Y_j = b\right\} = \sum_{s' \in S_k} \text{Prob}\left\{X_{k+1} = s, X_k = s', \sum_{j=1}^{k+1} Y_j = b\right\} \\
&= \sum_{s' \in S_k} \text{Prob}\left\{X_{k+1} = s, X_k = s', \sum_{j=1}^{k} Y_j = b - f_{k+1}(s)\right\} \\
&= \sum_{s' \in S_k} \text{Prob}\left\{X_{k+1} = s | X_k = s'\right\} \text{Prob}\left\{X_k = s', \sum_{j=1}^{k} Y_j = b - f_{k+1}(s)\right\} \\
&= \sum_{s' \in S_k} \mathbb{P}_k(s|s') W_k(s', b - f_{k+1}(s)).
\end{aligned}$$

Once W_{k+1} has been computed, we can just use the definition to find B_{k+1}.

Note that if X_k is not first order, then we can always create a new sequence that is first order by expanding the alphabet.

References

[1] Asohan Amarasingham. PhD thesis, Division of Applied Mathematics, Brown University, 2004.