Van Rossum Distance with Optimal Lag

Edmund Butler

December 4, 2022

1 Introduction

A spike train is a sequence of times representing neural spikes recorded over a short time interval. It can be important to determine whether two recorded trains are "the same", perhaps meaning responses to the same stimulus. A number of algorithms have been proposed for this problem; the most biologically motivated is the van Rossum distance [1], which might represent the difference in a synapse's response to two spike trains.

The standard van Rossum distance is defined as follows. Let $S = \{s_i\}, i = 1, ..M$ and $T = \{t_j\}, j = 1, ..N$ be the increasing times of two spike trains and let τ be the characteristic time for the response to a spike. Assume the responses sum in time. Let $\mathbf{1}()$ be the indicator function (0 or 1 as the argument is false or true), then the responses are idealized as follows. The van Rossum transform of the spike train $S = \{s_i\}$ is a real function $\mathfrak{R}_{\tau,S}$ defined by Equation 1 below, where τ is a positive parameter.

$$\mathfrak{R}_{\tau,S}(u) = \sum_{i=1}^{M} e^{-(u-s_i)/\tau} \mathbf{1}(u > s_i) \tag{1}$$

This expression decays exponentially in u, so is in L^p for any p. The van Rossum distance, D, is the L^2distance between $\mathfrak{R}_S$ and $\mathfrak{R}_T$:

$$D^2 = \int_{-\infty}^{\infty} (\mathfrak{R}_{\tau,S}(u) - \mathfrak{R}_{\tau,T}(u))^2 du \tag{2}$$

The distance may be computed with linear complexity [2].

In this document the parameter τ is usually omitted and left to context. Unless specified, the limits of integration are always -∞ and ∞.

Often the trains S and T are not directly comparable because of different time bases, so the lagged distance is more useful.

$$D^2(c) = \int (\mathfrak{R}_S(u) - \mathfrak{R}_T(u - c))^2 du \tag{3}$$

Then the optimal lag, c_{opt}, minimizes $D(c)$ and the optimal distance is $D(c_{opt})$. This document develops a new algorithm for finding the exact value of c_{opt} and computing $D(c_{opt})$.

1.1 Why use van Rossum at all?

Several other spike train distances have been proposed and the van Rossum distance is not always the recommended alternative. For example, a recent article of Satuvuori and Kreuz [3] looks at five distance algorithms: Victor-Purpura, van Rossum, ISI, SPIKE and RI-SPIKE. The authors distinguish between rate coding and temporal coding, and find that the first two ("spike-resolved") perform poorly at detecting temporal coding except at very low spike rates. For the Van Rossum distance, "low spike rates" means values of τ much less than the mean inter-spike time, so the Van Rossum transform is essentially a sum of delta functions - not so attractive.

On the positive side, the van Rossum transform is directly relevant to some neural simulation codes. Ref???. Also, the ability to calculate a precise lag value has independent interest. The lag based on any distance can of course be approximated, but perhaps only with significant implementation complexity, performance cost, or loss of accuracy.

The deficiencies noted by Satuvuori and Kreuz can in part be overcome by normalization. If one considers two spike trains, S and T, very similar except that T has twice the spike rate, then the van Rossum distance of Equation 2 is insensitive to the detailed shapes of the spike trains, simply because $\mathfrak{R}_T \approx 2\mathfrak{R}_S$. If one wishes a distance insensitive to rate but sensitive to time information one can normalize the van Rossum distance.

$$D^2_{norm} = \int (\mathfrak{R}_S/M - \mathfrak{R}_T/N)^2$$

A follow-on to this document may compare performance of the normalized van Rossum distance with other distances, based on simulations with multiple bursts of spikes and varying base spike rates. One guesses that the answer to "What spike train distance is best?" depends on details of the question.

2 The Optimal Lag

This section proves that there are only finitely many possible choices for c_{opt} and develops an efficient algorithm for finding it. Without loss of generality, $\tau = 1$ is assumed.

2.1 The Theorem

The main theorem asserts that there are only a finite number of possibilities for c_{opt}, obtained by matching a particular s_i with a particular t_j. Some formulas developed during the proof of the theorem form the basis for the algorithms.

2.1.1 Lemma 1

Let $\mathfrak{R}_S()$ and $\mathfrak{R}_T()$ be as above, then

$$\int \mathfrak{R}_S(u)\mathfrak{R}_T(u)du = 1/2 \sum_{i,j} e^{-|s_i - t_j|} \tag{4}$$

Proof

Multiply the product of sums term by term:

$\mathfrak{R}_S(u)\mathfrak{R}_T(u) = \sum_{i,j} e^{-(2u - s_i - t_j)} \mathbf{1}(u > \max(s_i, t_j))$

Take the integral inside the sum and use:

$\int_{t=a}^{\infty} e^{-2(u-b)} du = 1/2 e^{-2(a-b)}$

Substitute $b = (s_i + t_j)/2$ and $a = \max(s_i, t_j)$, and use $\max(s,t) - (s+t)/2 = |s-t|/2$:

$\int \mathfrak{R}_S(u)\mathfrak{R}_T(u)du = 1/2 \sum_{i,j} e^{-|s_i - t_j|}$□

2.1.2 Corollary

$$\int \mathfrak{R}_S^2 = \sum_{1 \le i \le j \le N} e^{s_i - s_j} - N/2 \tag{5}$$

Proof

Substitute $\mathfrak{R}_S$ for $\mathfrak{R}_T$ in Equation 4 and rearrange terms.□

2.1.3 Lemma 2

Let X be a finite set of reals, then $\sum_{x \in X} e^{-|x-c|}$ is maximized over all reals by some $c \in X$.

Proof

Consider $F(c) = \sum_{i=1}^{N} e^{-|x_i - c|}$, where $\{x_1, ...x_N\} = X$.

$F()$ is positive, continuous and tends to 0 at $\pm\infty$, so it must have a maximum.

For any c not in X, $F''(c) = F(c) > 0$. So no such c can be a maximum. The result follows.□

2.1.4 Theorem

The minimum for $D(c)$ is taken on by some c of the form $s_i - t_j$. [1]

Proof:

We can write

$$D^2(c_{opt}) = \int \mathfrak{R}_S^2 + \int \mathfrak{R}_T^2 - 2\max(c) \int \mathfrak{R}_S(u)\mathfrak{R}_T(u - c)du \tag{6}$$

so $c_{opt} = \text{argmax}(c) \int \mathfrak{R}_S(u)\mathfrak{R}_T(u-c)du$, which by Lemma 1 is $\text{argmax}(c) \sum e^{-|s_i - t_j - c|}$, so Lemma 2 shows $c_{opt} = s_i - t_j$ for some i and j.□

2.2 The Algorithms

This section develops recursive algorithms to calculate the terms in Equation 6.

[1] The proof works for any convex kernel, not just $\mathbf{1}(x > 0)e^{-x}$

2.2.1 Algorithm 1 - VRDnormSQ

The goal is to calculate $\int \Re_S^2$ using the representation in Equation 5. In order to calculate the sum, create a recursion for a numerically accurate calculation.

Algorithm VRDnormSQ
Input: S - vector of increasing times
Output: $\int \Re_S(u)^2 du$
Step 1. Compute $A_k = \sum_{i|i\leq k} e^{s_i - s_k}$ for $k = 1, ...N$ by the recursion:
$A_1 = 1, A_{k+1} = 1 + A_k e^{s_k - s_{k+1}}$
Step 2. Return $\sum_{k=1}^{N} A_k - N/2$

2.2.2 Algorithm 2 - VRDcorr

The goal is to calculate $\max(c) \int \Re_S(u)\Re_T(u-c)du$, together with the maximizing c, using the representation in Equation 4. First we sort the set of all differences between s and t. This defines the nondecreasing sequence $X = x_1, ..., x_{MN} = sort(\{s_i - t_j\})$. Equation 4 becomes

$$2\int \Re_S(u)\Re_T(u-c)dt = \sum_{x\in X} e^{-|x-c|}$$

Since $c \in$X, this can be written

$$2\int \Re_S(u)\Re_T(u-c)du = \sum_{x|x<c} e^{x-c} + 1 + \sum_{x|x>c} e^{c-x} \tag{7}$$

If we define

$$A_k = \sum_{i|i\leq k} e^{x_i - x_k}, B_k = \sum_{i|i\geq k} e^{x_k - x_i}$$

it is sufficient to maximize $A_k + B_k$, then $c_{opt} = x_{k_{max}}$.

This is essentially a quadratic algorithm (really O($MN \log(M+N)$) because of the sort.) So it is more expensive than the fast algorithm without a lag (linear), but if you need the lag, it is less expensive than trying out all MN possible lags (cubic.)

Algorithm VRDcorr
Inputs: Two increasing sequences, S and T.
Outputs: The optimal lag, C, and the correlation Corr=$\int \Re_S(u)\Re_T(u-C)$
Step 1: Compute $\{x_k\}$= sort($\{s_i - t_j | i \leq M, j \leq N\}$)
Step 2: Compute $A_k = \sum_{i=1}^{k} e^{x_i - x_k}$ for k=1,...,MN by the recursion:
$A_1 = 1, A_{k+1} = 1 + e^{x_k - x_{k+1}} A_k$
Step 3: Compute $B_k = \sum_{i=k}^{MN} e^{x_k - x_i}$ for $k = MN, ..., 1$ by the recursion:
$B_{MN} = 1, B_{k-1} = 1 + e^{x_{k-1} - x_k} B_k$
Step 4: Compute $k_{max} = \text{argmax}(k) A_k + B_k$
$C = x_{k_{max}}$
$Corr = (A_{k_{max}} + B_{k_{max}} - 1)/2$

2.2.3 Algorithm 3 - VRDfastCorr

The algorithm VRDcorr automatically applies the optimal lag, but it can be useful to compute a correlation without lag, especially if that computation is much faster. Algorithm VRDfastCorr uses essentially the same logic as reference [2] to compute $\int \mathfrak{R}_S(u)\mathfrak{R}_T(u)du$ in time O(M+N).

The main idea is to precompute partial sums of exponentials in the t direction. With care to account for $s_i == t_j$ only once, write Equation 4 as

$$\int \mathfrak{R}_S\mathfrak{R}_{\mathfrak{T}} = 1/2(\sum_{i,j|s_i<t_j} e^{s_i-t_j} + \sum_{i,j|s_i\geq t_j} e^{t_j-s_i}) \triangleq 1/2(U+V) \tag{8}$$

Calculate $U_i = \sum_{j|t_j>s_i} e^{s_i-t_j}$ and $V_i = \sum_{j|t_j\leq s_i} e^{t_j-s_i}$ by the following recurrences.

Algorithm VRDfastCorr
Input: Two increasing sequences S and T.
Output: The correlation $\int \mathfrak{R}_S\mathfrak{R}_T$
Step 1: Calculate U_k by the recurrence:

$$U_N = \sum_{j|t_j>s_N} e^{s_N-t_j}; U_{k-1} = e^{s_{k-1}-s_k}U_k + \sum_{j|s_{k-1}<t_j\leq s_k} e^{s_{k-1}-t_j}$$

Step 2: Similarly, calculate V_k by the recurrence:

$$V_1 = \sum_{j|t_j\leq s_1} e^{t_j-s_1}; V_{k+1} = e^{s_k-s_{k+1}}V_k + \sum_{j|s_k\leq t<s_{k+1}} e^{t_j-s_{k+1}}$$

Step 3: Return $1/2\sum(U_k+V_k)$

2.3 Implementation

A C++ implementation has main entry point laggedVRD() with the arguments:

```
bool laggedVRD(sIn, tIn, tau,
               sNorm, tNorm,
               corr, lag)
Inputs: sIn, tIn - the spike trains
        tau - the scale
Outputs: sNorm, tNorm - L2 norms of the transforms
         corr - the maximized correlation
         lag - the maximizing lag
```

The outputs may be used to calculate the lagged distance

$$D^2 = sNorm^2 + tNorm^2 - 2 * corr$$

or the correlation coefficient

$$CC = corr/(sNorm * tNorm)$$

.

Also, the normalized VR distance, $D^2_{norm} \triangleq \int (\mathfrak{R}_S(u)/M - \mathfrak{R}_T(u)/N)^2 du$ is given by

$$D^2_{norm} = sNorm^2/M^2 + tNorm^2/N^2 - 2corr/(MN) \tag{9}$$

3 Simulation Experiments

Simulated spike trains were created to answer three questions:

1. Do the two correlation calculations get the same answer?
2. What is the CPU performance of the implementations?
3. Does the calculated value of c_{opt} depend strongly on the scale parameter τ?

The third question could be significant for the following reason. It turns out that the function VRDfastCorr() is much faster than VRDcorr(), so one would like to use VRDcorr() with a single value of τ to calculate c_{opt} and then use VRDfastCorr() to explore the effect of changes to τ; this strategy requires a stable value for c_{opt}.

3.1 The Simulation

The simulation creates base trains and noised trains. A base train consists of the times of a Poisson process with rate $1/\gamma$. A noise train takes a base train and modifies it in three ways, depending on small parameters α and β.

1. Dropping out a proportion α of the times.
2. Combining with the times of an independent Poisson process with rate α/γ.
3. Adding noise to each time. The noises are IID centered Uniform random variables with width $\beta\gamma$.

Step 2 adds about the same number of noise spikes as is dropped in Step 1. The net effect is that both base and noised trains have average inter-spike time τ. (So this simulation avoids rate-discrimination effects, to be addressed in a follow-on study.)

The values $\alpha = 0.1$, and $\beta = 0.03$ are always used in this document. Figure 1 shows 50 noised trains using the same base train with $\gamma = 0.025$. (The value $\gamma = 1$ is used for the rest of this document.)

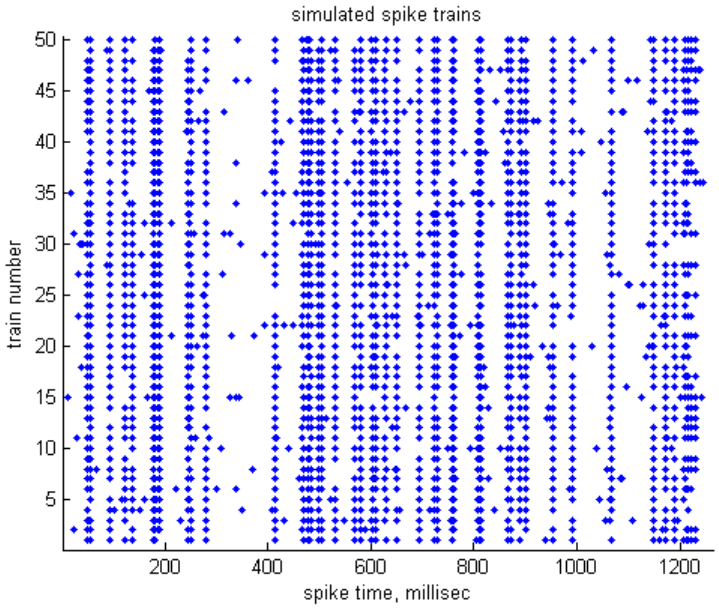

Figure 1 - Simulated spike trains, based on a 40 Hz original ($\gamma = 0.025$).

3.2 VRDcorr() and VRDfastCorr() Compute the Same Correlation

The two algorithms VRDcorr() and VRDfastCorr() compute correlations using different algorithms. The outputs may be compared as follows.

Let S and T be spike trains. VRDcorr() computes c_{opt} and $Corr = VRDcorr(S,T) = \int \mathfrak{R}_S(u)\mathfrak{R}_T(u-c_{opt})du$. Then translate the spike train T, $t_{opt}[i] = t[i] - c_{opt}$. A second version of the correlation is given by $Corr2 = VRDfastCorr(S, T_{opt})$.

The above procedure is done 10,000 times each for train lengths between 10 and 1,000. At each train length a Root Mean Square error is computed, based on $Err = Corr - Corr2$. The results are plotted in Figure 2. The increase in error is partly due to the larger number of terms in Equation 1, but in any case the error is negligible ($\leq 2 \times 10^{-12}$).

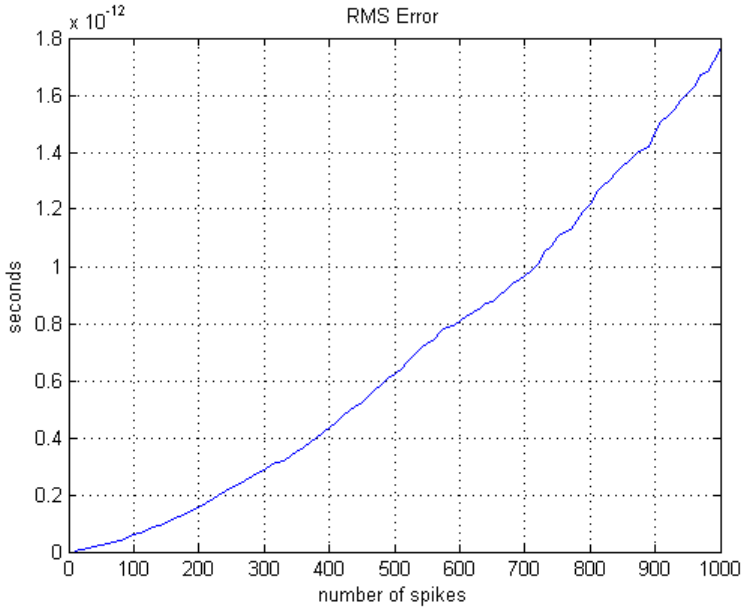

Figure 2. Errors in computation of correlations by VRDcorr() and VRDfastCorr()

3.3 CPU performance

Simulations were run under Windows 11 using an Intel 11th generation I9 processor. Executable were built using Microsoft Visual C++ 2022. (These performance times cannot be directly compared with older measurements because the support is so much better now.)

The simulation described in section 3.2 was instrumented to determine CPU timing. Predicted performance was that VRDcorr()'s CPU usage would grow approximately as N^2, while VRDfastCorr() should grow as N, where N is the number of spikes in a train. Figures 3 and 4 show the expected behavior.

More significantly, VRDfastCorr() is very fast, requiring only tens of microseconds for quite long trains. VRDcorr() is much slower; Figure 5 shows the ratio. VRDcorr() requires tens of milliseconds; as a practical matter, that speed would be adequate for many uses.

How expensive is the sort in VRDcorr()? The current implementation uses the sort from the C++ standard library. As displayed in Figure 6, this sort uses rather more than half of the CPU time. Most likely that could be improved by taking advantage of the special structure of the data.

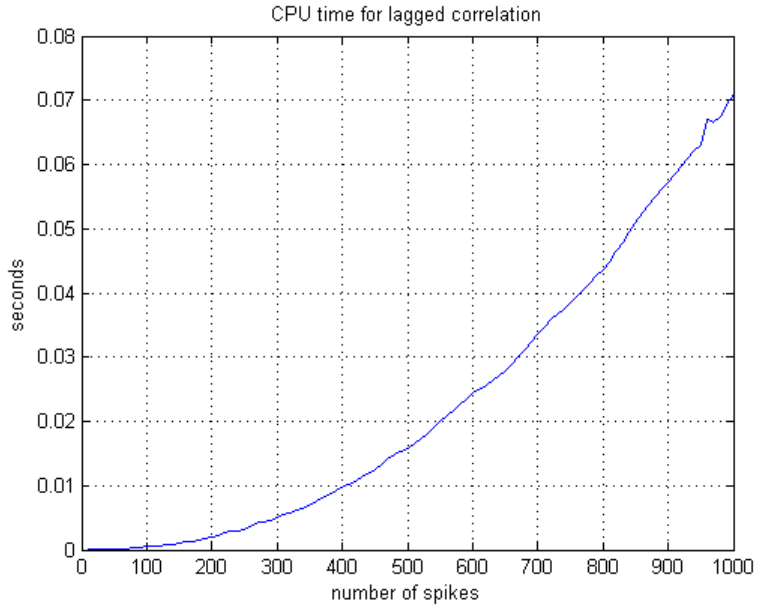

Figure 3. CPU time for a call to VRDcorr() as a function of the number of spikes

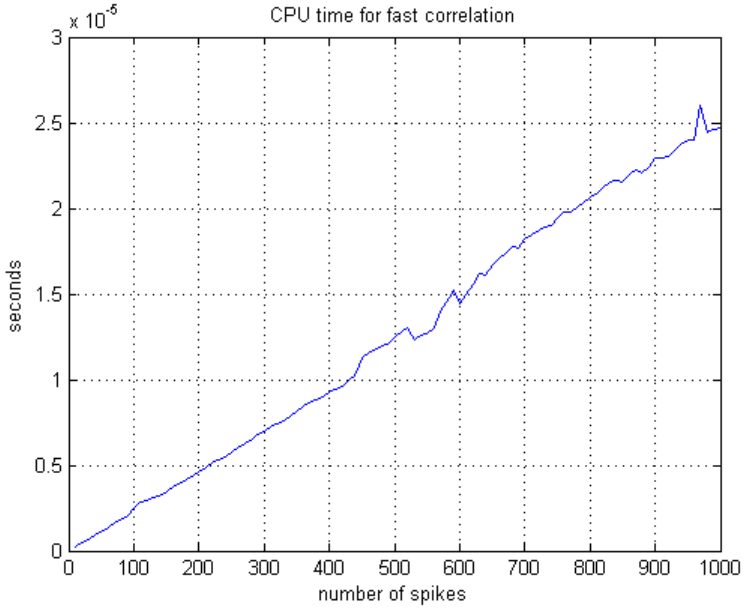

Figure 4. CPU time for a call to VRDfastCorr() as a function of the number of spikes

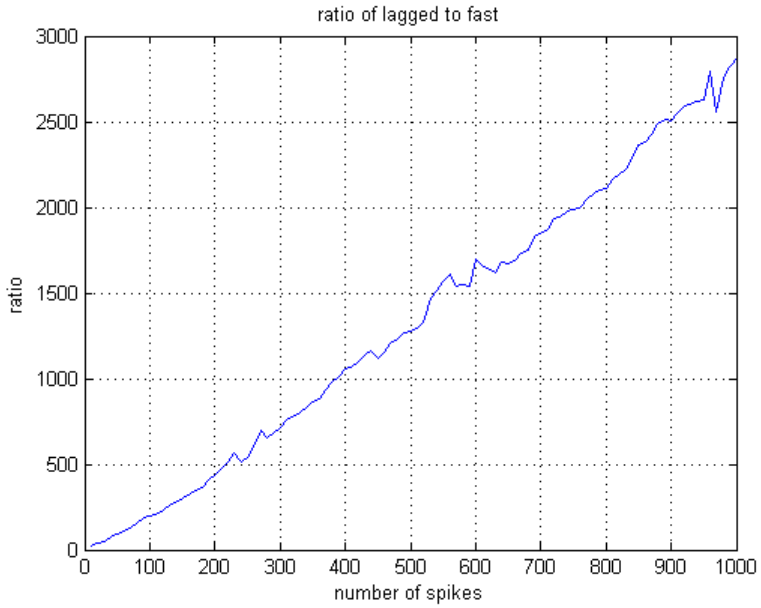

Figure 5. Ratio of CPU times: VRDcorr/VRDfastCorr

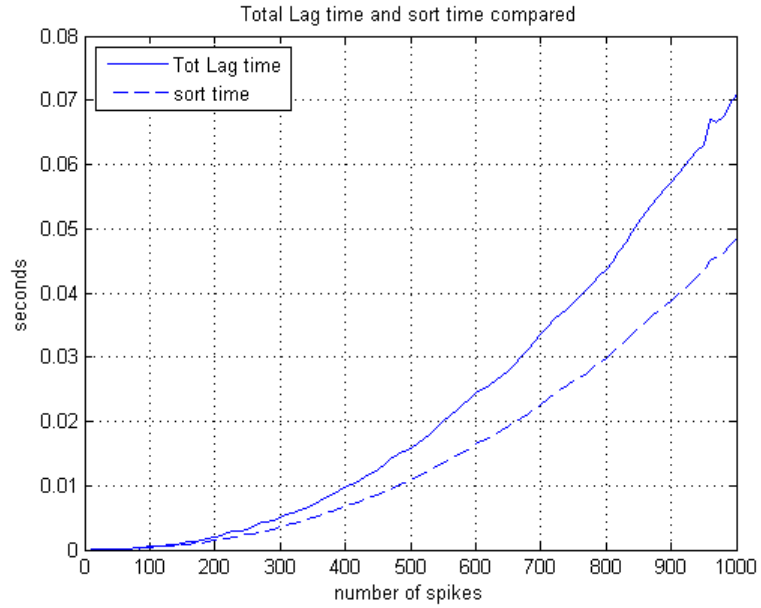

Figure 6. Comparing sort CPU time with total CPU time for VRDcorr

3.4 Dependence of Lag on Scale

How meaningful is the lag c_{opt}? When there is a known correct value for the lag, c_{opt} should well approximate that value. Also, the calculation of c_{opt} should not depend strongly on the value of τ used in the computation.

A simulation's results suggests the algorithm VRDcorr meets these goals rather well. 100,000 simulation scenarios were generated, each containing base trains and noised trains with 100 spikes on the average. The VRDcorr algorithm calculated c_{opt} for the based/noised pair using 41 values for τ: 1.1^i for $i = -20, ...20$, so τ varied between 0.1486 and 6.7275. In each scenario, the set of 41 c_{opt} values had mean and standard deviation computed.

The predicted value for c_{opt} is 0. The values for τ were chosen to cover a reasonable range about the expected interspike time of 1. Note: For τ much bigger than 6, the van Rossum transform approximates a sum of delta functions and the correlation degenerates due to noise in the individual spikes. Nothing interesting happens when τ is less than 0.14.

Figures 7 and 8 below histogram these sets of means and standard deviations.

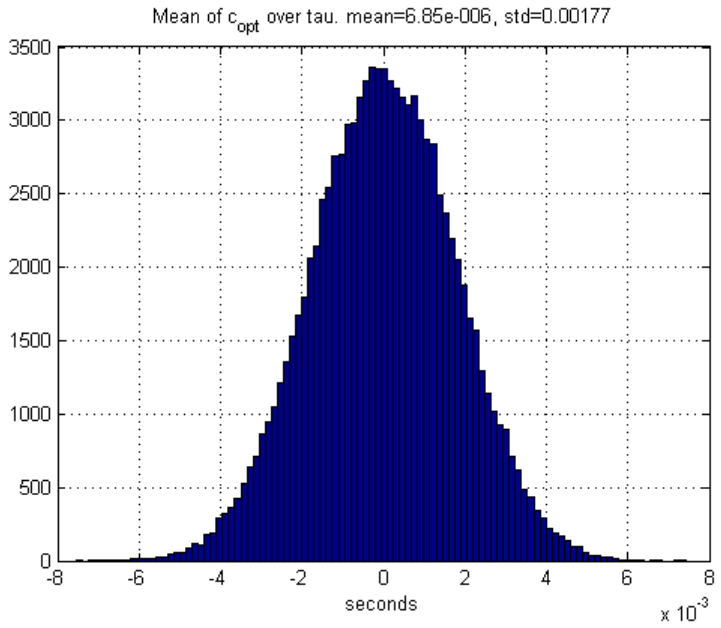

Figure 7. Histogram of calculated c_{opt} averaged over τ.

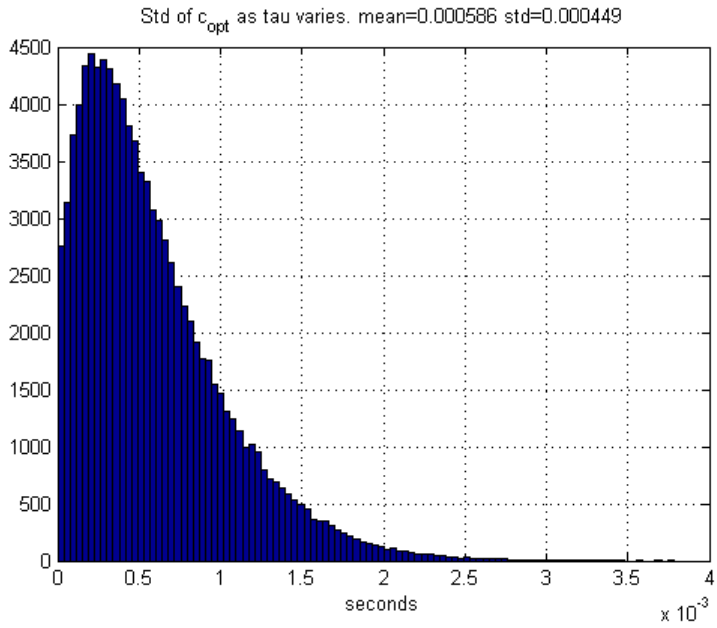

Figure 8. Histogram of variation in c_{opt} in τ

In short, c_{opt} is calculated correctly within about a millisecond. For any one scenario, the variation of c_{opt} with τ tended to be less than half a millisecond. The errors are small compared to the average interspike time of one second, and probably are produced by step 3 in the noising process, which adds a random value of standard deviation 8.7 milliseconds to each spike time. The errors in c_{opt} are significantly less than the noise in individual spike times.

So, for this simulation varying τ within a factor of 6 of the average interspike time, the c_{opt} calculation has relative error about 10^{-3}. The calculation bias (overall mean displayed in Figure 7) is about a factor of 10^{-5}. The error is not sensitive to τ.