Fast Methods Comp

Joshua T. Vogelstein, Baktash Babadi, Liam Paninski

February 20, 2008

Contents

1 Introduction **1**

2 Methods **1**

2.1 Model 1

2.2 Inferring the spike times 2

2.3 Inferring the parameters 6

3 Results **7**

4 Figures **8**

Abstract

1 Introduction

The goal of this work is to infer the underlying spike train, having only the observed fluorescence signal available to us. To do so, we propose a parametric model of the experimental system. Then, we develop a number of algorithms, each iteratively estimating the parameters and then using those parameters to infer the hidden spike trains.

2 Methods

2.1 Model

First, for simplicity, we assume that we have extracted a time series of fluorescence observations, F_t, which is a function of the intracellular calcium concentration of the imaged soma at that time, $[\text{Ca}^{2+}]_t$. In particular, we assume that F_t simply a noisy version of $[\text{Ca}^{2+}]_t$, i.e.,

$$F_t = [\text{Ca}^{2+}]_t + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, \sigma^2) \tag{1}$$

where ε_t is an additive Gaussian noise term with zero mean and variance σ^2. Note that this time series is naturally discrete, with time step size Δ corresponding to the frame rate of the image acquisition system, with t indexing the time step, and T indicating the final time step. The $[\text{Ca}^{2+}]$ signal is assumed to jump immediate after each spike by A μM and then decay back to rest, $[\text{Ca}^{2+}]_0$, with time constant τ sec. Therefore, we have:

$$[\text{Ca}^{2+}]_t = a[\text{Ca}^{2+}]_{t-1} + An_t + b, \tag{2}$$

where we have parameters $a = 1 - \frac{\Delta}{\tau}$ and $b = \frac{\Delta}{\tau}[\text{Ca}^{2+}]_0$, where $[\text{Ca}^{2+}]_0$ is the baseline $[\text{Ca}^{2+}]$, and n_t is the underlying spike train. Figure 1 shows an example fluorescence time-series (A), the true calcium signal (B), and the true spike train (C).

2.2 Inferring the spike times

Least-Squares Solution We would like to find the most likely spike train that led to the observed fluorescence signal. This problem may be formally cast as a least-squares problem:

$$\vec{n}_{lsq} = \underset{\vec{n}}{\operatorname{argmin}} \sum_{t=0}^{T} (F_t - [\text{Ca}^{2+}]_t)^2, \tag{3}$$

where we use the notation $\vec{n} = n_0, \ldots, n_T$, and $[\text{Ca}^{2+}]$ is implicitly a function of n. As (3) is a quadratic optimization problem, it may be solved with any gradient ascent technique. In particular, for any quadratic problem:

$$\vec{x}_{lsq} = \underset{\vec{x}}{\operatorname{argmin}} \left\| \boldsymbol{Y}\vec{x} - \vec{z} \right\|_2^2, \tag{4}$$

the analytical solution is simply:

$$\vec{x}_{lsq} = \vec{x}_0 - \boldsymbol{H}^{-1}\vec{g} = \vec{x}_0 - (\boldsymbol{Y}'\boldsymbol{Y})^{-1}\boldsymbol{Y}'\vec{z} \tag{5}$$

where $\vec{x}_0$ is some initial guess for the value of $\vec{x}$, $\boldsymbol{H}$ is the Hessian matrix (i.e., second derivative of $\|\boldsymbol{Y}\vec{x} + \vec{z}\|_2^2$ with respect to $\vec{x}$, and $\vec{g}$ is the gradient vector (i.e., first derivative of $\|\boldsymbol{Y}\vec{x} + \vec{z}\|_2^2$ with respect to $\vec{x}$. Thus to solve any quadratic optimization problem, one must simply cast it in the form of (4) and then solve for $\boldsymbol{H}$ and $\vec{h}$. The key to solving (3) problem efficiently is to write $\vec{n}$ as a function of $\vec{C} = [[\text{Ca}^{2+}]_0, \ldots, [\text{Ca}^{2+}]_T]$. In particular, after subtracting b, we may write:

$$\vec{n} = \boldsymbol{M}\vec{C} \tag{6}$$

where $\boldsymbol{M}$ is a $T \times T$ bidiagonal deconvolution matrix:

$$\boldsymbol{M} = \begin{bmatrix} 1/A & 0 & 0 & 0 & \dots & 0 \\ -a/A & 1/A & 0 & 0 & \dots & 0 \\ 0 & -a/A & 1/A & 0 & \dots & 0 \\ \vdots & & & & & \\ 0 & \dots & 0 & 0 & -a/A & 1/A \end{bmatrix}. \tag{7}$$

which follows from (2). Thus, letting $c = 1/(2\sigma^2)$ we may rewrite (3), and solve for $\vec{g}$ and $\boldsymbol{H}$:

$$\vec{C}_{lsq} = \underset{\vec{C}}{\operatorname{argmin}}\, c \left\| \vec{F} - \vec{C} \right\|_2^2 \tag{8}$$

$$\vec{g} = -2c(\vec{F} - \vec{C}) \tag{9}$$

$$\boldsymbol{H} = 2\boldsymbol{I} \tag{10}$$

where $\boldsymbol{I}$ is an identity matrix of appropriate size. Thus,

$$\vec{C}_{lsq} = \vec{C} - (2\boldsymbol{I})^{-1}\big(- 2c(\vec{F} - \vec{C})\big) = \vec{F}, \tag{11}$$

and therefore, $\vec{n}_{lsq} = \boldsymbol{M}\vec{F}$. This solution, plotted in Figure 1(D), is problematic for several reasons. First, in periods of quiescence (i.e., no spiking), the algorithm incorrectly infers the presence of many "spikelets", which do not exist in the true data. Second, the algorithm sometimes infers *negative* spikes, which also cannot exist. Third, the number of spikes in a frame is not necessarily an integer. Each of these issues may be treated using standard tools from the optimization literature.

Regularizing To deal with the spikelets, we use a technique called "regularizing". Essentially, we impose a prior on the probability of their being a spike at any time, and then we penalize the solution if it infers too many spikes. Formally, we write:

$$\vec{n}_{reg} = \underset{\vec{n}}{\operatorname{argmin}} \sum_{t=0}^{T} c\big((F_t - [\mathrm{Ca}^{2+}]_t)^2 + f_{reg}(n_t)\big), \tag{12}$$

where $f_{reg}(n_t)$ is some regularizing function of n_t, typically chosen such that the likelihood remains concave. For example, $f_{reg}(n_t) = \lambda_t n_t^2$, where $\lambda_t = 1/(\text{expected } n_t)$, then we have a standard *smoothing prior*.[1] To solve (12) efficiently with a smoothing prior, we rewrite (12) as a function of $\vec{C}$:

$$\vec{C}_{reg} = \underset{\vec{C}}{\operatorname{argmin}}\, c \left\| \vec{F} - \vec{C} \right\|_2^2 + \left\| \sqrt{\boldsymbol{\lambda}}\boldsymbol{M}\vec{C} \right\|_2^2 \tag{13}$$

[1] This may be thought of as a generalization of ridge regression and Wiener deconvolution.

where $\sqrt{\boldsymbol{\lambda}}$ is a matrix with components $\sqrt{\lambda_t}$ along the diagonal and zeros elsewhere. Because this is also a quadratic problem, the solution is given by finding Hessian and gradient with respect to $\vec{C}$:

$$\vec{g} = -2c(\vec{F} - \vec{C}) + 2(\sqrt{\boldsymbol{\lambda}}\boldsymbol{M}\vec{C})'(\sqrt{\boldsymbol{\lambda}}\boldsymbol{M}) \tag{14}$$

$$\boldsymbol{H} = \left(2c\boldsymbol{I} + 2(\sqrt{\boldsymbol{\lambda}}\boldsymbol{M})'(\sqrt{\boldsymbol{\lambda}}\boldsymbol{M})\right)^{-1} \tag{15}$$

The key to solving this problem efficently is that the Hessian is a tridiagonal matrix (which follows from the fact that $\sqrt{\boldsymbol{\lambda}}$ is diagonal and $\boldsymbol{M}$ is bidiagonal). Therefore, $\boldsymbol{H}^{-1}\vec{g}$ may be solved in $O(T)$ time (for instance, using Matlab's \), instead of the typical $O(T^3)$ time normally required. Figure 1(F) shows the results of (13). Problematically, the solutions from such an approach possibly yield negative spikes, which are not possible in the true spike train.

Barrier method It is of interest, therefore, to find the best underlying *non-negative* spike train:

$$\vec{n}_{nng} = \underset{\vec{n}:n_t \geq 0}{\operatorname{argmin}} \sum_{t=0}^{T} \left(c(F_t - [\mathrm{Ca}^{2+}]_t)^2 + \lambda_t n_t\right). \tag{16}$$

Note that we have chosen to let $f_{reg}(n_t) = \lambda n_t$ here, which ensures that not too many spikes are inferred. To efficiently solve this problem, we adopt the "barrier method". This approach iteratively solves a related problem that penalizes the algorithm for inferring negative values, but with each iteration, reduces the penalty, in such a way that is guaranteed to converge to (16)**??**. Formally, it may be written as

$$\vec{n}_{\eta} = \underset{\vec{n}}{\operatorname{argmin}} \sum_{t=0}^{T} \left(c(F_t - [\mathrm{Ca}^{2+}]_t)^2 + \lambda_t n_t + \eta f(n_t)\right), \tag{17}$$

where $f(n_t)$ is a barrier function, which is any function that approaches infinity from the right as n_t approaches zero, e.g., $-\log(n_t)$. So, as $\eta \to 0$, $\vec{n}_{\eta} \to \vec{n}_{nng}$. While this function is no longer quadratic, it is still concave. Therefore, the solution may be found be iterating any gradient descent technique. The solution to Newton's method (which we used for both (3) and (12)), in terms of $\vec{C}$, is:

$$\vec{C}_i = \vec{C}_{i-1} + s\vec{d} \tag{18}$$

where s is the step size and $\vec{d} = \boldsymbol{H}^{-1}\vec{g}$ is the step direction. s must be chosen such that all of our constraints, namely $n_t > 0$ for all t, are satisfied:

$$0 < n_t \tag{19}$$

$$< \boldsymbol{M}(\vec{C} + s\vec{d}) \tag{20}$$

$$< \boldsymbol{M}\vec{C} + s\boldsymbol{M}\vec{d} \tag{21}$$

$$\Rightarrow s > -\boldsymbol{M}\vec{C}(\boldsymbol{M}\vec{d})^{-1}. \tag{22}$$

So s must be larger than every component of the vector on the right-hand-side of the last inequality. Furthermore, it is possible to "over-step", or rather, take a step so large as to *increase* the likelihood that we are trying to minimize. As such, prior to stepping, we check that the likelihood would indeed decrease, if not, we reduce s until it does.

To compute $\vec{d}$, we must solve for the Hessian and gradient of our likelihood function with respect to $\vec{C}$:

$$\mathcal{L} = c\left\|\vec{F} - \vec{C}\right\|_2^2 + \boldsymbol{\lambda}\boldsymbol{M}\vec{C} - \eta\log(\boldsymbol{M}\vec{C}) \tag{23}$$

$$\vec{g} = -2c(\vec{F} - \vec{C}) + \vec{\lambda}\boldsymbol{M} - \eta\boldsymbol{M}'(\boldsymbol{M}\vec{C})^{-1} \tag{24}$$

$$\boldsymbol{H} = 2c\boldsymbol{I} + 2\eta\boldsymbol{M}'(\boldsymbol{M}\vec{C})^{-2}\boldsymbol{M}. \tag{25}$$

Plugging $\boldsymbol{H}$ and $\vec{g}$ into (18) and then choosing the step size s to satisfy (22) and decrease the likelihood yields $\vec{C}_i$. This is iterated until it converges, at which time η is reduced and the whole process repeats. Importantly, as for the regularized solution, the Hessian here is tridiagonal, facilitating $O(T)$ time to solve each Newton step, as opposed to $O(T^3)$, which would be unacceptably long for large data sets. Pseudocode for an efficient implementation of this approach is provided in Table 1. Figure 1(F) shows how this constraint further improves the inference accuracy.

Table 1: Pseudocode for barrier method

For each η, let $\vec{C}_i$ be the current estimate of $\vec{C}$ (which may be initialized at $\vec{1}$ for instance). While $\vec{C}_i < \vec{C}_{i-1} + \xi$ (for some small ξ), iterate the following steps. When complete, reduce η and repeat.

- Compute $\vec{g}$ using (24)
- Compute $\boldsymbol{H}$ using (25)
- Compute $\vec{d} = \boldsymbol{H}^{-1}\vec{g}$ efficiently
- Choose s such that (22) is satisfied and the likelihood decreases upon plugging s and $\vec{d}$ into (18)

Projection Pursuit Regression (PPR) Although the barrier method provides us with a meaningful spike train inference, we can further constrain the inferred spike train to be integers, i.e.,

$$\vec{n}_{ppr} = \underset{\vec{n}:n_t\in 0,1,2,\ldots}{\operatorname{argmin}} \sum_{t=0}^{T} \left(c(F_t - [\mathrm{Ca}^{2+}]_t)^2 + \lambda_t n_t\right). \tag{26}$$

Note that this formulation obviates the need to impose a non-negative constraint, but the prior $\lambda_t n_t$ still performs a useful function. In particular, in noisy scenarios, it helps the algorithm not infer too many spikes. Pseudocode for an algorithm called *projection pursuit regression* (PPR)**??** which efficiently solving (26) is provided in Table 2 . Figure 1(G) shows the inferred spike train using this algorithm. Although this approach imposes a desirable constraint (i.e., that spike trains include only integers), it also comes at a cost relative to the barrier method, as will be described in Section 3.

Table 2: Pseudocode for PPR

This algorithm iterates several steps, as schematized by Figure **??**. The algorithm is initialized with the fluorescence signal, which is the residual before iterating the following steps, $\vec{r}_0$:

- Compute the cross-correlation between the residual and the calcium kernel, $xc(t) = r_i(t) \otimes Ae^{-t/\tau}$.
- Let t_{i+1} be the maximum of that cross-correlation, $t_{i+1} = \max_t xc(t)$.
- Let $\vec{o}_{i+1}$ be a zero vector with unity at t_{i+1}. Convolve $\vec{o}_{i+1}$ with the calcium kernel to generate the $[\text{Ca}^{2+}]$ that would result from a spike having occurred at that time step, $[\text{Ca}^{2+}]_{i+1}(t) = o_{i+1}(t) * Ae^{-t/\tau}$.
- Subtract $[\vec{\text{Ca}}^{2+}]_{i+1}$ from the residual of the previous step to get the new residual, $\vec{r}_{i+1} = \vec{r}_i - [\vec{\text{Ca}}^{2+}]_{i+1}$
- Compute the sum of residual error, where $\vec{n}_{i+1}$ is a vector of zeros with ones inferred spike time, $\epsilon_{i+1} = \sum_t r_{i+1}(t)^2 + \lambda_t n_{i+1}(t)$
- If $\epsilon_{i+1} < \epsilon_i$, repeat. Otherwise, let $\vec{n}_{ppr} = \vec{n}_i$.

2.3 Inferring the parameters

All of the above approaches assuming a particular effect of spikes on $[\text{Ca}^{2+}]$. More precisely, they all assume that after each spike, $[\text{Ca}^{2+}]$ jumps by A μM and then decays back to $[\text{Ca}^{2+}]_0$ μM with time constant τ sec. Let $[\widehat{\text{Ca}}^{2+}]_t$ and $\widehat{n}_t$ be the estimated $[\text{Ca}^{2+}]_t$ and n_t, respectively. We may then solve for:

$$\{\widehat{a}, \widehat{A}, \widehat{b}\} = \underset{a,A,b>0}{\operatorname{argmin}} \left(F_t - a[\widehat{\text{Ca}}^{2+}]_{t-1} - A\widehat{n}_t - b\right)^2. \tag{27}$$

where, as before, we have $a = 1 - \frac{\Delta}{\tau}$ and $b = \frac{\Delta}{\tau}[\text{Ca}^{2+}]_0$. This may be solved in matrix form by making the following substitutions:

$$\vec{C}_{est} = \begin{bmatrix} [\widehat{\text{Ca}}^{2+}]_2 \\ \vdots \\ [\widehat{\text{Ca}}^{2+}]_T \end{bmatrix}, \vec{n}_{est} = \begin{bmatrix} \widehat{n}_1 \\ \vdots \\ \widehat{n}_{T-1} \end{bmatrix}, \vec{F} = \begin{bmatrix} F_1 \\ \vdots \\ F_{T-1} \end{bmatrix}, \boldsymbol{G} = \begin{bmatrix} \vec{C}_{est} \\ \vec{n}_{est} \\ \vec{1} \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} -a \\ -A \\ -b \end{bmatrix}, \tag{28}$$

where $\vec{1}$ is a column vector of ones of the appropriate length. Thus, we may write another quadratic optimization problem:

$$\vec{x}_{est} = \underset{\vec{x}>\vec{0}}{\operatorname{argmin}} \left\| \boldsymbol{G}\vec{x} + \vec{F} \right\|_2^2 \tag{29}$$

$$\vec{g} = \boldsymbol{G}'\vec{F} \tag{30}$$

$$\vec{H} = \boldsymbol{G}'\boldsymbol{G} \tag{31}$$

which may be efficiently solved using, for instance, Matlab's `quadprog`. The variance of the error, σ^2, is simply the mean-squared-error of the residual**??**:

$$\sigma^2 = \frac{1}{T}\sum_{t=0}^{T}(F_t - [\widehat{\mathrm{Ca}}^{2+}]_t)^2 \tag{32}$$

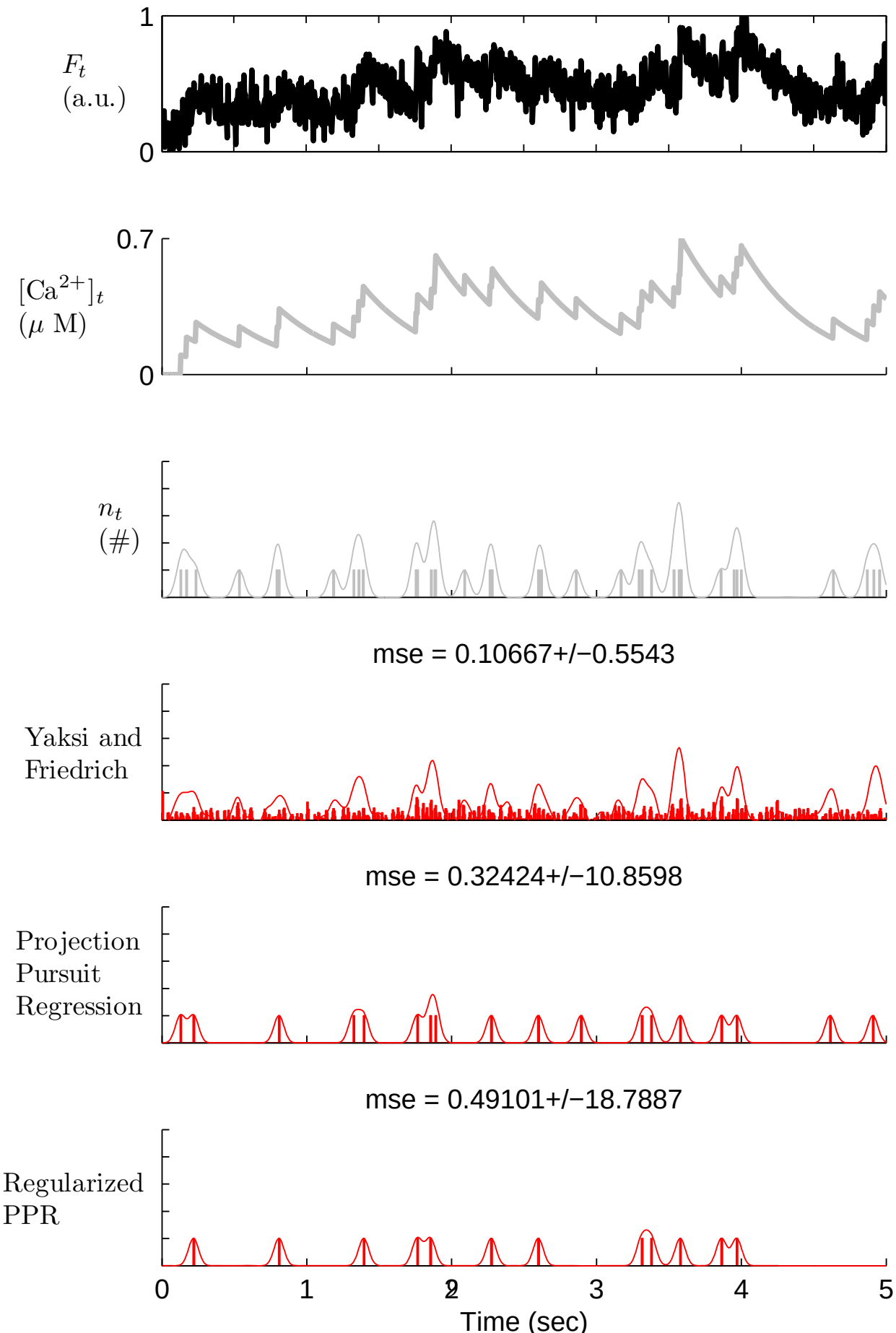
where $[\widehat{\mathrm{Ca}}^{2+}]_t$ is the $[\mathrm{Ca}^{2+}]$ having substituted $\widehat{a}$, $\widehat{A}$, and $\widehat{b}$ for a, A, and b, respectively.

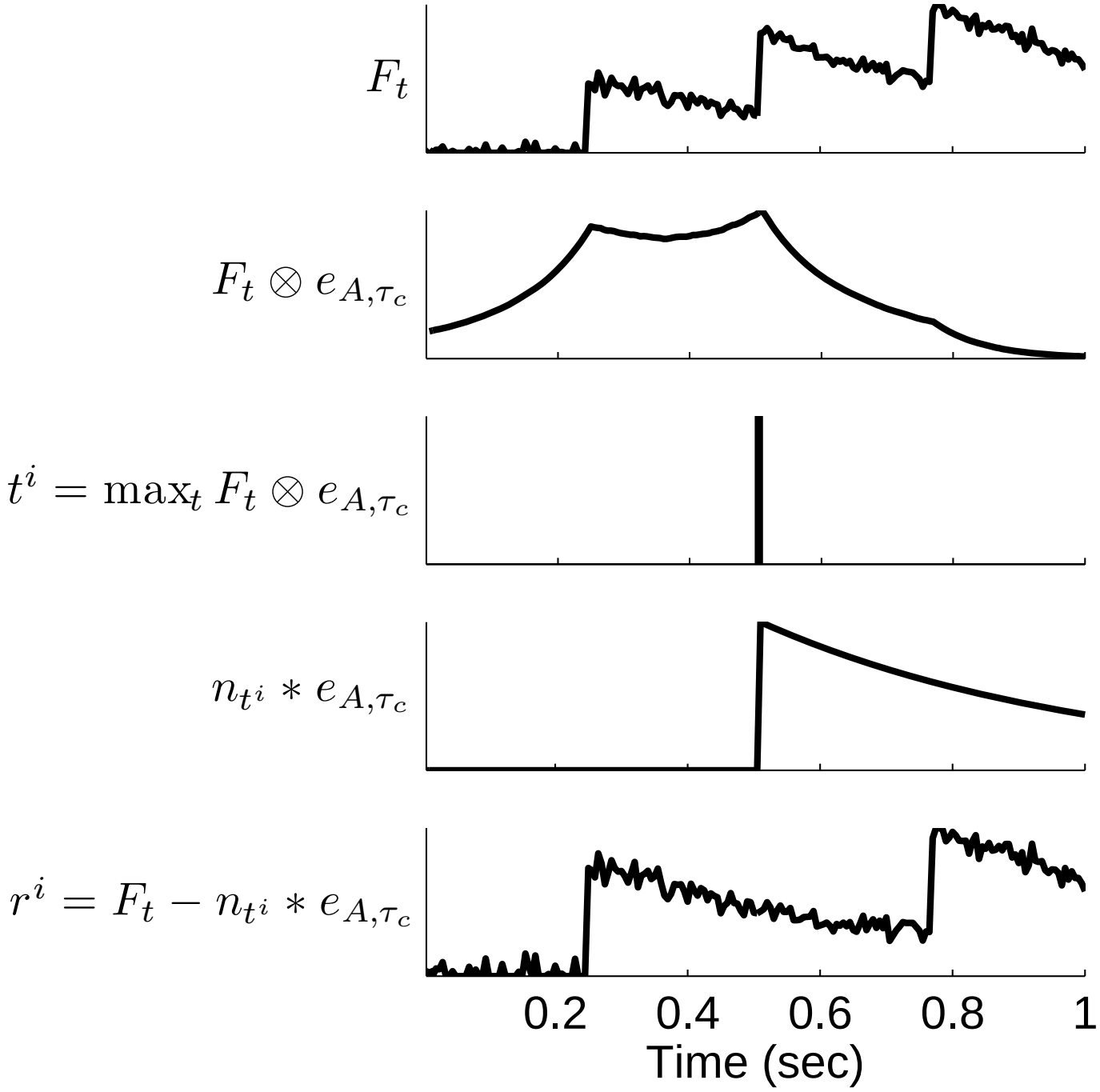
3 Results

For certain scenarios, the desired temporal resolution of $[\mathrm{Ca}^{2+}]_t$ may be finer than Δ

Figure 1(E) shows the output of the algorithm proposed previously by Yaksi and Friedrich (2006)**??**, which may be thought of as a custom approximation to either of these regularization approaches, and

4 Figures


F_t
(a.u.)
1
0
$[Ca^{2+}]_t$
(μ M)
0.7
0
n_t
(#)
mse = 0.10667+/−0.5543
Yaksi and
Friedrich
mse = 0.32424+/−10.8598
Projection
Pursuit
Regression
mse = 0.49101+/−18.7887
Regularized
PPR
0
1
2
3
4
5
Time (sec)


F_t
$F_t \otimes e_{A,\tau_c}$
$t^i = \max_t F_t \otimes e_{A,\tau_c}$
$n_{t^i} * e_{A,\tau_c}$
$r^i = F_t - n_{t^i} * e_{A,\tau_c}$
0.2
0.4
0.6
0.8
1
Time (sec)