System Identification of Adapting Neurons

Eric Hunsberger

CTN Technical Report
September 13, 2016

Abstract

This report investigates how neurons with complex dynamics, specifically adaptation, can be incorporated into the Neural Engineering Framework. The focus of the report is fitting a linear-nonlinear system model to an adapting neuron model using system identification techniques. By characterizing the neuron dynamics in this way, we hope to gain a better understanding of what sort of temporal basis the neurons in a population provide, which will determine what kinds of dynamics can be decoded from the neural population.

The report presents four system identification techniques: a correlation-based method, a least-squares method, an iterative least-squares technique based of Paulin's algorithm, and a general iterative least squares method based of gradient descent optimization. These four methods are all used to fit linear-nonlinear models to the adapting neuron model. We find that the Paulin least-squares method performs the best in this situation, and linear-nonlinear models fit in this manner are able to capture the relevant adaptation dynamics of the neuron model. Other questions related to the system identification, such as the type of input to use and the amount of regularization required for the least-squares methods, are also answered empirically. The report concludes by performing system identification on 20 neurons with a range of adaptation parameters, and examining what type of temporal basis these neurons provide.

1 Introduction

The Neural Engineering Framework (NEF) describes how a population of neurons can compute a wide range of nonlinear functions, using nonlinear encoding and linear decoding [1]. It models a population of neurons as each having a preferred direction vector. The neurons obey cosine tuning, with activation given by a nonlinear function of the dot product between the input stimulus and the neuron's preferred direction vector. Linear decoders can then be found using least squares, which when multiplied by the neuron activations, reproduce either the original stimulus or an arbitrary function of this stimulus. This creates a powerful framework for transforming computations in abstract vector spaces to computations in neural populations.

One shortcoming of the NEF is that it currently only applies to neurons with a limited range of dynamics. The NEF uses a rate approximation to neurons in a population when solving for the linear decoders to compute a function, so the firing rate of neurons in the population must remain constant given a constant input signal. This is equivalent to the neuron having a fixed response function that does not change over time.

Many neurons observed in vitro exhibit dynamics that cannot be captured by the static response function model. For example, many cortical neurons show a property called adaptation: Given a constant input, the neuron fires rapidly at stimulus onset, but then quickly reduces its firing rate. Other cortical neurons exhibit bursting, emitting short bursts of spikes rather than firing at a constant rate. It is unclear how complex neuron dynamics like these should be dealt with by the NEF.

One solution is to simply continue with the current NEF methods, computing the response functions of neurons over long time scales, and using these to find the linear decoders. For adapting neurons, the computed response function would be dominated by the neuron dynamics after adaptation; this decoding would work well for slowly varying signals, but would be inaccurate for higher frequency ones. Therefore this decoding method only works for neurons whose response functions reach a steady state for the given input signals. Another drawback to this decoding method is that it cannot take advantage of these neural dynamics to compute more complex temporal functions.

These problems motivate a second solution, which is to model the neuron dynamics in a way that they can be included when solving for decoders. The linear-nonlinear (LN) model is a natural candidate for this, since it decomposes the neuron into a linear dynamic system and a static nonlinearity. This model is very comparable to the current NEF model, but instead of a preferred direction vector in the input space, the neuron's linear kernel is a preferred direction vector in time. Fitting a population of adapting neurons to linear-nonlinear models will allow us to understand what sort of temporal basis the population provides, and give an idea of what types of temporal signals can be decoded from the population.

This report briefly presents an adapting neuron model and the linear-nonlinear (LN) system model, then describes in-depth a wide variety of system identification techniques for fitting LN models to a system, as well as many considerations related to performing system identification on our specific to our neural system. These system identification techniques are evaluated on the adapting neuron model; the results are presented and the best technique determined. This optimal technique is then applied to a population of neurons to characterize the range of linear kernels present in the system. The report concludes with suggestions as to how these results can be applied to improving decoding in the NEF.

2 Methods

This section first presents the adapting neuron model which serves as the system of study for the remainder of the report. It then describes the linear-nonlinear (LN) system model which will be used to characterize the dynamics of the adapting neuron model. Finally, it explores in detail the system identification techniques used to fit the LN model to the neuron model, and raises some challenges with system identification as it relates to our system of interest.

2.1 Adaptive Leaky Integrate-and-fire Neuron

Adaptation is a decrease in the firing rate of a neuron given a constant input, and is commonly observed in cortical neurons. It is a basic form of neural dynamics that is not currently captured by the NEF, and due to its simplicity and ubiquity it is the type of neuron dynamics examined in this report.

The adaptive leaky integrate-and-fire (ALIF) neuron model is an extension of the leaky integrate-and-fire (LIF) neuron model that accounts for neural adaptation [2]. The ALIF model has two state variables, the voltage $v(t)$ and the adaptation parameter $n(t)$. In practice, the model has a third variable $w(t)$, which is responsible for measuring the absolute refractory period. The model is given by

$$\tau_{RC}\dot{v}(t) = R\left[\alpha u(t) + \beta - g_n n(t)\right] - v(t) + \eta(t) \tag{1}$$
$$\tau_n \dot{n}(t) = -n(t) \tag{2}$$

where $R = 1$ is the membrane resistance, $g_n = 1$ is the adaptation conductance, τ_{RC} is the membrane time constant, τ_n is the adaptation time constant, and α and β are a scaling and offset for the input signal, respectively. When the voltage crosses the threshold $V_{th} = 1$, the neuron fires a spike. The voltage is then reset to zero, and the adaptation parameter is increased by n_{inc}. The voltage is held at zero for a short length of time after a spike, called the absolute refractory period t_{ref}.

The LIF neuron model (without adaptation) has an analytic response function that describes the neuron's rate as a function of the constant input level s:

$$r(s) = \left[t_{ref} - \tau_{RC}\log\left(1 - \frac{V_{th}}{R(\alpha s + \beta)}\right)\right]^{-1}. \tag{3}$$

The adapting version of the model seems to follow this response function initially, for the first few spikes before adaptation, but then settles to a much flatter response function, similar to a rectified linear function. However, the LIF tuning curve seems to be a good fit for the nonlinearity when fitting linear-nonlinear models to the ALIF neuron model, so Equation 3 is used for fitting nonlinearities in the following methods, with parameters α, β, and τ_{RC} variable and all other parameters fixed.

2.2 Linear-Nonlinear Models

The linear-nonlinear (LN) model, or Wiener model, is a simplified description of an arbitrary dynamical system [3]. The model consists of a dynamic linear component and a static nonlinear component. The system input $u(t)$ is first passed through the dynamic linear system, fully described by its impulse response $h(\tau)$ (also known as the linear kernel). The linear response $x(t)$ is given by the convolution of this kernel with the input signal

$$x(t) = \int_0^t u(\tau)h(t-\tau)d\tau. \tag{4}$$

This linear response is then passed through the static nonlinearity $m(\cdot)$ to produce the system response

$$y(t) = m\left(x(t)\right). \tag{5}$$

The LN model has previously been used in computational neuroscience to characterize the rate response of a neuron to a stimulus [4]. It bears a strong resemblance to the neuron encoding model currently used by the Neural Engineering Framework (NEF). In the NEF, the rate response of a neuron is described by the neuron's encoder $\underline{\phi}$ (often called a preferred direction vector in the literature) and the neuron's static nonlinear response. Specifically, the response $r(t)$ is given by the dot-product between the multidimensional input $\underline{u}(t)$ and the encoder, passed through the nonlinearity $g(\cdot)$:

$$r(t) = g\left(\left\langle \underline{\phi}, \underline{u}(t) \right\rangle\right) \tag{6}$$

where $\left\langle \underline{x}, \underline{y} \right\rangle$ is the dot-product between two vectors $\underline{x}$ and $\underline{y}$. In discrete time, the convolution integral of Equation 4 becomes a dot product, and therefore the linear kernel $h(\tau)$ is analogous to the preferred direction vector $\underline{\phi}$ of a neuron, except $h(\tau)$ is a preferred direction vector in time. Fitting a population of neurons with LN models will allow us to characterize what sort of temporal basis the population provides, which determines what sort of signals can be decoded from the population.

2.3 Linear-nonlinear Model Estimation

Here I describe the system identification methods that will be used to fit linear-nonlinear models to the ALIF neurons. All of these methods are described in detail in [3].

2.3.1 Correlation-based Methods

Correlation-based methods were some of the first methods developed for system identification, and are still in use [3]. They can be used to fit both linear and nonlinear models to a system output. I will first summarize how these methods apply to linear systems, and then I will present the extension to our nonlinear system of interest.

Given a linear system with impulse response $h(\tau)$, the system response $y(t)$ to input $u(t)$ is given by

$$y(t) = (u * h)(t) \tag{7}$$

where $(f * g)(t)$ is the convolution of two functions $f(\cdot)$ and $g(\cdot)$ evaluated at time t. Taking the correlation of both sides of this system with the input $u(t)$ results in the equation

$$\phi_{uy}(\tau) = (\phi_{uu} * h)(\tau) \tag{8}$$

where $\phi_{uy}(\tau)$ is the cross-correlation of $u(t)$ and $y(t)$, and $\phi_{uu}(\tau)$ is the autocorrelation of $u(t)$.

If we stimulate the system with a white input (*i.e.,* the signal values of $u(t)$ from one time point to the next are independent), then the autocorrelation $\phi_{uu}(\tau) = \sigma_u^2 \delta(\tau)$, where σ_u is the standard deviation of $u(t)$ and $\delta(\cdot)$ is the Dirac delta function. This greatly simplifies Equation 8 due to the filtering property of the delta function in convolution, making the linear kernel equal to a scaled copy of the cross-correlation between the system input and output

$$h(\tau) = \phi_{uy}(\tau)/\sigma_u^2. \tag{9}$$

Thus the kernel of a linear system is easily determined from the cross-correlation of the system input and output.

Bussgang's theorem allows us to apply these results from linear systems to nonlinear systems. The theorem states that

> *For two Gaussian signals, the cross-correlation function taken after one signal has undergone a nonlinear amplitude distortion is identical, except for a factor of proportionality, to the cross-correlation function taken before the distortion.* [3]

This theorem allows us to apply Equation 9 to nonlinear systems to estimate the linear kernel $h(t)$. Once we have an estimate of this kernel, we can form the linear response of the system

$$x(t) = (u * h)(t). \tag{10}$$

This linear response can then be compared with the actual system response $z(t)$, and a static nonlinearity $m(\cdot)$ fit to the set of points $\{(x(t_i), z(t_i))\}$. The output of the linear nonlinear system is then given by

$$y(t) = m\left[(u * h)(t)\right]. \tag{11}$$

For our linear system, preliminary results showed that using a white-noise signal to fit the nonlinearity $m(\cdot)$ did not produce good results (see

Section 3.1). For this reason, when applying correlation-based methods, I used white noise and Equation 9 to identify the linear kernel of the system, and then used a different (non-white) input signal to generate the linear response of the system and fit the nonlinearity $m(\cdot)$, resulting in a much better fit.

2.3.2 Least-Squares Method

The correlation-based methods described above are able to solve for the linear kernel at low computational cost, but to do this they make some assumptions about the input signal. Specifically, they assume not only that the input is white, but also that the finite-length input sequence is a perfect instantiation of white noise (*i.e.,* equal power at all frequencies). For any finite-length signal, this will never be the case. Though some correlation-based methods have been extended to colored noise, all of them encounter this second problem.

To address this, we can explicitly formulate the least-squares problem as a linear system, and solve this linear system for the linear kernel. This formulation is based on the fact that at any given time point t the convolution between two discrete signals is simply a dot-product. This allows us to rewrite the convolution $y(t) = (u * h)(t)$ as

$$\begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(T-1) \\ y(T) \end{bmatrix} = \begin{bmatrix} u(1) & 0 & \cdots & 0 & 0 \\ u(2) & u(1) & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ u(T-1) & u(T-2) & \cdots & u(T-N+1) & u(T-N) \\ u(T) & u(T-1) & \cdots & u(T-N+2) & u(T-N+1) \end{bmatrix} \begin{bmatrix} h(1) \\ h(2) \\ \vdots \\ h(N-1) \\ h(N) \end{bmatrix} \tag{12}$$

where T is the number of time points in the input and output signals, and N is the number of elements in the kernel. This linear system $\underline{y} = \mathrm{U}\underline{h}$ can now be solved for $\underline{h}$ using least-squares:

$$\underline{h} = (\mathrm{U}^T\mathrm{U})^{-1}\mathrm{U}^T\underline{y}. \tag{13}$$

This equation can be solved directly, using for example the Cholesky decomposition to invert the matrix $(\mathrm{U}^T\mathrm{U})$. It can also be solved using the QR factorization $\mathrm{U} = \mathrm{QR}$, in which case

$$\underline{h} = \mathrm{R}^{-1}\mathrm{Q}^T\underline{y}. \tag{14}$$

This second method is known as the ordinary Orthogonal Algorithm (OOA) [3].

To improve efficiency, the Fast Orthogonal Algorithm (FOA) avoids forming the matrix U by finding the matrix $\Phi_{uu} = \mathrm{U}^T\mathrm{U}$ and the vector $\Phi_{uy} = \mathrm{U}^T\underline{y}$ directly [3]. This is done using a combination of the fast Fourier transform (FFT) and the structure inherent to the U and by extension the Φ_{uu} matrices.

The first column of Φ_{uu} is simply the autocorrelation of $u(t)$, found using the Fourier transform:

$$\Phi_{uu}(:,1) = \mathfrak{F}^{-1}\left\{\mathfrak{F}\left\{u(t)\right\}^{*}\mathfrak{F}\left\{u(t)\right\}\right\} \tag{15}$$

where $\mathfrak{F}^{-1}\{\cdot\}$ is the Fourier transform, $\mathfrak{F}^{-1}\{\cdot\}$ is the inverse Fourier transform, and $\cdot^{*}$ is the complex conjugate. Subsequent columns of Φ_{uu} are autocorrelations of truncated versions of $u(t)$ (*i.e.,* where some of the end terms $u(T), u(T-1)$, etc., have been removed from the signal). This results in the following equation, which can be used to calculate the subsequent columns of Φ_{uu}:

$$\Phi_{uu}(i,j) = \Phi_{uu}(i,j-1) - u(T-i+1) * u(T-j+1). \tag{16}$$

Furthermore, the above calculation only needs to be performed for the half of the matrix $i > j$, since Φ_{uu} is symmetric. The vector Φ_{uy} is the cross-correlation of $u(t)$ and $y(t)$, and is also found using the FFT method. The resulting system $\underline{h} = \Phi_{uu}^{-1}\Phi_{uy}$ can be solved efficiently using the Cholesky decomposition.

In addition to improved accuracy offered over the correlation method, the least-squares method also opens up the possibility of regularizing the solution. Regularization is an additional constraint on the solution that penalizes undesired properties of the parameters. For example, one can penalize the magnitude of the linear kernel components, the slope between one component and the next, or the change in slope across three components. Regularization is implemented by adding a regularization matrix R to the autocorrelation matrix:

$$(\Phi_{uu} + \lambda\mathrm{R})\underline{h} = \Phi_{uy} \tag{17}$$

where λ controls the amount of regularization. R can be determined using a constraint matrix L, where each row of L is a constraint, and each column corresponds to an element of the linear kernel. To penalize the magnitude of kernel element j, one adds a row A to L with $A_j = 1$ and all other elements zero. To penalize the slope between two adjacent elements j and

$j+1$, one adds a row with $A_j = -1$ and $A_{j+1} = 1$ (again with all other elements zero). To penalize the curvature across three elements, one sets $A_{j-1:j+1} = [-1, 2, -1]$. In all cases, the regularization matrix is calculated as $\mathrm{R} = \mathrm{L}^T\mathrm{L}$,. For a detailed discussion of regularization, see [5].

2.3.3 Paulin Least-Squares Method

Fitting linear-nonlinear models in one step, specifically by fitting the linear kernel and then the static nonlinearity, has disadvantages. Estimation of the linear kernel could be improved if the nonlinearity were removed from the system, and estimation of the nonlinearity could be improved if the optimal kernel were known. These coupled constraints suggest that an iterative method may be advantageous, by alternately creating better estimates of the linear and nonlinear components in a type of bootstrapping operation.

Paulin developed an iterative algorithm for identifying LN models [3]. The key advantage of the algorithm is that, unlike previous iterative algorithms, it avoids inverting the nonlinearity $m(\cdot)$ explicitly. The original algorithm used repeated iterations of correlation-based methods, but I have adapted the algorithm to use repeated iterations of least-squares instead. A general outline of the algorithm is as follows:

1. Set the nonlinear output as the initial estimate of the linear output, $\hat{x}_0(t) = z(t)$
2. Estimate the linear kernel $h_k(\tau)$ using the LS method, with $\hat{x}_k(t)$ as the system output
3. Determine the linear response of the system $x(t) = (u * h_k)(t)$
4. Estimate the nonlinearity $m_k(\cdot)$, using $x(t)$ as the input and $z(t)$ as the output
5. Compute the mean-squared error (MSE) of the model, and if this has increased since last iteration, stop
6. Otherwise, update the estimate of the linear output

$$\hat{x}_{k+1} = x(t) + \alpha(z(t) - m_k(x(t))) \tag{18}$$

   and return to step 2.

The rate parameter $\alpha \in [0, 1]$ controls the rate of convergence. Higher α will allow the algorithm to converge quickly, but can also make it unstable. In practice, I found that a low $\alpha \approx 0.01$ ensured stability, and the algorithm still typically converged in under five iterations.

When performing the Paulin algorithm with least-squares, as described above, the computation time is not equivalent to performing the whole LS method at each iteration. This is because the autocorrelation matrix Φ_{uu} is the same for every iteration, and only needs to be computed and inverted once. Also, the parameters found for the static nonlinearity can be used as the initial conditions for the curve-fitting algorithm, allowing it to converge more quickly.

2.3.4 General Iterative Least-Squares Methods

The final type of method proposed by [3] for nonlinear system identification applies the large family of nonlinear optimization literature to the problem. Many iterative nonlinear optimization methods are based off the gradient descent algorithm, which uses the gradient of the objective (cost) function to iteratively pick parameters that minimize the objective function.

For our problem, we use a least-squares cost function

$$C(\theta) = \frac{1}{2N} \sum_{t=1}^{T} (z(t) - \hat{y}(t, \theta))^2 \tag{19}$$

where θ is a vector of all the model parameters (including parameters for both the linear kernel and static nonlinearity), $z(t)$ is the actual system output, $\hat{y}(t, \theta) = m_\theta((u * h_\theta)(t))$ is the output of the LN model, and N is the number of time points. Taking the derivative of this equation with respect to the linear kernel parameters gives the gradient

$$\frac{\partial C(\theta)}{\partial h_i} = -\frac{1}{N} \sum_{t=1}^{T} u(t)(z(t+i) - \hat{y}(t+i, \theta)) \frac{\partial m(x(t+i))}{\partial x(t+i)}. \tag{20}$$

The gradient for the linear kernel is dependent on the derivative of the nonlinearity $m(\cdot)$ with respect to its input. It is therefore important to choose a nonlinearity that is everywhere differentiable for the optimization to succeed. The LIF function used with the previous system identification methods (Equation 3) does not have this property. Instead, I chose to use a nonlinearity that I call the noisy LIF approximation function, since it is similar to the nonlinearity produced by a LIF neuron with membrane noise, which allows the LIF function to occasionally fire below its firing threshold resulting in a smooth transition from the silent regime to the firing regime. The equations for the noisy LIF approximation nonlinearity are:

$$g(x) = \left[t_{ref} + \tau_{RC} \log\left(1 + \frac{V_{th}}{Rj(x)}\right)\right]^{-1} \tag{21}$$

$$j(x) = \sigma \log\left[1 + \exp\left(\frac{a(x)}{\sigma}\right)\right] \tag{22}$$

$$a(x) = \alpha x + \beta. \tag{23}$$

where σ is analogous to the standard deviation of the membrane noise, insofar as larger σ results in a smoother function, and as $\sigma \to 0$ the function $g(x)$ approaches an LIF tuning curve, but the scaling of σ is in no way tuned to correspond to the response of an actual LIF neuron with a particular level of membrane noise. Taking the derivatives of this function with respect to the model parameters:

$$\frac{\partial g(x)}{\partial x} = \frac{\partial g(a(x))}{\partial a(x)}\alpha \tag{24}$$

$$\frac{\partial g(x)}{\partial \alpha} = \frac{\partial g(a(x))}{\partial a(x)}x \tag{25}$$

$$\frac{\partial g(x)}{\partial \beta} = \frac{\partial g(a(x))}{\partial a(x)} \tag{26}$$

$$\frac{\partial g(x)}{\partial \tau_{RC}} = -\frac{\log\left(1 + \frac{1}{j(x)}\right)}{\left[t_{ref} + \tau_{RC}\log\left(1 + \frac{1}{j(x)}\right)\right]^2} \tag{27}$$

$$\frac{\partial g(a(x))}{\partial a(x)} = \frac{\tau_{RC}}{\left[t_{ref} + \tau_{RC}\log\left(1 + \frac{1}{j(x)}\right)\right]^2 \left[j(x)(1 + j(x))\right] \left[1 + \exp\left(\frac{-a(x)}{\sigma}\right)\right]}. \tag{28}$$

These gradient equations can be used with any gradient-based optimization algorithm. For my experiments, I chose the limited-memory BFGS algorithm (L-BFGS) [6]. This algorithm is a common and general optimization routine that in addition to using gradient information generates an approximation of the Hessian (second derivatives) of the objective function. I used the implementation of the algorithm included in the open-source SciPy library [7].

2.4 LN Estimation Considerations

In addition to the algorithm used to fit a model to a system, there are a number of additional considerations when performing system identification. One consideration is the choice of input signal used to stimulate the system. It is important for the input signal to activate all the relevant system dynamics; for the ALIF neuron, this includes activating the adaptation dynamics. Another consideration that is specific to our system is firing rate estimation.

2.4.1 Input Signals

Section 2.3 examined a number of methods for system identification with linear-nonlinear models, some of which required a specific type of input (*i.e.,* white) and some of which could deal with more general types of input. Furthermore, it is possible to use a different input to identify the linear kernel than to fit the nonlinearity. Here I describe the different types of input that I experimented with for system identification of the ALIF model, namely white noise, pink noise, an equal-power signal, and a Poisson-switching signal.

White noise refers to a random signal where, in discrete time, all samples from the signal are independently and identically distributed (i.i.d.). This results in a signal with no correlation between any points in the signal. Equivalently, the signal's autocorrelation is a Dirac delta function, and thus the signal has a flat power spectral density (PSD), or equal power at all frequencies. Many types of white noise exist, depending on the probability distribution used to generate the i.i.d. samples. The three examined in this report are Gaussian white noise, where a Gaussian (normal) distribution is used, uniform white noise, where a uniform distribution is used, and binary white noise, where a Bernoulli distribution is used.

Coloured noise refers to random signals where there is a specific correlation between signal elements. Brownian noise is the integral of white noise, and has a $1/f^2$ power spectral density (-20 dB per decade). Pink noise is in between white noise and Brownian noise in terms of correlation, and has a $1/f$ power spectral density (-10 dB per decade). Different types of Brownian noise are possible, depending on the distribution of the white noise used for integration. Theoretically, different types of pink noise should also be possible, but literature on the generation of pink noise is limited, and no such methods were found. For the purposes of this work, pink noise was generated by creating Fourier coefficients with the desired PSD and a random phase component, and then transferring this signal into the time domain.

Other types of input signals can be generated to better mimic the types of signals observed by neurons. One such signal, often used by [1], is the equal-power signal. This signal has equal power at all frequencies below a given cutoff frequency, and no power at any frequency above the cutoff. This can equivalently viewed as perfectly-filtered white noise. Such signals are generated in the Fourier domain, by creating Fourier coefficients with the described PSD and random phase.

One problem with the equal-power signal is that it does not contain any higher frequencies. These higher frequencies are needed to fully activate the adaptation dynamics of the neuron; otherwise, the neuron settles to an adapted steady state, where the adaptation variable $n(t)$ remains relatively constant. To activate the adaptation dynamics of the neuron, I developed what I call the Poisson-switching signal. This signal draws values from an arbitrary distribution, but rather picking a new value every time step like white noise, the signal only switches values occasionally, and otherwise holds the previous value. The time between switches is exponentially distributed, so the number of switches in a given time period is Poisson distributed. The exponential distribution can be offset to create a minimum time between switches (*i.e.,* a maximum frequency), and its mean determines the mean time between switches (mean frequency). This process results in a signal similar to what is seen by the retina and visual cortex as the eye saccades around a scene. Though a similar signal can be created with a fixed length of time between switches, such a signal is ill-suited for system identification because it has notches in its PSD at harmonics of the switching frequency, so these frequencies are not activated in the target system.

2.4.2 Firing Rate Estimation

An important consideration when performing system identification on spiking neural systems is how to estimate the firing rate from the spiking output of the system. A number of methods have been developed, with [4] summarizing some of the basic methods, and more advanced methods are described in [8].

For this work, I experimented with both fixed kernel methods [4] and adaptive kernel methods [8], but I was not pleased with the performance of these methods and decided to use an alternate method not described in either of these sources. This method estimates the firing rate between two spikes as the inverse of the inter-spike interval (ISI) between the spikes. These measurements can then be interpolated using any interpolation method. I found that zero-order interpolation, where the rate between two spikes is constant at the inverse of the ISI between those spikes, to be both very efficient, and to yield good results in system identification. The main disadvantage of the ISI method is that it will never predict a firing rate of zero in the middle of a signal, since there will always be some finite time between the last spike and the next spike, though this did not appear to adversely affect the system identification.

3 Results

The following section presents the results of applying the LN model fitting methods Section 2.3 to the ALIF neuron model. For the ALIF model, the parameters $\alpha = 0.244$, $\beta = 1.219$, $\tau_{RC} = 0.02$, $t_{ref} = 0.002$, $\tau_n = 0.2$, and $inc_n = 0.05$ were used. The corresponding initial firing rate is 40 Hz (when the input signal is 1.0) and x-intercept is -0.9; both these parameters only apply before adaptation. The parameters were chosen because they clearly demonstrate the adaptation of the neuron with a reasonable time constant of adaptation, and otherwise roughly fit what is known about the parameters of cortical neurons.

3.1 Correlation-based Methods

Correlation-based methods are based on the assumption that the input used to identify the linear kernel is white. Using white noise to then identify the nonlinearity leads to very poor results. Figure 1 shows that the kernel itself is not the problem; the problem is with the nonlinearity. Using the same kernel with a Poisson input signal to fit the nonlinearity demonstrates that correlation-based methods can provide a LN model that captures most of the features of the neuron firing rate.

I also investigated three different types of white noise—Gaussian, uniform, and binary—to determine if this had any effect on the accuracy of system identification. Figure 2 shows that the type of white noise used does have an

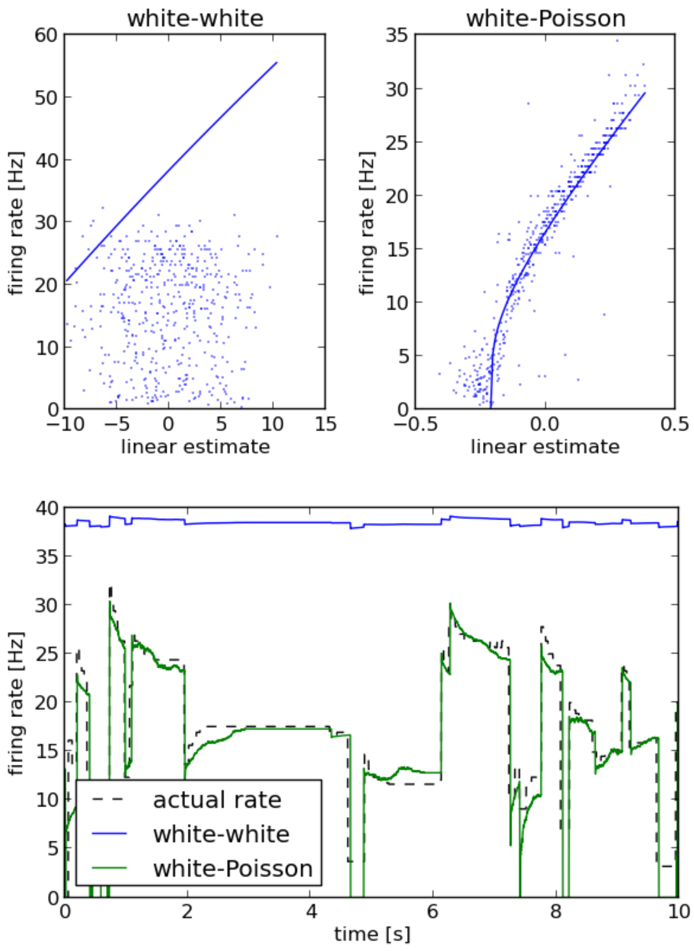

Figure 1: Comparison of using white noise versus a Poisson signal for fitting the nonlinearity. When using the correlation method, the linear kernel must be found using white noise as the input signal, but the nonlinearity can be found using any type of input. The top figures show the nonlinearity in each case, and how well it fits the points. Fitting the nonlinearity using a Poisson input produces points that are well-fit by the LIF curve; the points produced from a white input cannot be well-fit by any nonlinearity. The bottom figure shows the difference in signal reconstruction.

effect; the kernels produced by binary white noise are very different from those produced by Gaussian or uniform white noise, and results from a number of trials indicate that the performance of binary white noise is generally worse than the other types of white noise. The kernels produced by Gaussian and uniform white noise are typically quite similar, and neither type of noise appears to have a consistent advantage.

It is important to note that though the kernels approach zero for large delays (time values), when examined closely the average of most kernels over long time periods ($\tau \in [0.1, 1]$) is slightly below zero. This is how the kernels achieve the smooth firing rate decrease over time seen in the signal reconstructions from the Gaussian and uniform kernels (Figure 2, bottom). Interestingly, the binary kernel does not show this smooth decrease, indicating that unlike the other kernels, it approaches zero quickly for $\tau > 0.05$.

3.2 Least-squares Method

The least-squares (LS) method of system identification is able to offer significantly better results than the correlation based methods, by taking into account the exact autocorrelation of the input signals, rather than assuming they obey an ideal autocorrelation. One of the first steps in applying the LS method is to determine the amount and type of regularization to apply when solving the least-squares problem. I investigated three types of regularization: zero-order, first-order, and second-order. Of the three, second-order is the most intuitive for this problem, and also yields slightly better results. The optimal level of regularization depends on the amount of input data used in the system identification, specifically the number of trials n and length of each trial t. Generally, using more data results in a better estimate of the kernel, so less regularization is necessary because there is less chance of over-fitting. Despite this dependence on the amount of input data, setting $\lambda \in [10^1, 10^2]$ provided good results for lower number of trials ($n = 10$ trials each $t = 10$ seconds, see Figure 3), and did not have an adverse effect on performance for large numbers of trials ($n > 100$), when regularization was unnecessary.

The type of input used for system identification can greatly affect the results. I tested the LS method using four different types of input—white noise, pink noise, equal-power, and Poisson-switching—to identify the linear kernel. The nonlinearity was always found using a Poisson-switching input, so that the effects of each input could be evaluated only on their ability to fit

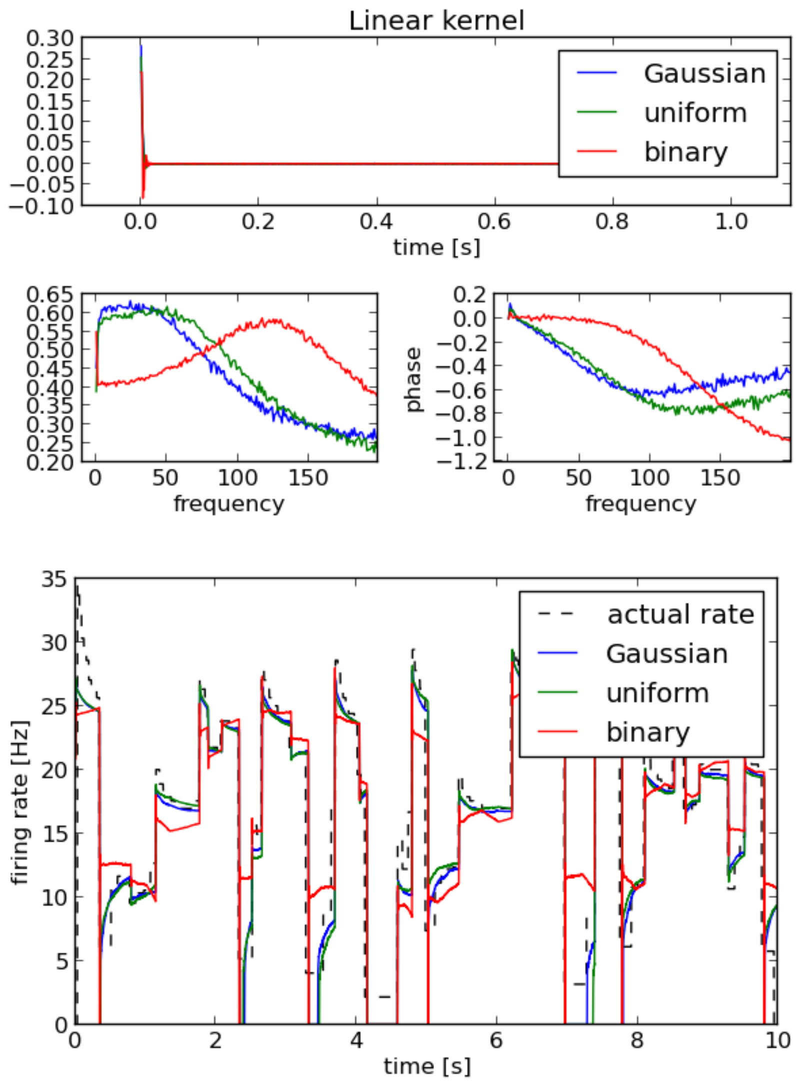

Figure 2: Effects of using different types of white noise in correlation-based system identification. Using binary white noise to fit the LN model produces less accurate kernels than using Gaussian or uniform white noise.

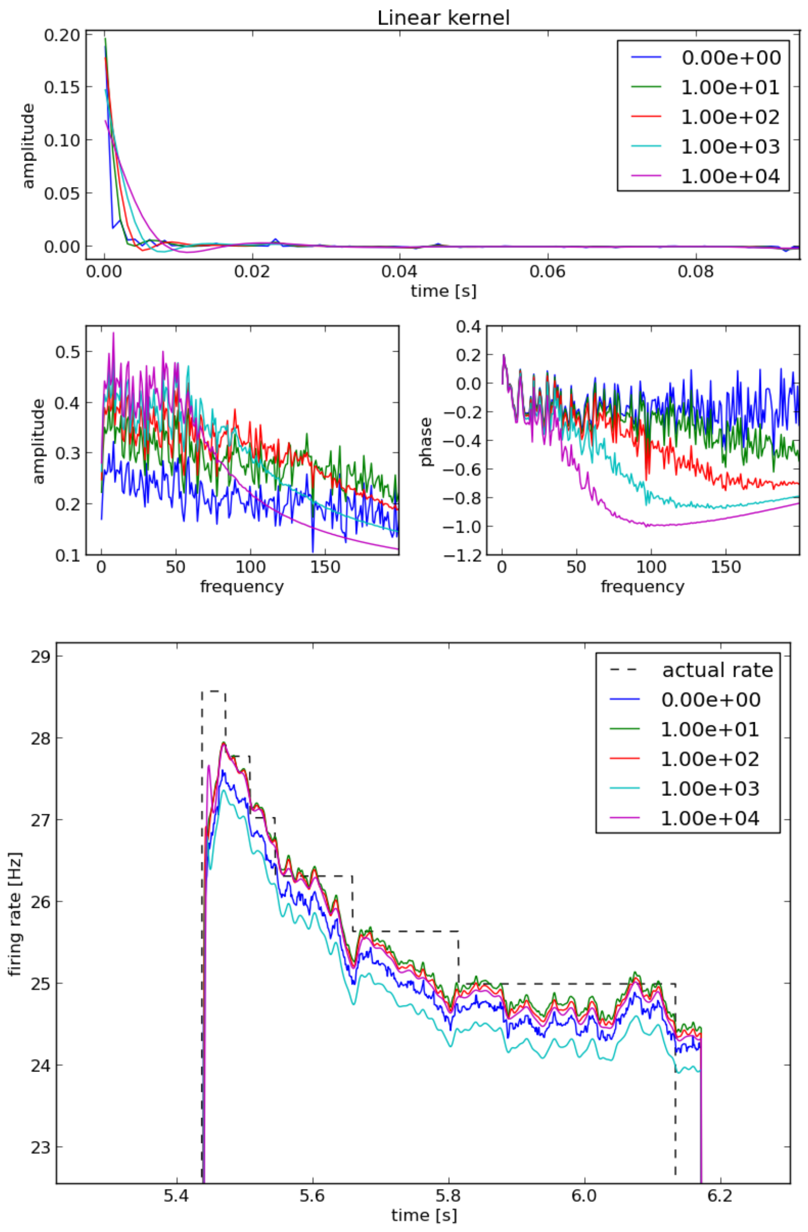

Figure 3: The effects of different levels of second-order regularization. Top: Second-order regularization penalizes changes in slope. The kernel with no regularization (blue) has sharp spikes, increasingly regularized kernels are flatter. **Bottom:** A moderate level of regularization ($\lambda \in [10^1, 10^2]$) is optimal for fitting this new signal.

	white	pink	equal-power	Poisson
equal-power test	6.89	6.43	8.37	6.20
Poisson test	3.72	3.60	7.66	3.39

Table 1: Root mean-squared errors for LN models fit with various types of input (columns), when tested using two different types of input (rows). The type of input used can greatly affect the identification results; the kernels identified using Poisson-switching noise generally show the best results.

the linear kernel. (As we have previously seen, some types of input, namely white noise, are very poor at fitting the nonlinearity.)

The results of these experiments are shown in Figure 4. The largest difference in kernel is between the equal-power input and the other types of input: the equal-power input creates a kernel with a large degree of resonance around 40-50 Hz, resulting in an oscillatory estimate of the neuron's firing rate. The results of a quantitative comparison of the algorithms are shown in Table 1. These results show that using Poisson-switching inputs produces the best results when tested with both Poisson-switching inputs and equal-power inputs. This suggests that Poisson-switching inputs are best able to activate all modes of the neural dynamics, capturing both the adaptive and non-adaptive properties of the ALIF neuron, creating a robust kernel. One concern related to the results is that using white noise, pink noise, and Poisson-switching inputs all result in good performance, but the kernels found from using Poisson-switching inputs is quite different from those found by white and pink noise. It is possible that one of the types of inputs is capturing additional neural dynamics that are missed by the others, and that if the kernels are tested on additional types of input (other than the equal-power and Poisson-switching inputs used for testing), one kernel will perform significantly better than the others.

3.3 Iterative Methods

Ideally, the iterative methods should be able to provide better results than the LS method, since the LS method is biased by the nonlinearity in the system. The Paulin method does show an improvement over LS, especially when using a limited amount of data for the system identification. A dramatic result for one such neuron is shown in Figure 5. The first iteration of LS provides a relatively poor fit of the system. The second iteration shows

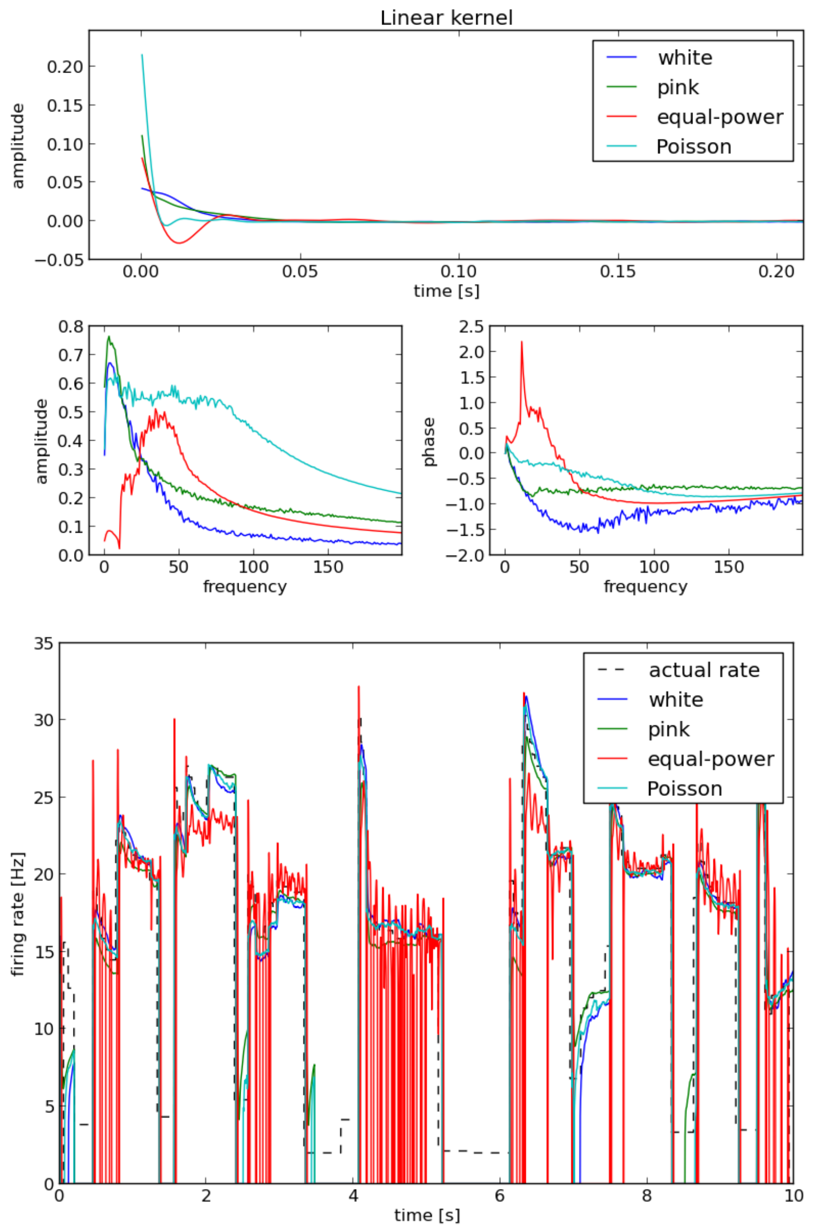

Figure 4: Results of using different types of input for fitting the linear kernel with the LS method. Using different types of input results in significantly different inputs (top). Kernels identified using white noise, pink noise, and Poisson-switching inputs all perform well, but those identified using an equal-power input show oscillations (bottom).

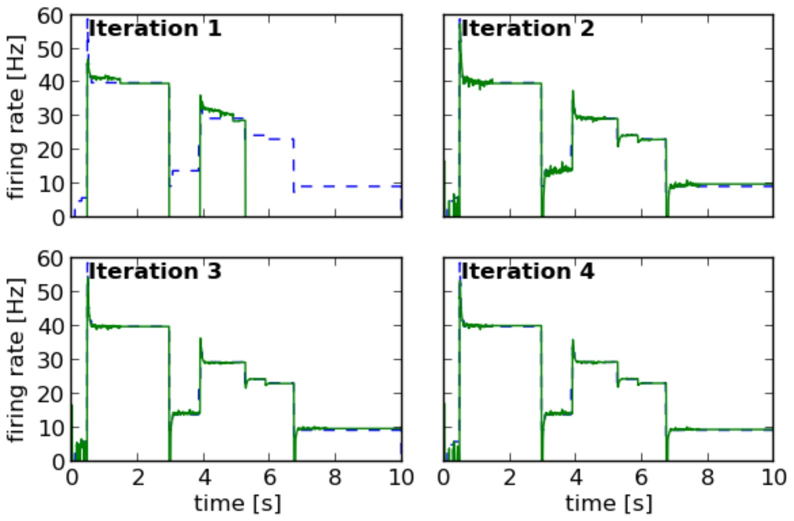

Figure 5: An example of the improvement from repeated Paulin iterations. The first iteration provides only a rough fit of the data. The second iteration provides a much better fit, and the third and fourth iterations offer further fine-tuning.

significant improvement, to the point that the LN model captures all the relevant neural dynamics. The third and fourth iterations perform fine-tuning to remove some of the noise in the kernel. Though the Paulin method does not show such dramatic improvements on all neurons, it is not uncommon. Furthermore, it will always reduce the MSE of the LN model fit, because the MSE is measured after each iteration, and the algorithm only continues if performance continues to increase (*i.e.,* MSE decreases).

One possible drawback to the algorithm is overfitting the training data, but this does not seem to be a problem in practice; the kernels generated from the Paulin method generalize well to other inputs of the same type and of other types. Another drawback of the algorithm is that it takes longer than the LS method. The additional computational cost can be minimized by forming and inverting the autocorrelation matrix Φ_{uu} at the beginning of the algorithm since this matrix is common across all iterations. This computation is the rate-limiting step of the least-squares algorithm, making the Paulin LS algorithm is only slightly slower than the LS algorithm, since additional Paulin iterations are cheap.

The general method for nonlinear least-squares, using the L-BFGS algorithm of optimization, did not perform well. If starting with untuned parameters (either random parameters, preset parameters, or some combination of the two) the algorithm generally did not converge to a solution that was at least as good as the LS solution. Often the algorithm would find a solution roughly approximated the actual neuron firing rate, and then get stuck in a local minimum and terminate. I also applied the algorithm to solutions already found by the LS algorithm, to see if the general nonlinear algorithm was able to fine-tune these solutions. The general nonlinear algorithm offered an improvement to the performance metric (MSE) in such cases, but the improvement was small and the difference in the kernel and nonlinearity insignificant.

3.4 Population identification

The Paulin method was found to be the most accurate of the system identification methods investigated, and is only slightly more expensive than the least-squares method. I applied the Paulin method to the problem of identifying the kernels of a small population of 20 neurons. The ALIF neuron parameters were the same as those used in the previous experiments, except the adaptation parameters were now chosen randomly from the uniform intervals $\tau_n \in [0.05, 0.30]$ and $inc_n \in [0.05, 0.5]$. These parameters provide a wide range of adaptation behaviour on the biologically-plausible time-scale of hundreds of milliseconds.

When fitting LN models to a population of neurons, the input signal must be chosen either so that it is able to activate the relevant dynamics of all possible neurons in the population, or individual input signals must be chosen for each neuron based on that neuron's parameters. For these experiments, a Poisson-switching input was used for the identification of both the linear kernel and the static nonlinearity, with a mean frequency of 1 Hz and a maximum frequency of 10 Hz. This mean frequency is lower than that used in previous experiments (3 Hz), because it improved the fit of many of the neuron models, especially those with higher values of inc_n.

The kernels found by the population identification are shown in Figure 6. When viewed in the frequency domain, all the kernels appear to be performing band-pass filtering. The low-frequency cutoff for all the kernels is approximately the same, approaching zero only for very low frequencies. The high-frequency cutoff varies greatly between kernels, with some kernels hav-

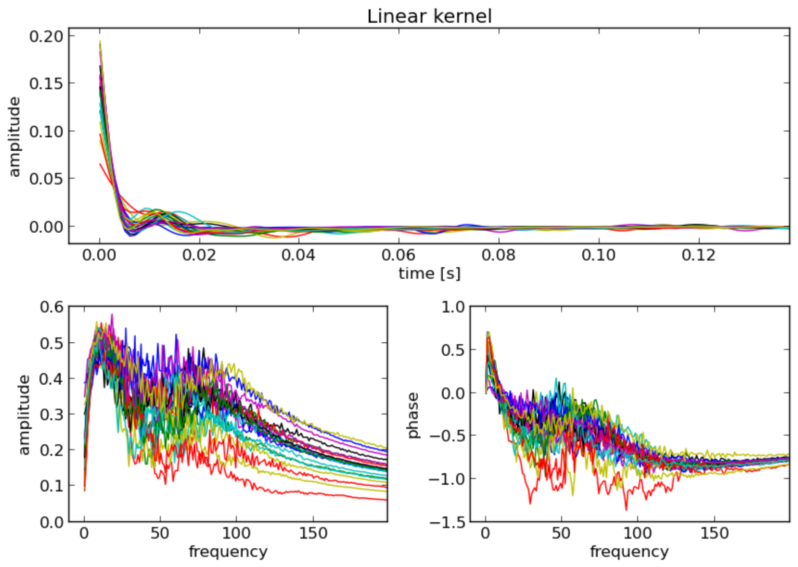

Figure 6: Linear kernels for a population of 20 neurons. Linear kernels for a population in the time and frequency domains. The kernels all act as bandpass filters. the bottom cutoff frequency remains relatively constant between filters, whereas the top cutoff frequency varies largely (bottom left).

ing diminished amplitude for all frequencies above 20 Hz, and other kernels allowing all frequencies under 100 Hz to pass equally. At around 100 Hz is where the the second-order regularization begins to come into effect, and all kernels decay smoothly above this frequency. Many kernels show a mild double-resonance phenomenon, with peaks in the power spectrum both at a lower frequency ($\sim$ 20 Hz) and at a higher frequency ($\sim$ 70 Hz).

4 Conclusions

This report demonstrated that system identification techniques can be used to fit linear-nonlinear models to the adaptive leaky integrate-and-fire neuron model, and that the linear-nonlinear model is able to capture all the relevant adaptation dynamics of the neuron model, especially when fit using the Paulin least-squares algorithm. System identification was performed on all neurons in an example population, and showed that neurons in the population act as band-pass filters.

This work provides the first steps to extending the Neural Engineering Framework to neuron models with complex dynamics. The next step is to use the identified linear-nonlinear models to inform the process of solving for the linear neuron decoders. The user will now have to specify a desired output not only in terms of a desired output static nonlinearity—as in the current formulation of the NEF—but also in terms of desired output dynamics. It remains unclear how best to solve for decoders that will simultaneously achieve the desired linear dynamics and nonlinearity of the population. Other future work includes extending the system identification techniques developed here to other neuron models, for example the Izhikevich neuron model [9] with bursting dynamics. I also observed that different types of input (*e.g.,* pink noise versus Poisson-switching) can identify significantly different kernels which are both able to provide a good approximation of the neuron's firing rate. Further work is needed to determine if one kernel is capturing more of the neuron's dynamics than the other.

References

[1] C. Eliasmith and C. H. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems.* Cambridge, MA: MIT Press, 2003.

[2] C. Koch, *Biophysics of computation: Information processing in single neurons.* New York, NY: Oxford University Press, 1999.

[3] D. Westwick and R. Kearney, *Identification of nonlinear physiological systems.* Piscataway, NJ: IEEE Press, 2003.

[4] P. Dayan and L. F. L. Abbott, *Theoretical neuroscience: Computational and mathematical modeling of neural systems.* Cambridge, MA: MIT Press, 2001.

[5] P. Fieguth, *Statistical Image Processing and Multidimensional Modeling.* New York, NY: Springer, 2010.

[6] D. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical programming*, vol. 45, pp. 503–528, 1989.

[7] E. Jones, T. Oliphant, P. Peterson, and Others, “SciPy: Open source scientific tools for Python,” 2001. [Online]. Available: http://www.scipy.org

[8] J. Cunningham, V. Gilja, S. Ryu, and K. Shenoy, “Methods for estimating neural firing rates, and their application to brainmachine interfaces,” *Neural Networks*, vol. 22, no. 9, pp. 1235–1246, 2009.

[9] E. M. Izhikevich, “Simple model of spiking neurons,” *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–72, jan 2003.