Implementation of the Density Evolution method with Fokker-Planck for L-NLIF Model in Python

Valentin Haenel
valentin.haenel@gmx.de
BCCN Berlin

February 23, 2009

1 Introduction

This report is a description of the work done by Valentin Haenel for a Lab Rotation within the Bernstein Center for Computational Neuroscience in Berlin. The rotation was completed under the supervision of Gabriel and Bartosz Telenczuk.

Throughout the project I have implemented a variant of the Leaky Integrate and Fire Neuron (LIF), and a method to fit the parameters of this neuron using a model of the input and extracellularly recorded spikes. Primarily this involved solving a partial differential equation and constructing a maximum likelihood objective function.

This project is motivated by the fact that recording intracellularly from awake behaving animals is usually not possible, and extracellularly recorded spike trains are much more common in this setting. The model implemented allows us to gain at least some partial insight into the subthreshold voltage of such neurons. A further motivation comes from the fact, that algorithms described are not publicly available in the form of an easy to use software package.

2 Model

The original model is a L-NLIF model, which consists of a Linear (L) Filter, followed by a probabilistic or Noisy (N) form of Leaky Integrate and Fire spike generation (LIF) (Paninski et al.[2]). For sake of completeness the model is briefly described here.

The evolution of the Voltage V is given by:

$$dV = (-g(V(t) - V_{leak}) + I_{stim}(t) + I_{hist}(t))dt + W_t \tag{1}$$

Where g is the leak conductance V_{leak} is the leak reversal potential I_{stim} is the convolution of linear filter with the input signal, I_{hist} is the spike current history and W_t is standard Gaussian white noise. More precisely:

$$I_{stim}(t) = \vec{k} \cdot \vec{x}(t) \tag{2}$$

$$I_{hist}(t) = \sum_{j=0}^{i-1} h(t - t_j) \tag{3}$$

Where k is a linear filter and h is a fixed postspike current waveform.

3 Algorithms

3.1 PDE Solver

3.1.1 The Problem

Let

$$P(V,t) \equiv P(V(t) \cap V(s) < V_{th} \forall s < t) \tag{4}$$

This refers to the probability of $V(t)$, the membrane potential, being less than V_{th} the firing threshold until time t.

As stated in [2] the method of density evolution requires us to solve the following Fokker-Planck drift diffusion equation numerically.

$$\frac{\partial P(V,t)}{\partial t} = \frac{\sigma^2}{2}\frac{\partial^2 P(V,t)}{\partial V^2} + g\frac{\partial[(V - V_{rest})P(V,t)]}{\partial V} \tag{5}$$

We will be solving the evolution of the probability density for a given inter-spike interval (ISI). And therefore we will constrain the range of V and t.

$$P(V_{th}, t) = 0 \tag{6}$$

$$P(V, 0) = \delta(V - V_{reset}) \tag{7}$$

Condition (6) ensures that the probability of obtaining a spike during an ISI is zero. Condition (7) ensures that at the beginning of an ISI, i.e. at the time of the last spike, the probability of the neuron being at reset potential is exactly one. Thereby we have obtained the constraints for the top and the left hand side of our solution grid, however, when solving numerically we also need a lower bound on the voltage V_{lb}, so this becomes an additional boundary condition, for the bottom:

$$P(V_{lb}, t) = 0 \tag{8}$$

Ideally $V_{lb} = -\infty$.

V_{rest} is defined as the stationary point of the noiseless subthreshold dynamics:

$$V_{rest}(t) = V_{leak} + \frac{1}{g}(\vec{k} \cdot \vec{x}(t) + \sum_{j=0}^{i-1} h(t - t_j)) \tag{9}$$

3.1.2 Finite Differencing

In order to solve (5) numerically we discretized time and potential. We adopt the notation that Potential is discretized into W intervals of length w and indexed by $\nu = 0, 1, \dots W$. Time is discretized into U intervals of length u and indexed by: $\tau = 0, 1, \dots U$. We adopt the notation: $P_{\nu,\tau} = P(\nu w, \tau u)$

A forward finite differencing scheme would like:

$$\hat{P}(v,t) = P_{\nu,\tau} \tag{10}$$

$$\frac{\partial \hat{P}(v,t)}{\partial t} = \frac{P_{\nu,\tau+1} - P_{\nu,\tau}}{u} \tag{11}$$

$$\frac{\partial \hat{P}(v,t)}{\partial V} = \frac{P_{\nu+1,\tau} - P_{\nu-1,\tau}}{2w} \tag{12}$$

$$\frac{\partial^2 \hat{P}(v,t)}{\partial V^2} = \frac{P_{\nu+1,\tau} - 2P_{\nu,\tau} + P_{\nu-1,\tau}}{w^2} \tag{13}$$

However as suggested in [1] we will use the computationally more stable Crank-Nicolson scheme [3]. We rewrite the derivatives using the new scheme which is centered around $t + u/2$.

$$P_{CN}(v,t) = \frac{P_{\nu,\tau} + P_{\nu,\tau+1}}{2} \tag{14}$$

$$\frac{\partial P_{CN}(v,t)}{\partial t} = \frac{P_{\nu,\tau+1} - P_{\nu,\tau}}{u} \tag{15}$$

$$\frac{\partial P_{CN}(v,t)}{\partial V} = \frac{P_{\nu+1,\tau} + P_{\nu+1,\tau+1} - P_{\nu-1,\tau} - P_{\nu-1,\tau+1}}{4w} \tag{16}$$

$$\frac{\partial^2 P_{CN}(v,t)}{\partial V^2} = \frac{P_{\nu+1,\tau} - 2P_{\nu,\tau} + P_{\nu-1,\tau} + P_{\nu+1,\tau+1} - 2P_{\nu,\tau+1} + P_{\nu-1,\tau+1}}{2w^2} \tag{17}$$

3.1.3 Tridiagonal Equations

If we now let:

$$a = \frac{\sigma^2}{2}$$
$$b = g(V - V_{rest})$$
$$c = g$$

and multiply throughout with $4w^2u$ we may rearrange all the $P_{*,\tau+1}$ terms on the left hand side:

$$\overbrace{-(2au + bwu)}^{A_\nu} P_{\nu+1,\tau+1} + \overbrace{(4au - 2cw^2u + 4w^2)}^{B_\nu} P_{\nu,\tau+1} \overbrace{-(2au - bwu)}^{C_\nu} P_{\nu-1,\tau+1} = \underbrace{(2au + bwu)P_{\nu+1,\tau} + (-4au + 2cw^2u + 4w^2)P_{\nu,\tau} + (2au - bwu)P_{\nu-1,\tau}}_{D_\nu} \tag{18}$$

For each $\nu = 1, \dots, W-1$

Using (18) and incorporating the boundary conditions 6 and 7 we have obtain a complete system of $W-1$ linear equations which we may now rewrite in the following tridiagonal matrix notation.

$$\underbrace{\begin{pmatrix} B_1 & A_1 & 0 & \cdots \\ C_2 & B_2 & A_2 & 0 \\ & \ddots & \ddots & \ddots \\ \cdots & 0 & C_{W-1} & B_{W-1} \end{pmatrix}}_{\Lambda} \underbrace{\begin{pmatrix} P_{1,\tau+1} \\ P_{2,\tau+1} \\ \vdots \\ P_{W-1,\tau+1} \end{pmatrix}}_{\chi} = \underbrace{\begin{pmatrix} D_1 \\ D_2 \\ \vdots \\ D_{W-1} \end{pmatrix}}_{\beta} \tag{19}$$

We can then use a tridiagonal matrix algorithm to solve $\Lambda\chi = \beta$. This allows us to iteratively obtain the density evolution $P(V,t)$ of the membrane potential for a given ISI. Where the initial β for an ISI is given by (7) and the halting criterion is the time of the next spike. So for the ith spike we obtain values for V in the range $[V_{lb}, V_{th}]$ and for t in the range $[t_{i-1}, t_i]$. An example can be seen in figure 1.

3.2 First Passage Time

The maximum likelihood optimizer proposed in [2] however uses the first passage time density at a given time point which is defined as:

$$\text{FPT}(t) = -\frac{\partial}{\partial t} \int P(V,t)dV \tag{20}$$

and this can be easily computed from a $P(V,t)$ matrix by a finite differences / sums scheme.

3.3 Maximum Likelihood Estimator

Putting together the PDE Solver and method for computing the First Passage Time The final likelihood is then simply

$$L_{\vec{x},t_i} = \prod_i \text{FPT}(t_i) \tag{21}$$

This is the product of the first passage time at the times of spiking for all spikes.

The pseudocode for the objective function can be described as follows:

```
for each spike interval do:

  1. compute $P(V,t)$ as matrix
  2. compute FPT$(t)$ as vector
```

```
return the product of the last scalars in the FPT vectors.
```

This objective function could be presented to a standard numerical optimizer, for example the Nelder-Mead Downhill simplex [3], this needs no gradients and is fairly robust but slow.

3.4 Monte Carlo Method

To double check the results of the PDE Solver and the resulting First Passage Time, we additionally implemented some basic Monte Carlo simulations. The basic idea is to simulate a few thousand traces of the neuron up to the first spike, with noise. The resulting potential traces, and spike times were used to obtain both $P(V,t)$ and fpt. The density evolution is compared qualitatively to the density evolution technique as can bee seen in: 1. The FPT is compared using a Kolmogorov-Smirnov (K-S) test[3].

4 Implementation

This section describes the concrete implementation

4.1 Availability and Design

The algorithm has been implemented in Python, and the full source code has been made available within a revision control system under a free software license at: http://github.com/esc/molif/tree/master. Given that the Python Language was named after the British Comedians Monty Python we decided to call the software *Meaning Of LIF*, (abbreviated to *molif*) which is a reference to the Monty Python movie *Meaning of Life*.

The software is structured into a series of modules contained within the package `molif`

- `model` implementation of the neuron model
- `density` implementation of the PDE solver
- `likelihood` connects the optimizer and the PDE solver
- `montecarlo` implementation of Monte Carlo simulator
- `plotting` methods used to generate plots
- `util` methods to determine timing

The software uses the additional packages numpy, scipy for vector based and scientific computing, and pylab/matplotlib for plotting the results. We used the modules *scipy.sparse* and *scipy.linsolve* for computing the solution to tridiagonal equations, and the method *optimize.fmin*, which implements the Nelder-Mead Downhill Simplex, for minimizing the negative likelihood.

4.2 Limitations and Future Work

The PDE solver uses sparse matrix algorithms to store and solve the tridiagonal matrix. Ideally we would like to use the routine $TRIDAG$ described in [3], however an open source implementation in python was not available, although feasible. This improvement could possibly speed up the computation of $P(V, t)$.

There is a serious bug, whereby the value of sigma, must be within a specific range of $0.1 - 0.05$, otherwise strange oscillatory instabilities occur. Among the possible causes, are a numerical instability in the finite differencing scheme, a bug in our code, a bug in the solver for the linear equations or the discretization of V and t being to large.

Currently the gradient of the objective function isn't implemented, and any gradient based optimizer would need to compute their approximation from the function values.

We used a Kolmogorov-Smirnov (K-S) test to check that the first passage time computed by the PDE-solver and the Monte Carlo method come from the same distribution. However the test currently returns a negative result, although at first glance the distributions look very similar.

Lastly we should mention that the objective function isn't fully operational yet. A simple test is executed as follows: first we generate some spikes with given parameters, and then we attempt to fit parameters to these spikes. If the code operates correctly the optimizer should not deviate away from the initial parameters vector. However currently this is not the case and the optimizer finds a minimum at a different position in parameter space.

4.3 Results

This section presents some plots.

Figure 1 shows a comparison of PDE Solver (top) and Monte Carlo (bottom) method when computing the density evolution. As expected the two plots look the same, and corresponds very nicely to what we would expect from the iterative solution to a Fokker-Planck equation. The drift term is responsible for the shift of the mean, and the diffusion is responsible for the flattening.

Figure 2 shows the comparison of the PDE Solver and Monte Carlo method when computing the first passage time. Unfortunately a K-S test allows us to reject the hypothesis that these two come from the same distribution, even though they look very much alike at a first glance.

5 Conclusion

Not all of the initial Project Goals were reached to my full satisfaction. We did implement an Integrate and Fire neuron, and a method to fit the parameters, however we did not get as far as actually fitting real experimental data. None the less useful code has been produced and solid numerical techniques (Solving Partial Differential Equations) were learnt, and I consider the lab rotation to be successful.

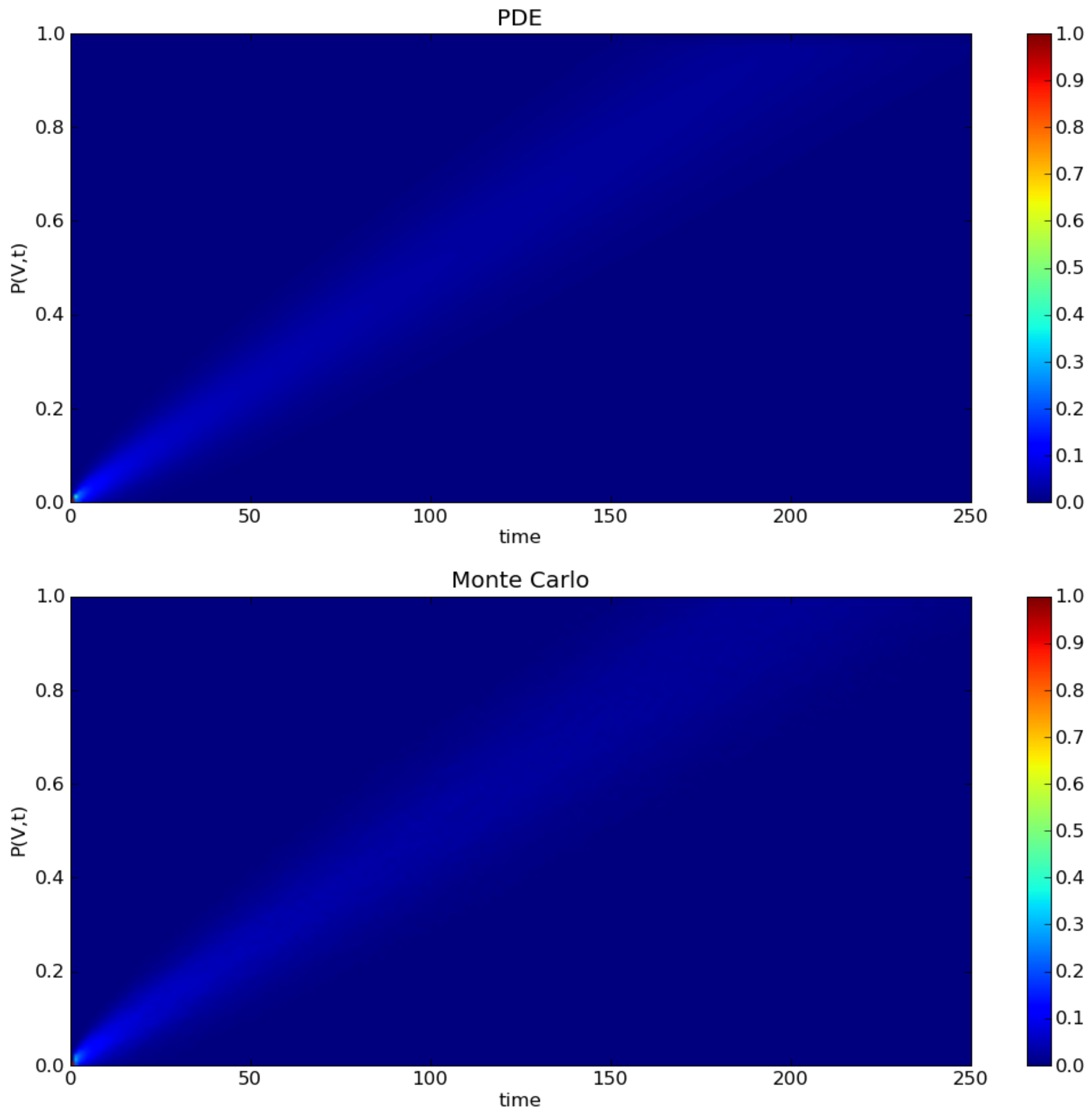

Figure 1: Density Evolution

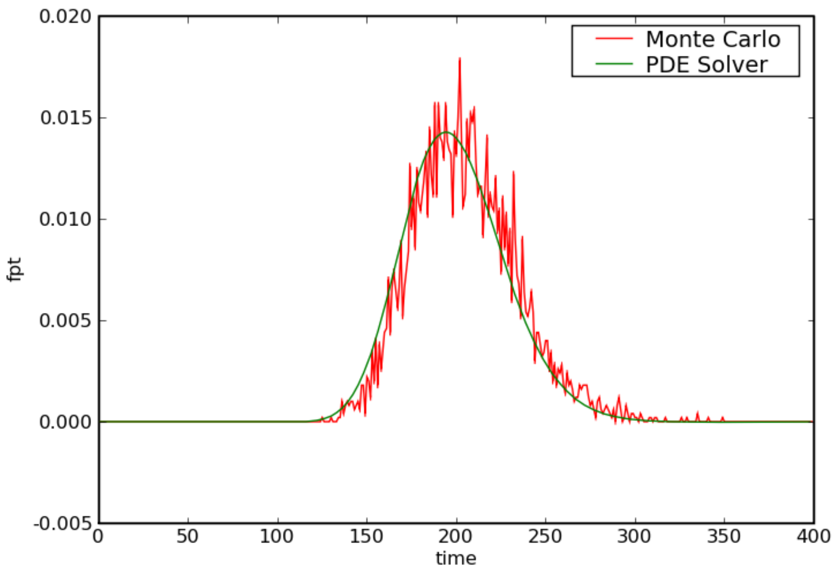

Figure 2: First Passage Time Comparison

References

[1] L. Paninski, A. Haith, and G. Szirtes. Integral equation methods for computing likelihoods and their derivatives in the stochastic integrate-and-fire model. *J Comput Neurosci*, 24(1):69–79, Feb 2008.

[2] L. Paninski, J. W. Pillow, and E. P. Simoncelli. Maximum likelihood estimation of a stochastic integrate-and-fire neural encoding model. *Neural Comput*, 16(12):2533–2561, Dec 2004.

[3] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 2nd edition, 1992.