Contents

1 2/1/17

1.1 Epileptor paper | Jirsa, 2014

Describes the epileptor model for seizure activity [1]. The model is derived by considering invariant properties of seizures: spike frequence and amplitude trends during siezure onset and offset. They consider SLEs to be within the standard repertoir of brain states explaining why they are associated with such a wide range of afflictions. They consider that the brain then passes a bifurcation point when starting or ending an SLE. Interictal states are modeled by oscillatory states (limit cycles) and non-SLE states are in the space of a stable equilibrium point. SWEs in the preictal interval can be seen as a result of the state approaching a bifurcation point.

The model is governed by 3 ensembles comprising 5 total equations. The first ensemble is composed of two equations and deals with fast discharges (fast time scale). The second ensemble is again two equations, but now at a longer time scale and represented SWEs. The third ensemble is a single variable, called the permittivity variable, z. This variable determines how near the current state is to a seizure onset. The permitivity variable operates on a very long time scale.

The Virtual Brain Project has a GUI and a bunch of code for playing with this model - they add in some additional parameters.

In the paper, the model equations will produce a plot that looks more like the one in the paper if the g coefficient is set to 2 (instead of 0.002). Additionally, I think the traces shown are $-x_1 + x_2$ (rather than $x_1 + x_2$).

1.2 Reproducing the Model

Since the equations, parameter values, and initial conditions for the model were all given in the paper, it is easy enough to reproduce their plots (with the minor alteration in the g

coefficient mentioned above). Compare the plots in Figure 1.

[hb]

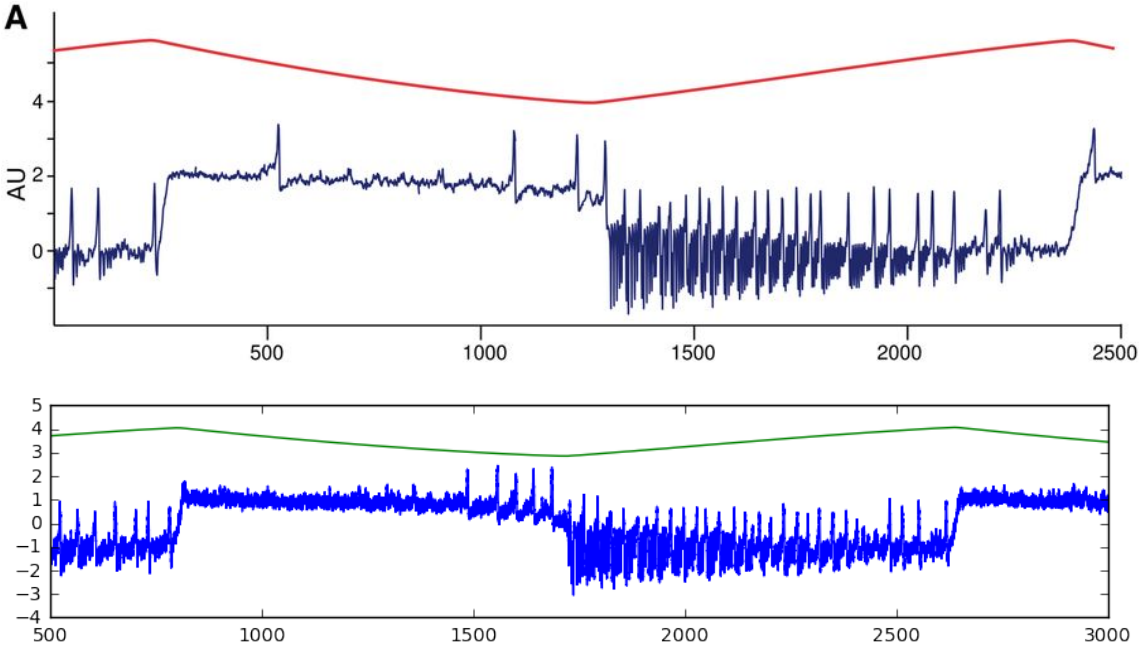

Figure 1: Comparison of model outputs. The top figure is Figure 5a from Jirsa; the bottom figure was generated using the *sdeint* package for integrating stochastic differential equations in Python. The trace shows the sum $-x_1 + x_2$.

Note that there are a few differences: 1) the cycle seems a bit faster for mine and the initial conditions may differ - in particular, I cut off the first 500 seconds in mine to align the traces. 2) The x_2 spike frequency seems a bit higher in mine and more regular (see Fig.2).

[htbp!]

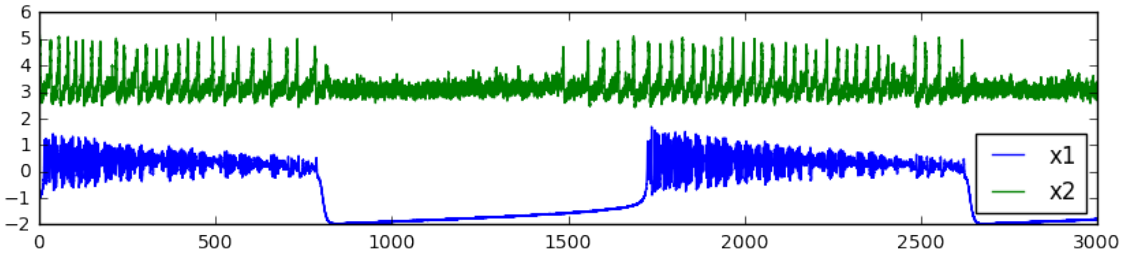

Figure 2: Variables x_1 and x_2 separated.

1.3 Next steps

- Set up Kalman filter
- Look over BluePyOpt paper [4]

- Would like to read about stochastic calculus (Ito vs. Stratonovich) and review different integration methods. It looks like in the VBP they switch to integrating using the Heun method (implicit) - supposed to allow larger step sizes. Check [2] for both of these.

2 2/11/17

2.1 Checking the Kalman filter

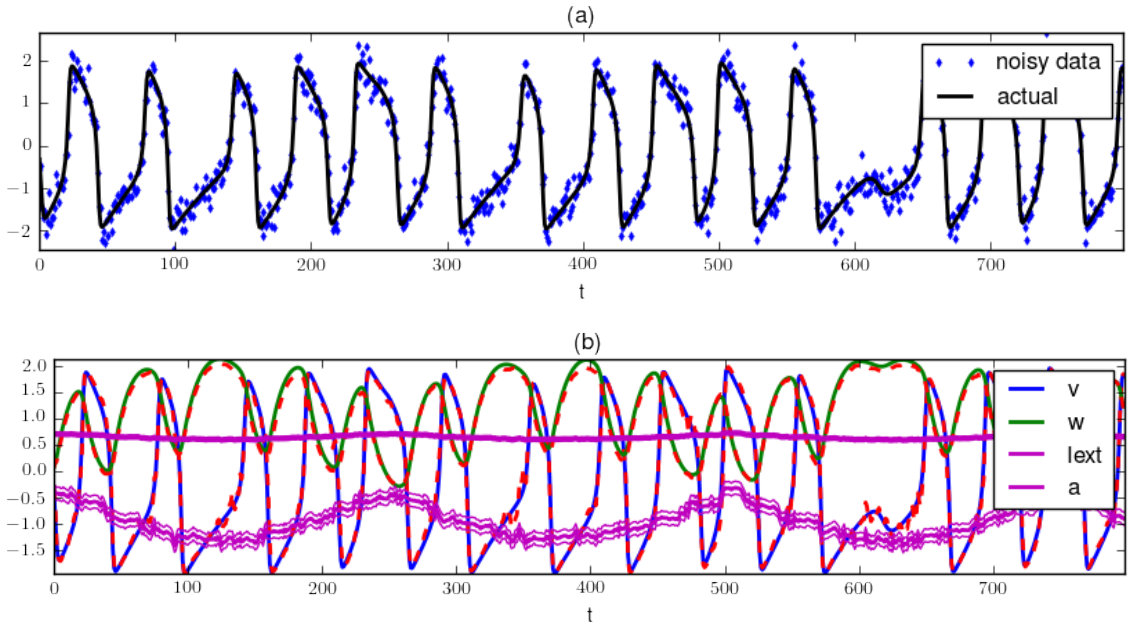

Figure 3: Testing Kalman filter. *(Top)* True trajectory based on Fitzhugh-Nagumo model (black) and noisy observations. *(Bottom)* Kalman filter tracking of variables (red) and parameters (magenta). Confidence intervals are shown for the parameters. Process noise for the parameters was set to $[0.015, 0.0015]$ for $Iext$ and a, respectively. Initial conditions for the Kalman filter use true values of variables and parameters.

Parameter estimation gets worse with more parameters. With only one parameter, the Kalman filter tracks the hidden state and parameters well (Fig.5). Extending the trial to longer times and increasing noise does not seem to improve estimation (not shown). Additionally, the two estimated parameters will track similar shapes if the noise is set the same - splitting the difference in the estimated functions.

2.2 Testing the Kalman filter on an Epileptor simulation

... I should turn the filter into a separate Python function so I can just run all of these things in the Notebook and have everything there are ready to go rather than using Tex...

The top plot in Fig.6 shows the output of $-x1+x2$ from a non-noisy Epileptor simulation in black. The blue show noisy observations based on the black trace where the level of noise $0.04 * \sigma^2_{sim}$ (same as in Schiff/Voss). Using the noisy data and setting all initial estimates

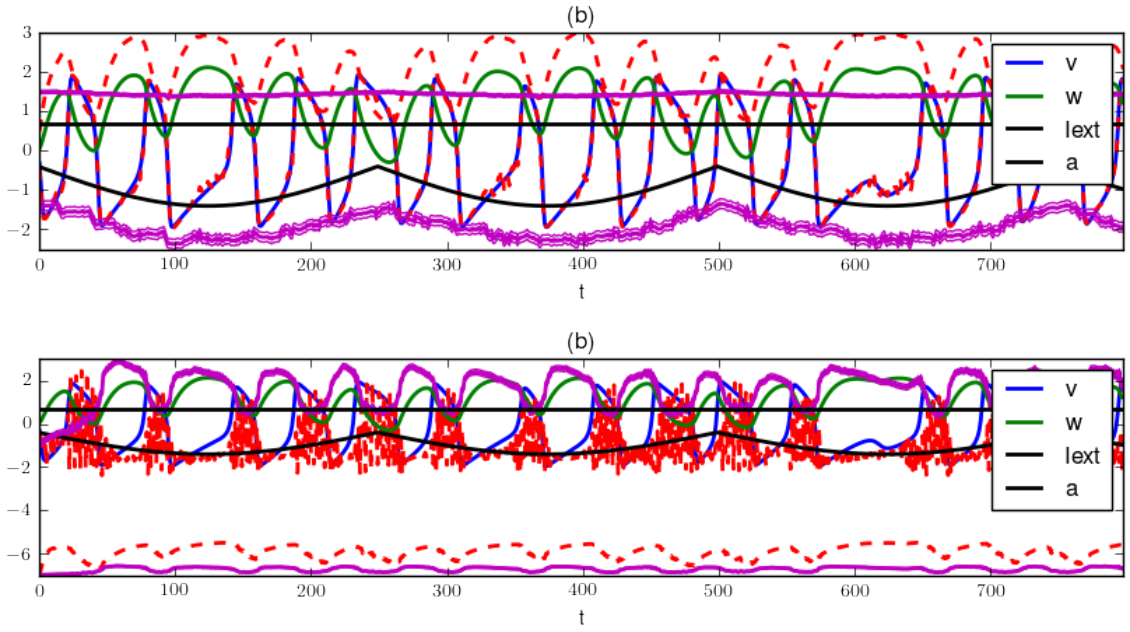

Figure 4: Kalman filter with bad initial estimates. *(Top)* Initial estimates of hidden variable and parameters are within 1.5 of actual values. *(Bottom)* Initial estimates here are within 10 of actual values. In both cases, the initial value of the observable was set to the simulated noisy initial point and actual values of the parameters are shown in black.

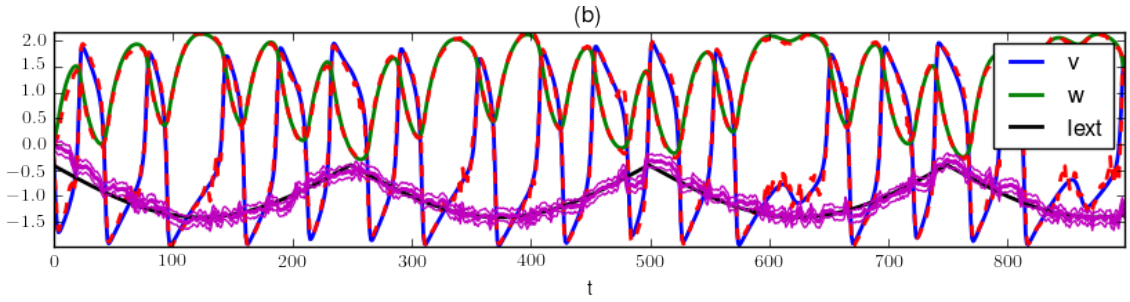

Figure 5: Kalman tracking of a single parameter. This is the figure that is shown in Voss [5] and Schiff [3], chapter 5. The filter quickly generates a pretty tight fit around the $Iext$ parameter trace.

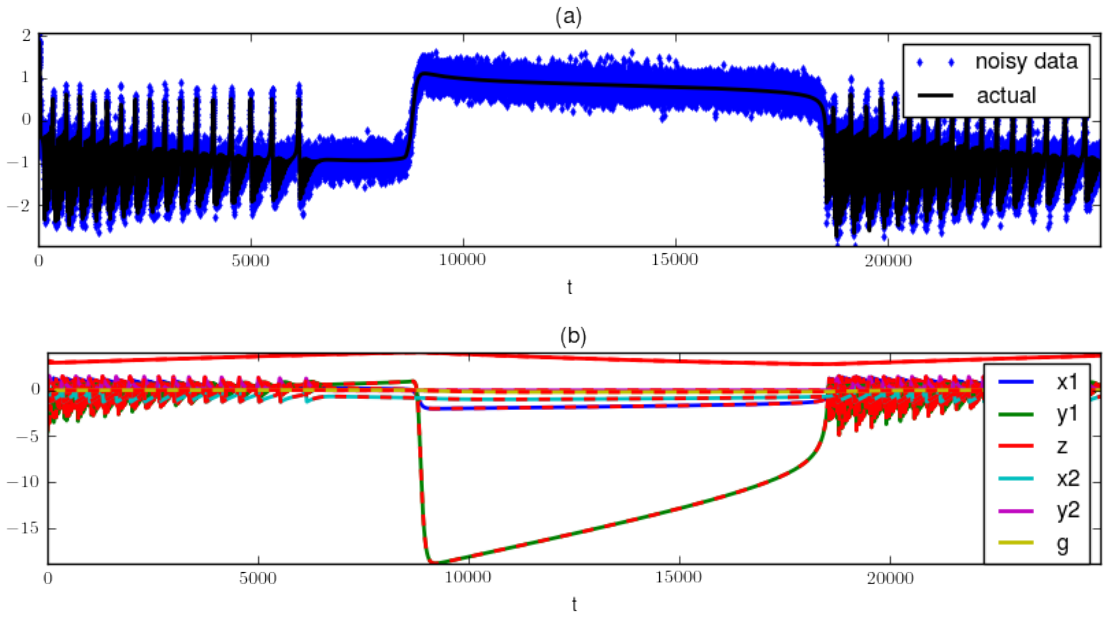

Figure 6: Tracking hidden states of the Epileptor. Just showing that the filter can track the hidden states (no parameters here) given a perfect start (initial estimate matches initial conditions). Did not verify that the red dashed lines correspond to the correct traces...

to the actual initial values yields the lower plot in Fig.6. It might be worth looking at how initial conditions and noise affect these traces at some point.

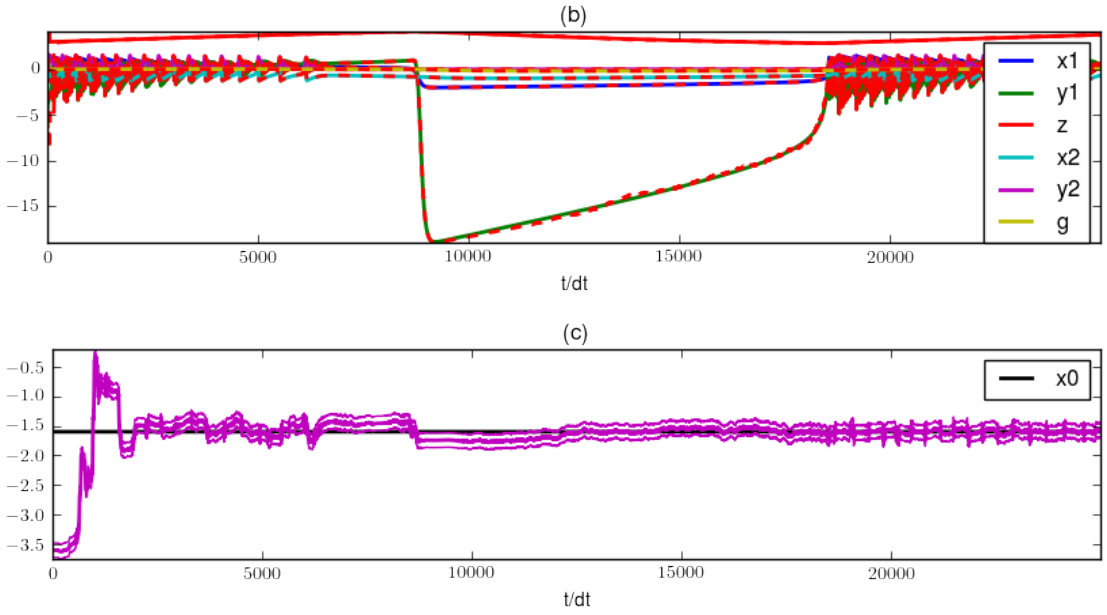

Figure 7: Filtering the Epileptor with one static parameter. Here the hidden state estimates and parameter estimates are separated with the bottom plot showing the parameter estimates.

Figure 7 shows the filter tracking a single parameter, x_0. The parameter is set to the same constant as in the simulation shown in Fig.6. The initial estimates of the state variables and parameters are within 3 units of the actual values except for x_1 and x_2, which are set such that $-x_1 + x_2 = Y_0$, where Y_0 is the first observation in the simulated data set. The filter estimate did not stabilize around the true value until more than 1000 time steps had passed.

3 2/19/17

- BluePyOpt installation and examples
- DEAP optimization

3.1 BluePyOpt

3.2 Installation

I cloned the BluePyOpt repository and got their examples running. I had to install NEURON to get the ephys module to work properly and ended up needing to use conda install to make the python support work. Also, NEURON changed some lines in .bash_profile (reset PYTHONPATH without including the previous value), so those had to be updated after the install. The simplecell and l5pc examples will run with Python 3, but the Graupner-Brunel STDP example won't - Python 2.7 worked.

3.3 Testing

In the BluePyOpt paper, Van Geit et al describe setting up the optimization for a model that does not require the `ephys` package [4]. Only the model and an evaluator need to be written in order to run this type of optimization. An evaluator is a class object that maps parameters to objectives. The basic layout for setting up the model evaluator is as follows:

1. Construct a dictionary of parameters. In the STDP example, this is done by defining a method that takes *params* (parameters to be optimized) as input and returning a dictionary of all parameters. The parameters to be optimized are initialized in the evaluator class constructor with the range of possible values. For initial testing, only the $x0$ parameter is set to be optimized while the rest are frozen. The full list of frozen parameters is shown in Listing 1. Notice that all noise values are set to zero for the evaluator. When generating the simulation, these will be nonzero, and it may make sense to change the ensemble noise values, but for simplicity they start at zero.

2. Set up the protocols to be optimized. In the STDP example, this is a series of signalling locations corresponding to different mean and standard error values according to the literature; here, the mean and standard errors are the objectives. In Epileptor, the simulation (or real data) will be the objective. For initial testing, the protocols shown in Listing 2 were used. These protocols set the parameters for the simulation that will be used as the target trace. The resulting targets are shown in Figure 8. Protocol 1 includes both process and observation noise, Protocol 2 only includes process noise, and Protocol 3 is fully deterministic. The black line is the underlying process and the blue points represent the possibly noisy observations that are the actual targets of the optimization.

3. Evaluate the optimization results against the protocol objectives. Finally, the evaluator quantifies the error in the results in order to minimize this error. For the Epileptor, this is done by computing the root mean square error between the two traces (Is there a better way to do this?).

The Epileptor model itself and all supporting functions are constructed separately and stored in a different file. Because this code was reused from the Kalman filter, there are functions which are not used and some of the information is redundant. Also, I don't really know what I'm doing as far as object oriented programming so it's sloppy. (Someday I can clean this up and optimize it if it seems worthwhile to do so.)

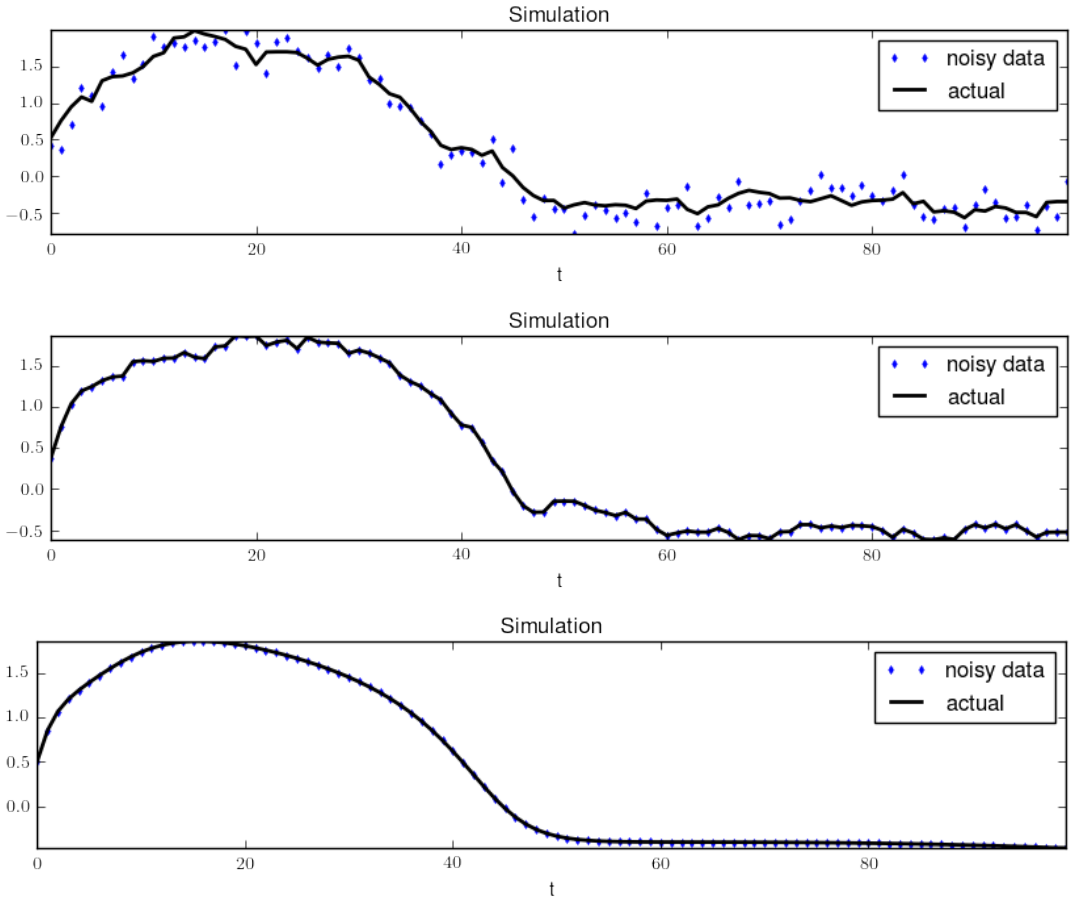

Figure 8: Initial testing of optimization of Epileptor model using BluePyOpt machinery. These figures show the 'target' traces for the protocols shown in 2. The black line is the underlying process and the blue points represent the possibly noisy observations that are the actual targets of the optimization.

Listing 1: Default Epileptor parameters.

```
ep_param = {
    'y0': 1.,
    'tau0': 2857.,
    'tau1': 1.0,
```

```
    'tau2': 10.,
    'Irest1': 3.1,
    'Irest2': 0.45,
    'gamma': 1e-2,
    'x1_init': 0.,
    'y1_init': -5.,
    'z_init': 3.,
    'x2_init': 0.,
    'y2_init': 0.,
    'g_init': 0.,
    'observation_sigmas': 0.,
    'noise_ensemble1': 0.,
    'noise_ensemble2': 0.}
```

Listing 2: Protocols for testing BluePyOpt optimization of Epileptor.

```
protocols = [Protocol(prot_id='default'),
                  Protocol(prot_id='clean', observation_sigmas=0.),
                  Protocol(prot_id='noiseless',
                           observation_sigmas=0.,
                           noise_ensemble1=0.,
                           noise_ensemble2=0.)]
target = [epileptor_model(params=protocols[0].params).
          generate_simulation(plot=plot),
          epileptor_model(params=protocols[1].params).
          generate_simulation(plot=plot),
          epileptor_model(params=protocols[2].params).
          generate_simulation(plot=plot)]
```

*** TODO: Describe results, get some EEG data and see if running it breaks everything... ***

3.4 DEAP optimization

4 03/04/17

Have been messing around with the BluePyOpt package using model simulations as input and downloaded some EEG data from ieeg.org - the code did not break and looks like it might be a good starting point to figure out which parameters should be optimized.

4.1 Looking at ieeg.org data

First, to download the data, I set up the command line interface (CLI) package and made an alias for the *ieeg* command so that it runs from anywhere. Data is set to download into *~/Kramer_rotation/ieeg_data*. A dataset can be downloaded using the command

```
ieeg download -c ch_06 I002_A0003_D010
```

where *I002_A0003_D010* can be substituted for the name of whatever dataset. Using the `-c` option lets you pick a channel. More instructions can be found at https://bitbucket.org/ieeg/ieeg/wiki/cli.

Once the data is downloaded, it has to be converted into an EDF file:

```
mef2edf ch_06.mef --output-file outputEdf.edf
```

The result is saved in a subfolder in the directory where the data is saved (*mef2edf* is also aliased to work from any directory). The package *pyedflib* will read these files and can be installed using *pip install pyedflib* for python 2.7 and beyond. Usage can be found in the package documentation online and is relatively straightforward:

```
f = EdfReader(
    '/Users/emilyschlafly/BU/Kramer_rotation/ieeg_data/' +
    'I002_A0003_D010/outputEdf_EDF/outputEdf_0.edf')
chan = 0
data = f.readSignal(chan)
sample_freq = f.getSampleFrequency(chan)
f._close()
```

DON'T FORGET TO CLOSE THE FILE. The portion of code shown above is from `epileptor_util -> load_protocols`.

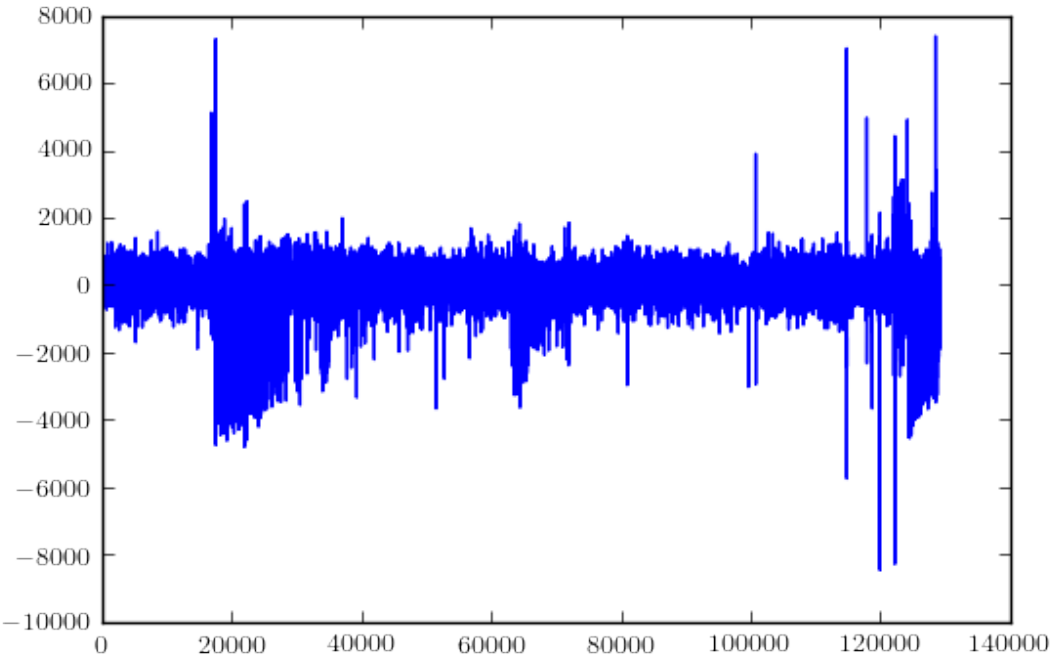

Figure 9: EEG data from ieeg.org, dataset I002_A0003_D010 ch_06.

This dataset is shown in 9. The set was chosen arbitrarily and I don't know whether or not this is representative or particularly interesting, but it looks like it has three SLEs, although the baseline shift is not obvious (note that in subsequent plots using this dataset the y-axis is scaled down by a factor of 1000). I did a first pass optimization using bpop with only 3 offspring and max 10 generations. The unfrozen parameters were $\tau 0$, which alters the rate of change of the permittivity variable, and the initial conditions for the y_1 and z variables. The results from the trial run are shown in Figure 10.

Just looking at these plots, which are the product of very few generations, it looks like it might be worth looking at the parameters that determine the extent of the baseline shift.

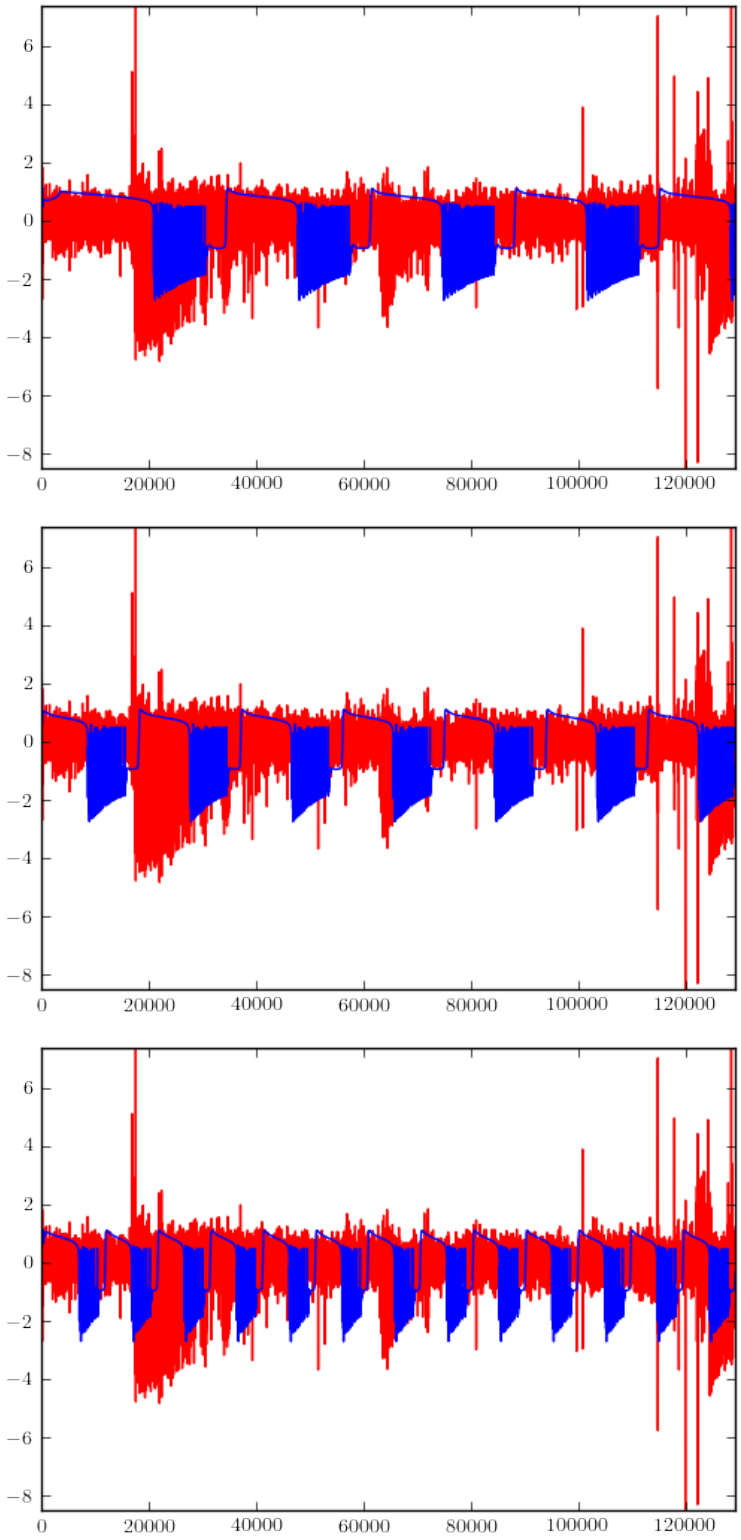

Figure 10: Test run of BluePyOpt. Only the parameters τ_0, $y_1(0)$, and $z(0)$ were unfrozen.

The other major difference between the observed data and the simulation is the fact that the SLEs occur at regular intervals in the simulations. Maybe consider windowing the data. Also, with the Kalman filter, we can watch τ_0 vary.

4.2 Next steps

- What parameters should be optimized
- What data should we be looking at
- Put real data into the Kalman filter with τ_0 as a parameter

References

[1] Viktor K Jirsa et al. "On the nature of seizure dynamics". In: *Brain* 137.Pt 8 (Aug. 2014), pp. 2210–30. DOI: 10.1093/brain/awu133.

[2] Peter E Kloeden and Eckhard Platen. *Numerical solution of stochastic differential equations*. 2nd corr. print. Vol. 23. Berlin: Springer, 1995. ISBN: 3540540628 (Berlin : acid-free paper).

[3] Steven J Schiff. *Neural control engineering: the emerging intersection between control theory and neuroscience*. Computational neuroscience series. Cambridge, MA: MIT Press, 2012. ISBN: 9780262015370 (hardcover : alk. paper).

[4] Werner Van Geit et al. "BluePyOpt: Leveraging Open Source Software and Cloud Infrastructure to Optimise Model Parameters in Neuroscience". In: *Front Neuroinform* 10 (2016), p. 17. DOI: 10.3389/fninf.2016.00017.

[5] H. U. Voss, J. Timmer, and J. Kurths. "Nonlinear dynamical system identification from uncertain and indirect measurements". In: *Int. J. Bifurcation Chaos* 14.6 (June 2004), pp. 1905–1933. ISSN: 0218-1274.