Annealing code `minAone` User Guide 2.0

Jingxin Ye and Nirag Kadakia

June 9, 2016

Contents

1 About `minAone`

The annealing code `minAone` described in this document is used for calculating action levels of the Gaussian error action for the estimation of dynamical systems. The code is developed as an extention of `minAzero` written by Bryan Toth and Chris Knowlton (hence the the uninspired name `minAone`). In another aspect, following the lowest action level A_0, A_1 represents the second lowest one, which is an interesting quantity we care about and has significant application in statistical data assimilation.

2 Problem Statement

Given a dynamical system modeled by D-dimensional discrete map

$$x_a(n+1) = f_a(\mathbf{x}(n)), \quad a = 1, \ldots, D$$

the probability distribution of its states can be expressed as $P(X|Y) = \int \mathcal{D}X \exp(-A_0[X])$, when L-dimensional obersations Y are present. If one assumes both measurement noises and model error are independent and gaussian, the action A_0 in discrete time has the format of

$$A_0(X) = \sum_{n=0}^{m} \frac{R_m(n)}{2} \sum_{l=1}^{L} [x_l(n) - y_l(n)]^2 + \frac{R_f}{2} \sum_{n=0}^{m-1} \sum_{a=1}^{D} [x_a(n+1) - f_a(\mathbf{x}(n))]^2. \tag{1}$$

where R_m and R_f are the inverse of variances.

The annealing method is based on the observation that the minima solution X^q of A_0 at $R_f = 0$ is $x_l(n) = y_l(n)$, the other $D - L$ components of the model state vector are undetermined, and the solution is degenerate. As we increase R_f, the action levels split, and depending on R_m, R_f, L and the precise form of the dynamical vector field $\mathbf{f}(\mathbf{x})$, there will be 1,2,... minima of A_0.

3 Annealing Procedure

The annealing process proceeds as follows: with very small initial R_f, we call it R_{f0}, solve the $(m+1)D$-dimensional search problem with an optimization algorithm that seeks minima of $A_0(X)$. Start the search with a set of trial paths whose components are selected from a uniform distribution within limits suggested by examining the times series generated by the model $\mathbf{x} \to \mathbf{f}(\mathbf{x})$ (or any other selection process for the initial guess). This will generate a collection of approximate paths X^q. Increase R_f by a small increment (we choose $R_f = \{R_{f0}\alpha^\beta\}$, where $\alpha = 2, \beta = 0, 1, \dots$ in our examples), and using the paths found for the smaller R_f as initial guesses, find a new set of approximate X^q. Continue this process until the lowest action level path X^0 produces a $A_0(X^0)$ near expected value, which can be identified from our knowledge of measurement noises. In our example, as the values $[y_l(n) - x_l(n)] \sim \mathcal{N}(0, \sigma^2)$ by our choice, the measurement error term $\sum_{n=0}^{m} \sum_{l=1}^{L} [(x_l(n) - y_l(n))/\sigma]^2/2$ has a χ^2 distribution with $L(m+1)$ degrees of freedom. The mean and uncertainty of this distribution over different choices of noise waveforms are $(m+1)L/2$ and $\sqrt{(m+1)L/2}$, respectively.

After identifying the global minima and other local minima of A_0, we can employ laplace method to approximate the expected value $\langle G(X) \rangle$ of a function $G(X)$ is

$$\langle G(X) \rangle = \frac{\int dX\, G(X) \exp[-A_0(X)]}{\int dX\ \exp[-A_0(X)]} \approx G(X^0). \tag{2}$$

plus exponentially small corrections. If the action level $A_0(X^0)$ is substantially less than the action level on the next path $A_0(X^0) \ll A_0(X^1)$, all statistical data assimilation expected values $\langle G(X) \rangle$ are given by X^0 and fluctuations about that path with exponential accuracy of order $\exp[-(A_0(X^1) - A_0(X^0))]$.

More details can be found in Ye, J., Kadakia, N., Rozdeba, P. J., Abarbanel, H. D. I., and Quinn, J. C.: Improved variational methods in statistical data assimilation, Nonlin. Processes Geophys., 22, 205-213, doi:10.5194/npg-22-205-2015, 2015

4 Installing Required Programs and Packages

This document will assume that the user is using a Linux distribution and has basic compliers installed including gcc, gfortran and python.

4.1 Python Packages

These python scripts link to the sympy library. To install these, use `apt-get/yum install sympy` or download directly from sympy.org.

4.2 IPOPT

Download

Get it here: `https://projects.coin-or.org/Ipopt`

- Download and unzip latest version of IPOPT
- As of right now this is 3.11.7 - Efficacy of installation instructions may degrade over time as packages are updated.
- Go into ThirdParty folder in the IPOPT directory then do the following commands.

```
$ cd Blas
$ ./get.Blas
$ cd ../Lapack
$ ./get.Lapack
$ cd ../ASL
$ ./get.ASL
$ cd ../Metis
$ ./get.Metis
```

- Get the HSL subroutines from `http://hsl.rl.ac.uk/ipopt`
- Note that there are two releases for HSL - you will want the more complete one that contains ma57, ma77, and ma97.
- While the freely available ma27 will work for many problems, the newer packages are faster, work on larger problems, and can use multi-core architecture.
- This will require filling out a form stating essentially that you are in academia and waiting a couple hours for a link to download.
- Unpack the resulting library into the ThirdParty folder such that the path is (IPOPT Path)/ThirdParty/HSL/coinhsl

Install

- Go to the IPOPT directory

```
$ mkdir build
$ cd build
$ ../configure
```

- Note that if you have lapack or blas installed previously you can use –with-lapack and –with-blas to link to those packages

- If something goes wrong refer here `http://www.coin-or.org/Ipopt/documentation/node19.html#ExpertInstall`

- Assuming everything worked:

```
$ make
$ make test
$ make install
```

5 minAone.py Description

minAone is a python script used to write C++ code and compiler instructions using the IPOPT (Interior Point OPTimization) libraries to estimate unmeasured states and parameters in dynamical systems with limited measurements. The scripts take a set of differential equations and state and parameter names provided by a text file ”equations.txt” and returns a set of C++ files consisting of a set of constraints based on a discretized version of those differential equations. A second text file ’specs.txt’ allows for changes in run specific quantities state and parameter bounds, as well as input files without the need to recompile.

List of Files

- discAone.py
  -Discretizes equations and creates strings for Jacobian and Hessian Elements.

- makecppAone.py
  -Writes C++ file linking to IPOPT libraries using strings from discAone.py

- makehppAone.py
  -Writes header file for above

- makemakeAone.py
  -Writes makefile for problem. Will need to be changed based on install location of IPOPT

- makeoptAone.py
  -Writes settings file for IPOPT

These files can be put in /usr/local/sbin for ease of use.

Modify makemakeAone.py

The Makefile compiles C ++ object files and links them with the installed IPOPT libraries, in order to create an executable. Since the location of the IPOPT libraries, as well as the flags used to compile them, differ between installations, this file will be unique to a given machine. Modification of the makemake.py script to give correct Makefiles for a given machine consists of:

- Ensure that the IPOPT installation proceeded correctly, as evidenced by zero errors for the make install step.
- In the IPOPT build directory, try to compile (make) one of the examples, for instance at /build/Ipopt/examples/hs071 cpp.
- If this compiles and runs correctly, open the Makefile in this directory.
- Make note of the entries in the following fields of this Makefile: CXX, CXXFLAGS, CXXLINK- FLAGS, INCL, LIBS.
- In makemake.py, replace the default entries for these fields with those given in the example Makefile.
  - makemakeAone.py is formatted differently than a Makefile, since it is a python code generation script.
  - Lines that begin with the # sign will be comments in the Makefile - leave these alone.
  - All lines must end with \n\in order for the Makefile to be generated correctly.
  - The best way to ensure that all the compile flags are correct is to copy and paste from the example Makefile, ensuring that the end line characters are in place.
- The modification of makemake.py must only be done once for a given machine, unless IPOPT is reinstalled for whatever reason.

6 Running the Code

minAone uses as input a) any needed data files containing measurement or stimulus data, and b) two additional documents, `equations.txt` and `specs.txt`.

equations.txt contains information on the model and is used once for generating the needed cpp and hpp files for the run. The file should be written as described below in this order.

- The first line is the problem name, this name will be used to name the resulting executable.
- The second line tells minAzero how many dynamical variables, parameters, coupling terms, stimuli, functions, and measurements there are, in that order as a comma delimited list.
- A list of every differential equation.
- The measurement term of the cost function. A penalty term for coupling terms is suggested as any coupling to measurements is not present in physical systems.
- The names of all the variables. These must be the same as used in the differential equations and should be multiple letters/and or numbers such that variable name is contained in any other name or common function.
- The names of parameters, names of control variables (these are time-dependent variables over which the search is performed but for which no explicit dynamical equations are given; one may also think of them as time-dependent parameters), names of data, and names of stimuli, in that order. Again use fully unique names.
- Function names and number of arguments of that function separated by a comma. Use a function if there is some component of the dynamics with a removable singularity or other difficult numerical object that requires an alternative local definition.
- Functions will require an additional file 'myfunctions.cpp' containing the function definition along with its first and second derivatives. These allow for piecewise functions, removable singularities, etc. An example of this is given.

specs.txt contains run specific information such as file names, variable bounds, and problem length. This file can be edited without recompiling the code. Depending on the format of the input files (observations, initial conditions and stimuli), and the desired format of the output, `specs.txt` fies vary in their length. Here we specify **in detail** the required information, in the order it is to appear in `specs.txt`.

In the following, nY refers to the number of state variables, nM to the number of measured variables, nI to the number of stimuli, nU to the number of controls and nP to the number of parameters.

- **Line 1** number of discrete timesteps isteps the code will use. *Importantly*, because the code is compiled using a midpoint method, the actual problem length will double this plus one. If your data file is N time points in length, this value should be $N/2$.

- **Line 2** number of lines in each input file to skip. This allows for the code to start at any point in a long data set. Note that this same amount of data will be skipped in initial condition and stimulus files as well.

- **Line 3** double the time step of the data. Again, since a midpoint method is used, this time step is for a whole step - which includes two points. If your data is sampled at $dt = 0.05$, this value should be 0.1.

- **Line 4 and following several lines** Line 4 is extremely important; it indicates the format of the input files. The only accepted values are either 0, 1, 2, or 3.

→ **0**: You do not have an initial guess for estimated trajectory; it will be chosen uniformly at random from the dynamical range of the state variables and is seeded by the task ID for repeatability. Your observation data is saved in individual files, each of a single column indicating the value of the observation at successive times spaced by dt. Stimulus files are also saved invidually in this same format. After line 4, you will have nM lines for the file path of each observation, and nI lines for that of each stimulus. For example, you have 10 state variables, of which 2 are measured, and 1 stimulus file. Lines 4-7 may read:

```
0
./measuredVoltage.dat
./measuredCa.dat
./injectedCurrent.dat
```

→ **1**: Same as **0**‘, except you **do** have an initial guess for the estimated trajectory. This guess is saved in a file of nY columns and N rows, plus nP values at the end of the file for the parameters. Equivalently, the data can be saved as a single column, keeping in mind that the inner loop is over nY while the outer loop is time, with the parameter guesses added as the last nP rows. Also, keep in mind that if data is being skipped (second line in specs is not zero), then the same number of rows will be skipped from this file.

The initial guess file is written in Line 5 of `specs.txt`. The following nM + nI lines are for the observation and stimulus data file paths, as in the previous case. For example:

```
1
./initialization.dat
./measuredVoltage.dat
./measuredCa.dat
./injectedCurrent.dat
```

→ **2** You do not have an initial guess for estimated trajectory. Further, you have **many** datasets in appropriately indexed files. For example, you have 25 different sets of observed data, labeled `observations0.txt`, `observations1.txt`, ..., `observations24.txt`; each of these files has the data of *all* measured variables (in the previous cases, each column was a separate file). In addition, each data set was generated using a distinct stimulus, `stimulus0.txt`, ..., `stimulus24.txt`. Finally, you want to run 1000 estimations for *each* data set. This tag allows you to run all of this from a single `specs.txt` file.

Note that your observation files must have as many columns as variables, nY, *not* measurements, nM. Thus, if only nM = 2 out of nY = 5 variables are measured, the first 2 columns must be these measured data, while the last 3 columns must be composed of dummy data. For the stimulus files, the number of columns is nI, as expected.

With these datasets, specs is written in the following way. Line 5 is the number of desired runs per dataset. Line 6 is the file extension for your data. Line 7 is the prefix of the data file path, while Line 8 is the prefix for the stimulus file path. Thus, in the example given above, Lines 4-8 are the following:

```
2
```

```
1000
txt
./observations
./stimulus
```

The choice of data set and initial guess are made by taking the task ID modulo Line 5. Thus, in this example task IDs 0 - 999 will produce 1000 estimations using `observations0.txt` and `stimulus0.txt`, while task IDs 1000-1999 will produce 1000 estimations using `observations1.txt` and `stimulus1.txt`, using the *same* initial conditions as the first set. That is, task ID 525 and 1525 are initialized with the exact same guess since they are both seeded with 525 (1525 mod 1000 = 525 mod 1000 = 525).

Finally, there is one special case: when Line 5 equals 0. In this case, all observations and stimuli are still in a single data file, but now you do not have multiple indexed datasets. For example, you have a single data file of `twindata.dat` and stimulus file `current.dat`. Then Lines 4-8 should read:

```
2
0
dat
./twindata
./current
```

In this case, there is only 1 dataset, so the task ID will correspond directly to the initializating guess.

→ **3** This last case is identical to **2** except that an initial data file is provided, and must be indexed. Using the previous example, we may have 25 data sets, of which we intend to do 1000 estimations each, but now we want to initialize each of these estimations manually. Say the initializing files are initdata0.dat, initdata1.dat, ..., initdata999.dat. Then our specs file would be similar to the previous case, with an extra line put after Line 6 which gives the initializations:

```
3
1000
txt
```

```
./initdata
./observations
./stimulus
```

As in the previous case, if Line 5 is set to 0, you only need 1 initial data file, unindexed, say initialdata.txt:

```
3
0
txt
./initialdata
./twindata
./current
```

- **The next nY lines** For each variable, list the lower bound, upper bound, and RF0 value separated by commas.
- **The next nU lines** For each control, a lower bound, upper bound, and initialializing value for the control, separated by commas.
- **The next nP lines** For each parameter, a lower bound and upper bound, separated by commas.
- **Next line** The annealing settings: alpha, increment of beta, and maximum beta, separated by commas
- **Next line, optional** A tag indicating how the data will be saved; values can be 0, 1, -1, 2, or -2. If Line 4 equals 0 or 1, the data is saved to a file like `D5_M3_IC2.dat`, where the number after "D" is nY, that after "M" is nM, and that after "IC" is the task ID. If Line 4 equals 2 or 3, the data is saved to a file like `D5_M3_PATH15_IC23.dat` where the number after "PATH" is the relevant data set (0 to 24 in the example above), while the number after "IC" is the initialization, found by task ID modulo Line 5 (If Line 5 is 0, the file is saved in the format `D5_M5_IC2.dat`).

→ **0**: This is the default value if this line is omitted. Data for each beta value is saved by row. The first 3 columns are beta, exitflag, and action value. Exit flags can be found in the IPOPT documentation. The remaining values in each row are the estimated trajectory values, (inner loop over the state variables, outer loop over time), with the parameter estimates appended at the end.

→ **-1**: Same as (0), but only the estimated trajectory for the *final* beta value is saved; the file has only one row. This may help save space when running a large batch of runs, in which one is only interested in the final trajectory.

→ **1**: Same as (-1), except the first three values (beta, exitflag, and action value) are dropped. This format is in the correct format if one wants to use this file as an initial data file for a subsequent estimation (Note: it does not account for skipped data, however).

→ **2**: The format is same as (0), but the final values of the control variables, if used, are also saved to file (the inner loop is over nY + nU).

→ **-2**: Same as (2), but only data for the final beta value is saved.

Once everything is filled out and all data files are present, you can run the python scripts:

```
$ minAone.py
$ make
$ ./(problem_name)_cpp taskID
```

where taskID is a nonnegative integer specifying the task ID. If data files are missing or too short, or if specs is filled out incorrectly, the code will segfault.

7 Run in Parallel

One execution of `(problem_name)_cpp` can obtain the result for only one random initial path. To explore the landscape of action A_0, we need to start from different random paths and each of them will converge to different local minima. Since all those paths are indenpendent from each other, it is easy to implement the calculation in parallel using array job.

Here we give a example submission scripts on ccom-boom cluster

```
#!/bin/bash
#$ -t 1-100
#$ -N job_name
#$ -cwd
#$ -j y
#$ -M your@email.com
#$ -S /bin/bash
```

```
#$ -m beas
#$ -o ./output
#$ -e ./error
#$ -q batch.q
./problem_name_cpp $SGE_TASK_ID
```

Each path will be stored in individual file with the name like `D5_M1_IC0.dat`, `D5_M1_IC1.dat`, ..., etc., if using a single dataset, or `D5_M1_PATH0_IC0.dat`, `D5_M1_PATH0_IC1.dat`, ..., `D5_M1_PATH0_IC100.dat`, ..., `D5_M1_PATH50_IC0.dat`, ..., `D5_M1_PATH50_IC100.dat` if using multiple datasets.

8 Examples

Four examples are provided:

1. Lorenz96 D=10 to show the basic settings of `equations.txt` and `specs.txt`.
2. Lorenz96 D=10 with many sets of observations to show how to incorporate multiple datasets (Line 4 equals 2 or 3).
3. Lorenz96 D=10 with synchronizing control terms.
4. NaKL neuron model to show how to include external stimuli in `equations.txt` and `specs.txt`.
5. Multi-compartment neuron model to show how to include externally-defined functions in `equations.txt`.

8.1 Lorenz96 D=10 (single data set)

Vector field

$$\frac{dx_1}{dt} = x_{10}(x_2 - x_9) - x_1 + f \qquad \frac{dx_2}{dt} = x_1(x_3 - x_{10}) - x_2 + f$$
$$\frac{dx_3}{dt} = x_2(x_4 - x_1) - x_3 + f \qquad \frac{dx_4}{dt} = x_3(x_5 - x_2) - x_4 + f$$
$$\frac{dx_5}{dt} = x_4(x_6 - x_3) - x_5 + f \qquad \frac{dx_6}{dt} = x_5(x_7 - x_4) - x_6 + f$$
$$\frac{dx_7}{dt} = x_6(x_8 - x_5) - x_7 + f \qquad \frac{dx_8}{dt} = x_7(x_9 - x_6) - x_8 + f$$
$$\frac{dx_9}{dt} = x_8(x_{10} - x_7) - x_9 + f \qquad \frac{dx_{10}}{dt} = x_9(x_1 - x_8) - x_{10} + f$$

equations.txt

```
# Problem Name
lorenz96
# nY,nP,nU,nI,nF,nM
10,1,0,0,0,4
# Dynamical equations
yy9*(yy1-yy8)-yy0+FF1
yy0*(yy2-yy9)-yy1+FF1
yy1*(yy3-yy0)-yy2+FF1
yy2*(yy4-yy1)-yy3+FF1
yy3*(yy5-yy2)-yy4+FF1
yy4*(yy6-yy3)-yy5+FF1
yy5*(yy7-yy4)-yy6+FF1
yy6*(yy8-yy5)-yy7+FF1
yy7*(yy9-yy6)-yy8+FF1
yy8*(yy0-yy7)-yy9+FF1
# Measurement terms of cost function
(data0-yy0)*(data0-yy0) + (data1-yy1)*(data1-yy1) + (data2-yy2)*(data2-yy2)
+ (data3-yy3)*(data3-yy3)
# Variable names
yy0
yy1
yy2
yy3
yy4
yy5
yy6
yy7
yy8
yy9
# Control names (none)
# Parameter names
FF1
# Data names
data0
data1
data2
data3
# Stimuli names (none)
```

```
# External functions (none)
```

specs.txt

```
# N/2 (total data is 161 steps, at dt = 0.01)
80
# Skipped data
100
# Twice the timestep of the data file
0.02
# Input format of files (0: only 1 input file, no initial condition)
0
# Measured data file paths (one for each measurement)
./observations/data0.dat
./observations/data1.dat
./observations/data2.dat
./observations/data3.dat
# Stimuli data file paths (none)
# State variable bounds and Rf0 values
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
# Control variable bounds (none)
# Parameter bounds (last line is correct value; not read)
0, 20, 8.17
# Alpha, min beta, max beta
2,1,30
```

sub.sh

```
#!/bin/bash
#$ -t 1-100
#$ -N lorenz96_cpp
```

```
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -m beas
#$ -o ./output
#$ -e ./error
#$ -q batch.q
./lorenz96_cpp $SGE_TASK_ID
```

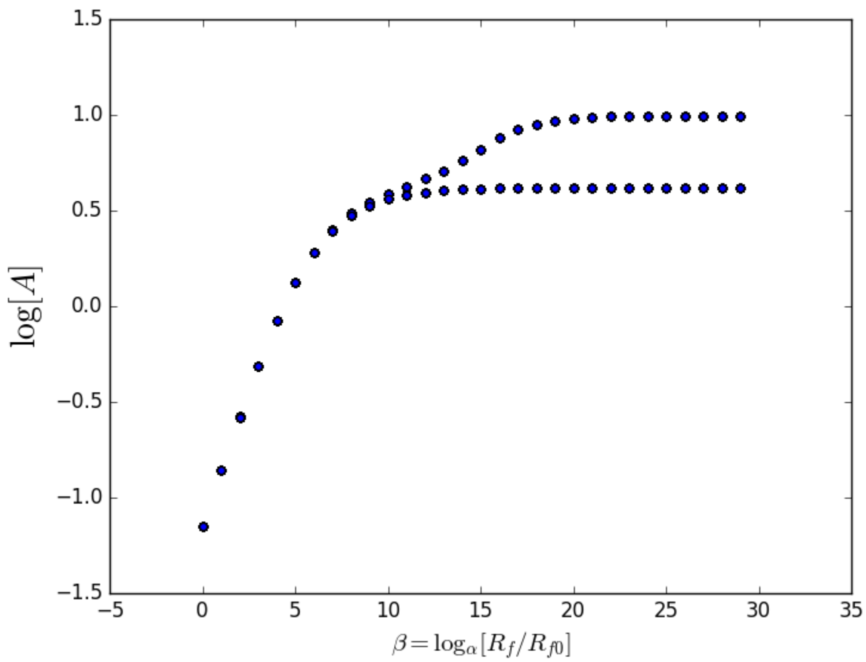

Figure 1: Lorenz 96 D=10 L=4 action plot for all 100 initial guesses.

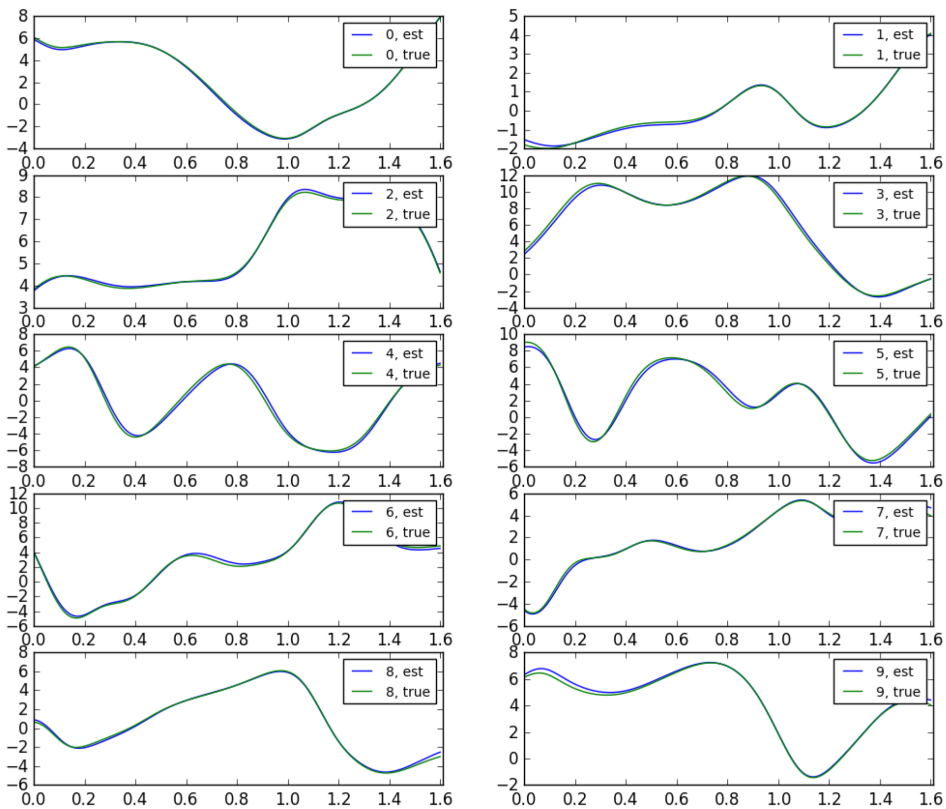

Figure 2: Lorenz 96 D=10 L=4 estimate for lowest action level.

8.2 Lorenz96 D=10 (mutiple data sets)

equations.txt is the same as in the previous section.

specs.txt

```
# N/2 (total data is 161 steps, at dt = 0.01)
80
# Skipped data
100
# Twice the timestep of the data file
0.02
# Input format of files (2: several datasets, no initial condition)
2
# 100 initial conditions for each data set
100
# data files have extension .txt
txt
./observations/observations_
# Stimuli data file paths (none)
# State variable bounds and Rf0 values
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
-15, 15, 0.01
# Control variable bounds (none)
# Parameter bounds (last line is correct value; not read)
0, 20, 8.17
# Alpha, min beta, max beta
2,1,30
```

sub.sh

```
#!/bin/bash
#$ -t 100-599
```

```
#$ -N lorenz96_cpp
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -m beas
#$ -o ./output
#$ -e ./error
#$ -q batch.q
./lorenz96_cpp $SGE_TASK_ID
```

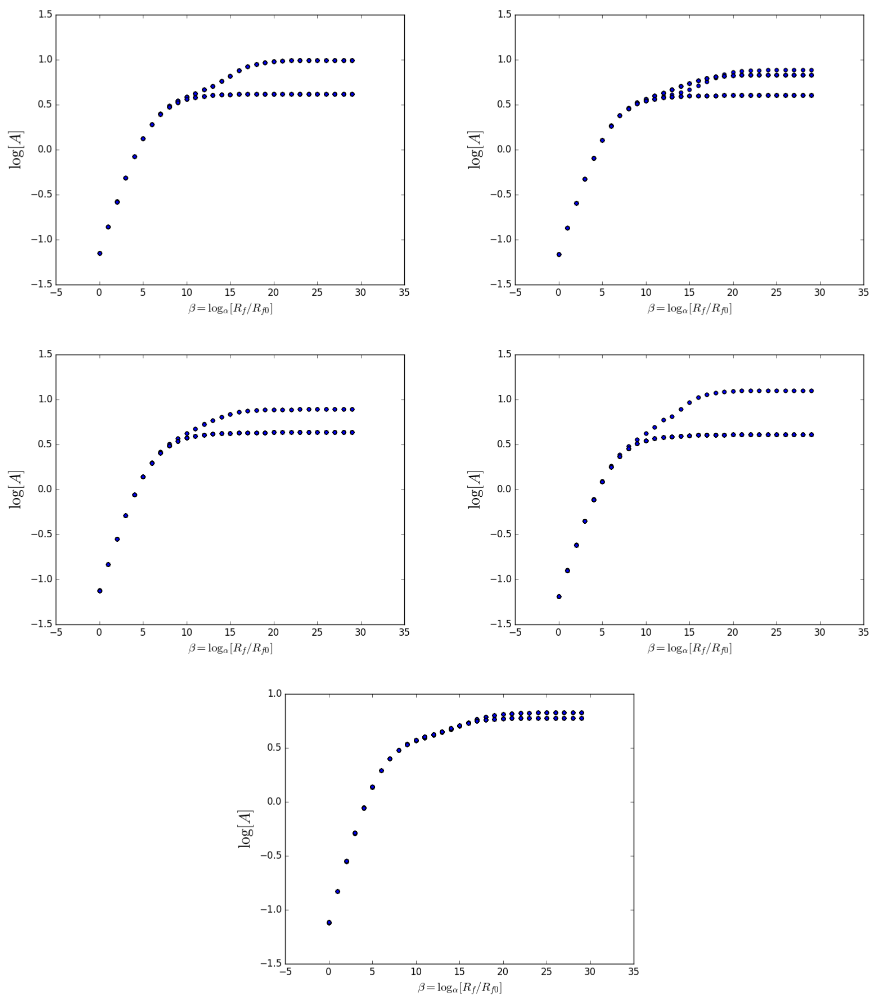

Figure 3: Lorenz 96 D=10 L=4 action plots for the 5 data sets.

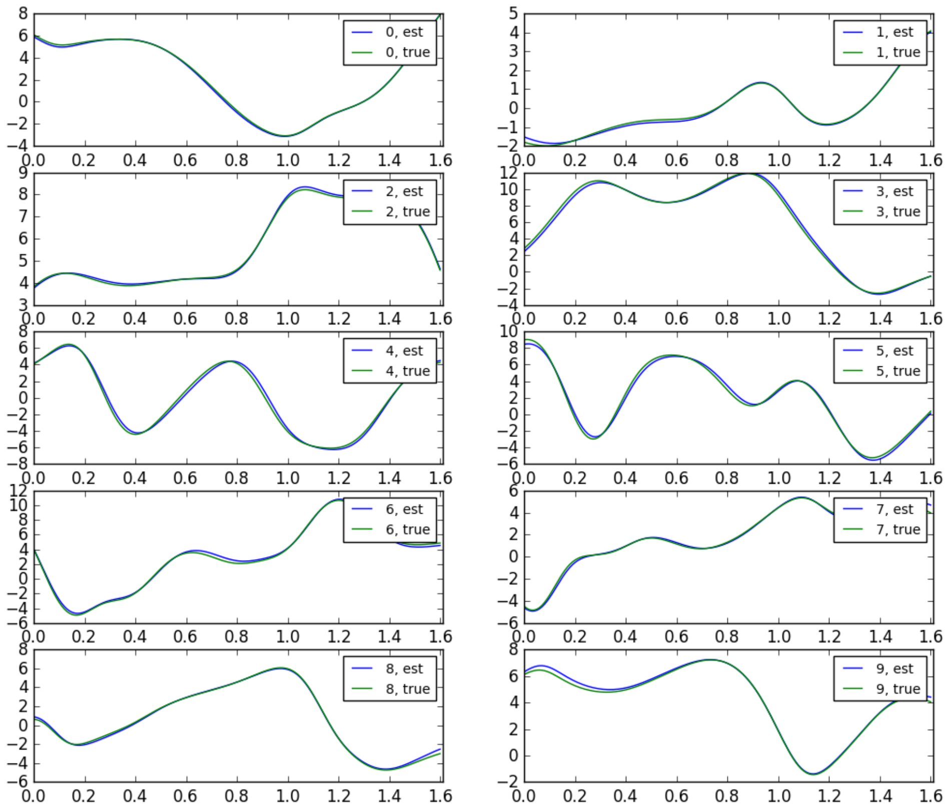

Figure 4: Lorenz 96 D=10 L=4 estimate for lowest action levels for data set 1. Data set 1 is the same used in the previous example, and one sees that both the action plot and best estimate are identical.

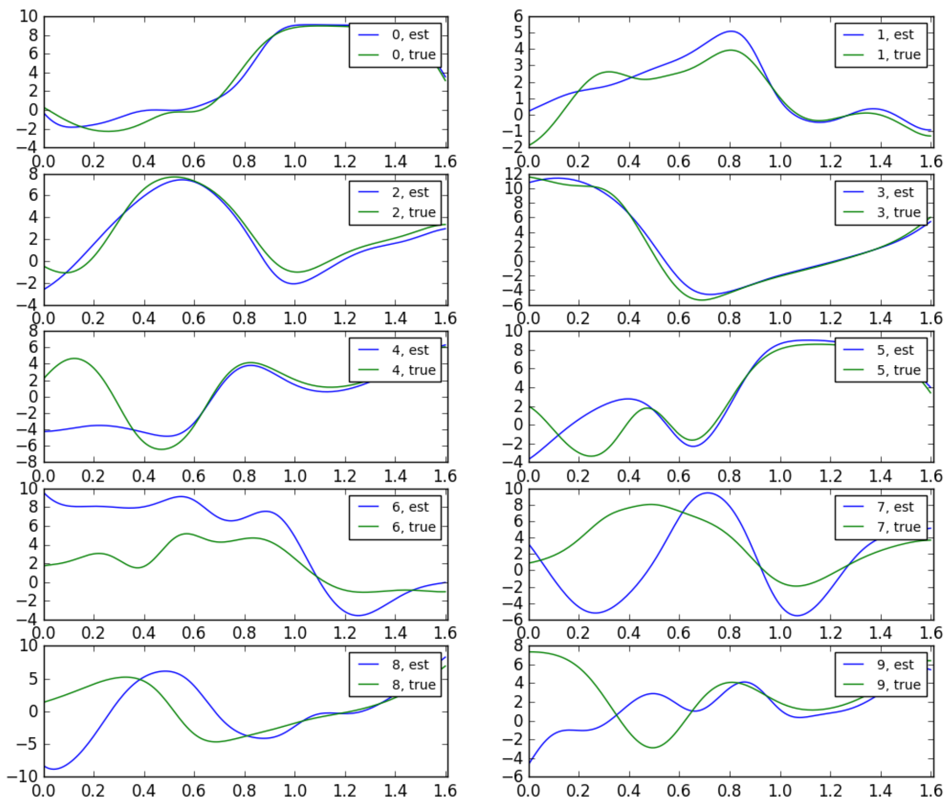

Figure 5: Lorenz 96 D=10 L=4 estimate for lowest action levels for data set 5. Here the lowest action level gives a poorer estimate. This is consistent with the action plots (Figure 3, where it is seen that the lowest action level for the 5th data set is not quite near the expected value of about 4.

8.3 Lorenz96 D=10 (synchronizing control terms)

Vector field

$$\frac{dx_1}{dt} = x_{10}(x_2 - x_9) - x_1 + f + u_1(y_1 - x_1) \qquad \frac{dx_2}{dt} = x_1(x_3 - x_{10}) - x_2 + f + u_2(y_2 - x_2)$$
$$\frac{dx_3}{dt} = x_2(x_4 - x_1) - x_3 + f + u_3(y_3 - x_3) \qquad \frac{dx_4}{dt} = x_3(x_5 - x_2) - x_4 + f + u_4(y_4 - x_4)$$
$$\frac{dx_5}{dt} = x_4(x_6 - x_3) - x_5 + f \qquad \frac{dx_6}{dt} = x_5(x_7 - x_4) - x_6 + f$$
$$\frac{dx_7}{dt} = x_6(x_8 - x_5) - x_7 + f \qquad \frac{dx_8}{dt} = x_7(x_9 - x_6) - x_8 + f$$
$$\frac{dx_9}{dt} = x_8(x_{10} - x_7) - x_9 + f \qquad \frac{dx_{10}}{dt} = x_9(x_1 - x_8) - x_{10} + f$$

equations.txt

```
# Problem Name
lorenz96
# nY,nP,nU,nI,nF,nM
10,1,3,0,0,3
# Dynamical equations (including synchronization terms)
yy9*(yy1-yy8)-yy0+FF1 + u1*(data0-yy0)
yy0*(yy2-yy9)-yy1+FF1 + u2*(data1-yy1)
yy1*(yy3-yy0)-yy2+FF1 + u3*(data2-yy2)
yy2*(yy4-yy1)-yy3+FF1
yy3*(yy5-yy2)-yy4+FF1
yy4*(yy6-yy3)-yy5+FF1
yy5*(yy7-yy4)-yy6+FF1
yy6*(yy8-yy5)-yy7+FF1
yy7*(yy9-yy6)-yy8+FF1
yy8*(yy0-yy7)-yy9+FF1
# Measurement terms of cost function
(data0-yy0)*(data0-yy0) + (data1-yy1)*(data1-yy1) + (data2-yy2)*(data2-yy2)
+ u1*u1 +u2*u2 + u3*u3
# Variable names
yy0
yy1
yy2
yy3
yy4
```

```
yy5
yy6
yy7
yy8
yy9
# Control names
u1
u2
u3
# Parameter names
FF1
# Data names
data0
data1
data2
# Stimuli names (none)
# External functions (none)
```

specs.txt

```
# N/2 (Using 401 data points from data files)
200
# Skipped data
100
# Twice the timestep of the data file
0.02
# Input format of files (2: data in single file)
2
# (0: there is only 1 dataset, but all measurements in 1 file)
0
txt
observations/observations_1
# Stimuli data file paths (none)
# State variable bounds and Rf0 values (Rf0 high to enforce constraints strictly)
-15, 15, 10^8
-15, 15, 10^8
-15, 15, 10^8
-15, 15, 10^8
-15, 15, 10^8
-15, 15, 10^8
```

```
-15, 15, 10^8
-15, 15, 10^8
-15, 15, 10^8
-15, 15, 10^8
# Control variable bounds and initial values
-1,1,1
-1,1,1
-1,1,1
# Parameter bounds (last line is correct value; not read)
0, 20, 8.17
# Alpha, min beta, max beta (only 1 step optimization)
2,1,1
# Output format (2: save control variables to file)
-2
```

sub.sh

```
#!/bin/bash
#$ -t 1-100
#$ -N lorenz96_cpp
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -m beas
#$ -o ./output
#$ -e ./error
#$ -q batch.q
./lorenz96_cpp $SGE_TASK_ID
```

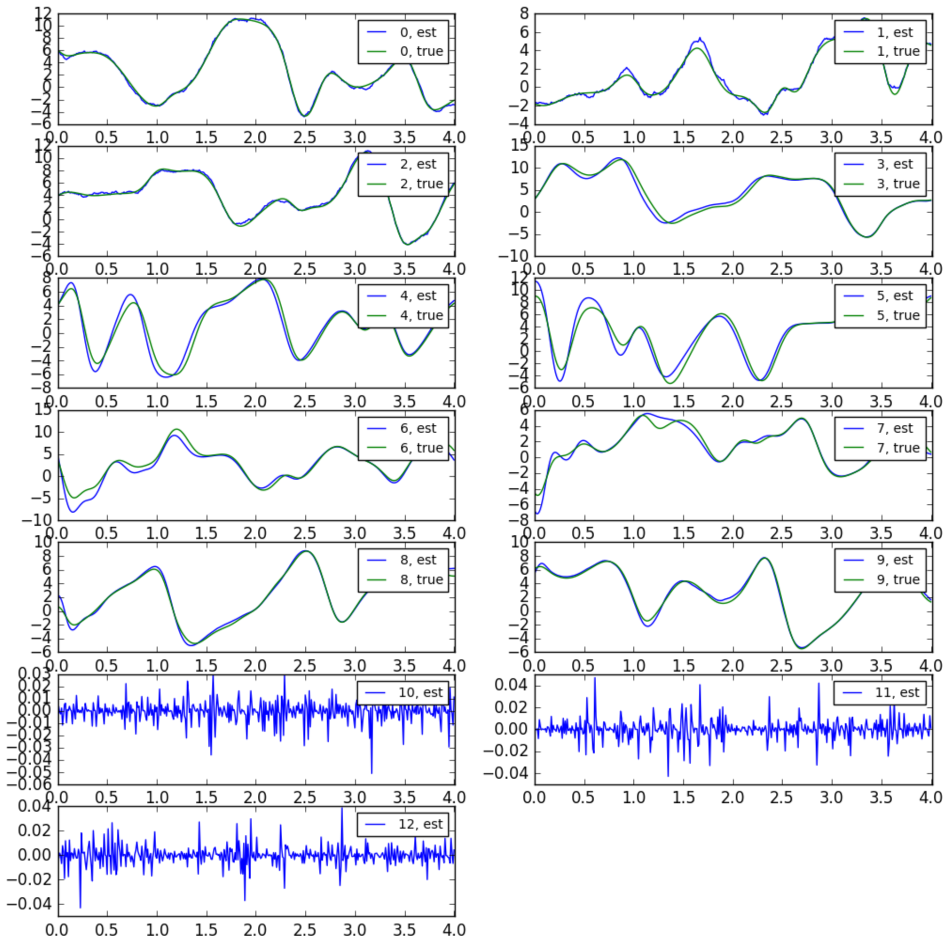

Figure 6: Lorenz 96 D=10 L=3 estimate for lowest action level. The last three variables are the control terms. The forcing parameter FF1 was estimated to be about 8.149 (true value 8.17).

8.4 NaKL

Vector field

$$\frac{dV}{dt} = CI_{inj}(t) + g_{Na}m^3h(E_{Na} - V) + g_K n^4(E_K - V) + g_L(E_L - V)$$
$$\frac{da}{dt} = \frac{a_\infty - a}{\tau_a}, \quad a = \{m, h, n\}$$
$$a_\infty = \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{V - V_a}{\Delta V_a}\right)$$
$$\tau_a = \tau_{a0} + \tau_{a0}\left(1 - \tanh^2\left(\frac{V - V_a}{\Delta V_a}\right)\right)$$

equations.txt

```
simple_nakl
4,19,0,1,0,1
# Dynamical equations (dV/dt, dn/dt, dh/dt, dm/dt)
gNa*(m0*m0*m0*h0)*(ENa-V0)+gK*n0*n0*n0*n0*(EK-V0)+gL*(EL-V0)+Area*Iinj
(0.5*(1+tanh((V0-Vmo)*dVm)) - m0)/(Cm1+Cm2*(1.0-tanh((V0-Vmo)*dVm)
*tanh((V0-Vmo)*dVm)))
(0.5*(1+tanh((V0-Vho)*dVh)) - h0)/(Ch1+Ch2*(1.0-tanh((V0-Vho)*dVh)
*tanh((V0-Vho)*dVh)))
(0.5*(1+tanh((V0-Vno)*dVn)) - n0)/(Cn1+Cn2*(1.0-tanh((V0-Vno)*dVn)
*tanh((V0-Vno)*dVn)))
# Measurement term of objective function
(VDATA0 - V0)*(VDATA0 - V0)
# State variable names
V0
m0
h0
n0
# Control variable names (none)
# Parameter names
gNa
ENa
gK
EK
gL
EL
Area
```

```
Vmo
dVm
Cm1
Cm2
Vho
dVh
Ch1
Ch2
Vno
dVn
Cn1
Cn2
# Data names
VDATA0
# External stimuli names
Iinj
# Externally defined functions (none)
```

specs.txt

```
# N/2 (total data is 6001 steps, at dt = 0.01)
3000
# Skipped data (none)
0
# Twice the timestep of the data file (data taken at 50 kHz)
0.04
# Input format of files (1: single data set, initial condition file)
1
# Initial data file
./input_data/initial_guess.dat
# Measured data file paths (one for each measurement)
./input_data/noise_measured.dat
./input_data/current.dat
# State variable bounds and Rf0 values
-150,70,1e-3
0, 1,1e1
0, 1,1e1
0, 1,1e1
# Control variable bounds (none)
# Parameter bounds (plus true values and names, which aren't read)
```

```
50,200,120, gNa
0,100,50, ENa
5,40,20, gK
-100,-50,-77, EK
0.1,1,.3, gL
-60,-50,-54, EL
0.5,1.5,0.8, Area
-60,-30,-40, Vmo
.01,0.1,0.06667, dVm
0.05,.25,.1, Cm1
.1,1,.4, Cm2
-70,-40,-60, Vho
-0.1,-.01,-.06667, dVh
.1,5,1, Ch1
1,15,7, Ch2
-70,-40,-55, Vno
.01,0.1,.03333, dVn
.1,5,1, Cn1
2,12,5, Cn2
# Anneal settings
2,1,30
```

sub.sh

```
#!/bin/bash
#$ -t 1-100
#$ -N simple_nakl_cpp
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -m beas
#$ -o ./output
#$ -e ./error
#$ -q batch.q
./simple_nakl_cpp $SGE_TASK_ID
```

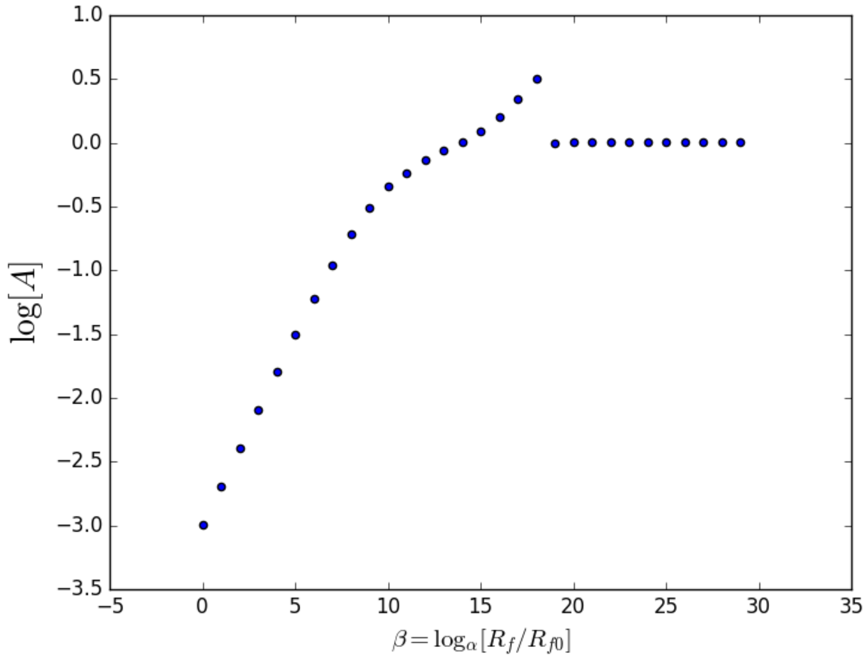

Figure 7: NaKL model action plot.

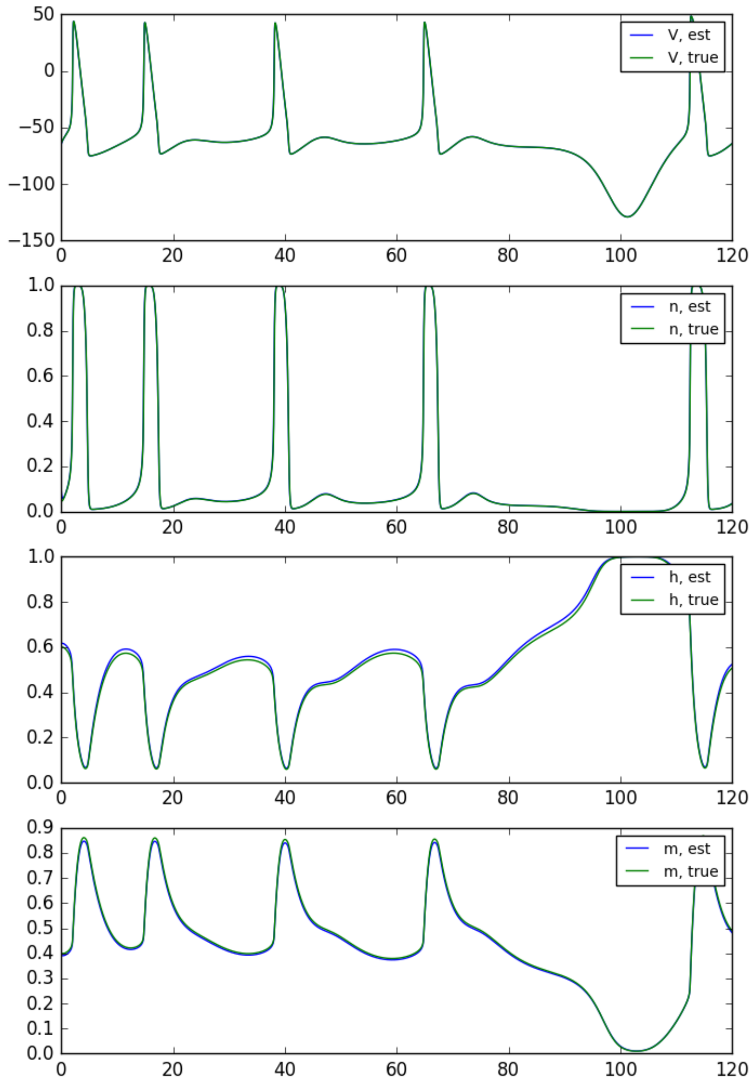

Figure 8: NaKL model best estimate, compared against the true trajectories.

9 Troubleshooting

I have tested these scripts over a wide range of problems, so I believe that the algorithms are correct. However, there are a few common errors that may crop up.

- Variable and parameter naming is very important. Never use a variable name that includes the name of another variable. For instance p1 and p11 would be bad, since p11 includes p1. In this case, p01 and p11 would be adequate. Along this vein, all variable names should be at least 2 characters long, just in case.