Guide to minAzero.py

Variational Method on Path Integral

Chris Knowlton

September 27, 2013

Path Integral Method Derivation

Many systems of interest can be modeled as a dynamical system for which information is available about only some subset of the state variables. Developing a method to estimate the optimum states and parameter values for these models conditioned on those measurements is essential for characterizing and predicting the future behavior of these systems. Unlike other common methods such as particle filtering or kalman filters that use the state vector at each time to predict the state vector at subsequent times, the path integral method uses variations in all the states at all times to arrive at estimates of unmeasured quantities of the system. As the entire trajectory is used, the method has some built in robustness against noisy data and chaotic behavior.

Consider a dynamical system $\mathbf{f}$, that describes the evolution of a d dimensional state vector $\mathbf{x}$:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{p}) \tag{1}$$

These dynamics lead to a Fokker-Planck equation relating the density function P($\mathbf{x}$(t)) at time N as a function of the state at time 0.

$$P(\mathbf{x}(N)) = \int d\mathbf{x}(i) K(\mathbf{x}(N); \mathbf{x}(0)) P(\mathbf{x}(0)) \tag{2}$$

where

$$K(\mathbf{x}(N); \mathbf{x}(0)) = \int \prod_{i=0}^{N-1} d\mathbf{x}(i) K(\mathbf{x}(i+1); \mathbf{x}(i)) \tag{3}$$

Working from the form in statistical mechanics and quantum mechanics, we rewrite the final state density as the function of the action, A_0, over the whole path - $\mathbf{X}$.

$$P(\mathbf{x}(N)) = \int \prod_{i=0}^{N-1} d\mathbf{x}(i) \exp[-A_0(\mathbf{X})] \tag{4}$$

Integrating over all possible states gives a means for calculating the expectation value of various quantities.

$$E[G(\mathbf{X})] =< G(\mathbf{X}) >= \frac{\int d\mathbf{X} G(\mathbf{X}) \exp[-A_0(\mathbf{X})]}{\int d\mathbf{X} \exp[-A_0(\mathbf{X})]} \tag{5}$$

To actually find A_0, we return to the probability distribution of the final state - which is conditioned on all the measurements that preceded it.

$$P(\mathbf{x}(N)|\mathbf{Y}(i)) = \int \prod_{i=0}^{N-1} d\mathbf{x}(i) \exp[-A_0(\mathbf{X}, \mathbf{Y})] \tag{6}$$

Where $\mathbf{Y}$(i) are all measurements $\mathbf{y}$(j) for j goes from 0 to i. Using Bayes' rule and the conditional mutual information [Fano, 1961] this can be further broken down.

$$\begin{aligned} P(\mathbf{x}(i)|\mathbf{Y}(i)) &= \frac{P\left(\mathbf{x}(i), \mathbf{Y}(i)\right)}{P(\mathbf{Y}(i))} \\ &= \frac{P(\mathbf{x}(i), \mathbf{Y}(i))/P(\mathbf{Y}(i-1))}{P(\mathbf{Y}(i))/P(\mathbf{Y}(i-1))} \\ &= \frac{P(\mathbf{x}(i), \mathbf{y}(i))|P(\mathbf{Y}(i-1))}{P(\mathbf{y}(i))|P(\mathbf{Y}(i-1))} \\ &= \left[\frac{P(\mathbf{x}(i), \mathbf{y}(i)|\mathbf{Y}(i-1))}{P(\mathbf{y}(i)|\mathbf{Y}(i-1))P(\mathbf{x}(i)|\mathbf{Y}(i-1))}\right] P(\mathbf{x}(i)|\mathbf{Y}(i-1)) \\ &= \exp\left[CMI(\mathbf{x}(i), \mathbf{y}(i)|\mathbf{Y}(i-1))\right] P(\mathbf{x}(i)|\mathbf{Y}(i-1)) \end{aligned} \tag{7}$$

Since the dynamics are assumed to be a Markov process the component outside the conditional mutual information can be defined in terms of the previous state.

$$P(\mathbf{x}(i)|\mathbf{Y}(i-1)) = \int d\mathbf{x}(i-1) P(\mathbf{x}(i)|\mathbf{x}(i-1)) P(\mathbf{x}(i-1)|\mathbf{Y}(m-1)) \tag{8}$$

We can now recursively apply this step down to t=0.

$$\begin{aligned} P(\mathbf{x}(N)|\mathbf{Y}) &= \int d\mathbf{X} \exp\left\{ \sum_{i=0}^{N} CMI(\mathbf{x}(i), \mathbf{y}(i)|(\mathbf{Y}(i-1)) \right. \\ &\quad \left. + \sum_{i=0}^{N-1} \log[P(\mathbf{x}(i+1)|\mathbf{x}(i)] + \log[P(\mathbf{x}(0))] \right\} \end{aligned} \tag{9}$$

Which means that

$$\begin{aligned} A_0(\mathbf{X}, \mathbf{Y}) &= -\left\{ \sum_{i=0}^{N} CMI(\mathbf{x}(i), \mathbf{y}(i)|\mathbf{Y}(i-1)) \right. \\ &\quad \left. + \sum_{i=0}^{N-1} \log[P(\mathbf{x}(i+1)|\mathbf{x}(i))] + \log[P(\mathbf{x}(0))] \right\} \end{aligned} \tag{10}$$

Simplifications and Approximations to the Action

Based only on the assumption that the dynamics represents a Markov process, the action of a dynamical system for which measurements can be made can be represented as:

$$\begin{aligned} A_0(\mathbf{X},\mathbf{Y}) &= -\Bigg\{ \sum_{i=0}^{N} CMI(\mathbf{x}(i),\mathbf{y}(i)|\mathbf{Y}(i-1)) \\ &\quad + \sum_{i=0}^{N-1} \log[P(\mathbf{x}(i+1)|\mathbf{x}(i))] + \log[P(\mathbf{x}(0))] \Bigg\} \end{aligned} \tag{11}$$

We will first assume that each measurement ($\mathbf{y}$(t)) is independently distributed from previous measurements. This is not to say that there are not errors that are correlated with activity - but that the noise or errors created by our measurement apparatus are uncorrelated. We will additionally assume that the errors are distributed in a Gaussian manner with $\sigma_i = 1/\sqrt{Rm_i}$. This assumption is not essential, and any distribution but a Gaussians is chosen absent another obvious choice because of the resulting simplification.

As the measurements are now uncorrelated, there is no conditional probability on $\mathbf{Y}$(i-1) - this means that the conditional mutual information is simply the log of the joint probability distribution of $\mathbf{x}$(i) and $\mathbf{y}$(i). As we have assumed a Gaussian distribution, this term becomes a least squared error between the L measurements and the L measured states (or potentially functions of states).

$$-\sum_{i=0}^{N} CMI(\mathbf{x}(i),\mathbf{y}(i)|\mathbf{Y}(i-1)) \approx \sum_{i=0}^{N}\sum_{k=1}^{L} \frac{Rm_k(i)}{2}\Big[y_k(i) - x_k(i)\Big]^2 \tag{12}$$

For the model term, if the dynamics are deterministic and correct, the probability distribution is a delta function.

$$P(\mathbf{x}(i+1)|\mathbf{x}(i)) = \delta(\mathbf{x}(i+1) - \mathbf{F}(\mathbf{x}(i),\mathbf{p})) \tag{13}$$

Where $\mathbf{F}$ is the integrated map derived from the dynamics, $\mathbf{f}$.

In general the dynamics are not exact and for modeled physical system always going to be wrong to a degree by stocastic processes, unmodeled activity, or a multitude of other reasons. To account for these errors, we broaden the delta function distribution into a Gaussian distribution on the difference between the mapping $\mathbf{F}$and the next state $\mathbf{x}$.

$$-\sum_{i=0}^{N-1} \log[P(\mathbf{x}(i+1)|\mathbf{x}(i)] \approx \sum_{i=0}^{N-1}\sum_{k=1}^{D} \frac{Rf_k(i)}{2}\Big[x_k(i+1) - F_k(\mathbf{x}(i),\mathbf{p})\Big]^2 \tag{14}$$

Where D is the size of the state vector X. Rf is essentially $1/\sigma^2$ for the 'model noise' and is scaled according to the range of the dynamics and the confidence level of the dynamics,

taking into account known sources of stochasticity. Together with the measurement term this gives a total simplified action in a numerical form that is simple to work with yet broad in scope.

$$A_0 = \sum_{i=0}^{N-1} \sum_{k=1}^{D} \frac{Rf_k(i)}{2} \Big[x_k(i+1) - F_k(\mathbf{x}(i), \mathbf{p}) \Big]^2 + \sum_{i=0}^{N} \sum_{k=1}^{L} \frac{Rm_k(i)}{2} \Big[y_k(i) - x_k(i) \Big]^2 \quad (15)$$

The code outlined here will use the variational method to find a minimum of this action that corresponds to the most likely set of states and parameters for the set of measurements provided.

A Note on Minima

For chaotic systems, the action space tends to have a fractal number of local minima, and as such finding a global minima becomes a challenge. The measurement term in the cost function provides and implicit coupling term to the measured quantities. This coupling term can alter the Jacobian of the system in such a way that there are no longer any local positive conditional Lyaponov exponents, in other words the coupled system is no longer chaotic around the measured point because of the information provided by the measurement. This 'synchronization' is heavily dependent on the choice of measurements and the activity of the system explored by the choice of stimulus. There is a huge amount to write on this subject which I am working on compiling for my thesis but the short summary is that like all tools, this method is difficult to use for some problems and even if used correctly may not be appropriate for the project at hand.

Installing Required Programs and Packages

This document will assume that the user is using a Linux distribution and has sudo access - while it may be possible to install this setup on another Unix system (such as Mac) I have never attempted to do so.

IPOPT

Download

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

- Download and unzip latest version of IPOPT
- 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
```

- Get the HSL subroutines from `http://hsl.rl.ac.uk/ipopt`
- 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
```

minAzero.py

minAzero 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

- discAzero.py
  -Discretizes equations and creates strings for Jacobian and Hessian Elements.
- makecppAzero.py
  -Writes C++ file linking to IPOPT libraries using strings from discAzero.py
- makehppAzero.py
  -Writes header file for above
- makemakeAzero.py
  -Writes makefile for problem. Will need to be changed based on install location of IPOPT
- makeoptAzero.py
  -Writes settings file for IPOPT

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

Installation

- These python scripts link to the sympy library. To install these, use sudo apt-get install sympy or download directly from sympy.org.
- In order to link to your IPOPT libraries correctly, several lines in makemakeAzero.py need to be modified.

Line 58

```
CXXLINKFLAGS = -Wl,--rpath -Wl,/home/chris/Ipopt-3.8.3/build/lib\n\
```

Change /home/chris/Ipopt-3.8.3/build/lib to the /lib folder in the build directory for your IPOPT installation.

Line 66

```
prefix = /home/chris/Ipopt-3.8.3/build\n\
```

Change this line to the build directory in which IPOPT is installed.

Running the Code

minAzero uses two text documents (along with any needed data files) as input, equations.txt and specs.txt. Once these are filled

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. It is essential that these numbers are accurate as minAzero uses this to know how many lines to read for each component of the code.
- 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 couplings, 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 jacobian and hessian (an example of this is included)

specs.txt contains run specific information such as file names, variable bounds, and problem length. This file can be edited without recompiling the code.

- First line is the number of full steps the code will use. Because the code is compiled using a midpoint method, the actual problem length will double this plus one.
- Second line is the number of lines in each input file to skip. This allows for the code to start at any point in a long data set.
- Third line is double the time step of the data. Again since a midpoint method is used, the time step is for a whole step - which includes two points.
- If you wish to start at a non constant guess, you can put a 1 followed by a line with an initial condition file. This file should have one column for each state. If you do not want to include an initial condition file, use 0
- One line for each of the measured data file names. Each file should be a single column.
- One line for each of the stimulus data file names. Each file should be a single column.
- For each variable, the lower bound, upper bound, initial guess, and RF value separated by commas. The initial value is ignored if the initial conditions file is used, but a value must be included regardless.
- For each coupling term, a line with lower bound, upper bound, and initial value separated by commas followed by another line for lower bound, upper bound, and initial value for the derivative of the coupling term.
- For each parameter a lower bound, upper bound, and initial guess separated by commas

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

```
$ minAzero.py
$ make
$ ./(problem_name)_cpp
```

If data files are missing or too short, the code will segfault. The outputs are **data.dat** containing all state variables at all times, and **param.dat** with parameter values.

Example equations.txt

```
# lines starting with # are ignored
# Problem Name
Colpitts
# nY,nP,nU,nI,nF,nM
3,3,1,0,0,1
# equations
yy+u00*(Data-xx)
-gam*(xx+zz)-qq*yy
eta*(yy+1-exp(-xx))
# measurement portion of Objective/Cost function
# the model portion is generated automatically
(Data-xx)*(Data-xx)+u00*u00
# variable names (nY)
xx
yy
zz
# parameter names (nP)
gam
qq
eta
# coupling names (nU)
# most minAzero formulated problems will not need a coupling term
# this term is essential for equality constrained dynamics
u00
# data names (nM)
Data
# stimuli names (nI)
# no stimuli in this problem
# functions (nF)
# list of functions and number of arguments - will require myfunctions.cpp
# with definitions of f(x) and its jacobian and hessian.
# alpha,4
```

Example specs.txt

```
# lines starting with # are ignored
# The problem length - actual length will be 4001 due to midpoint method
2000
# How much data to skip.
# In case you do not want to start at the beginning of the data file
10
# Time step - this is twice the time step of the data,
# since the data includes time and midpoints.
0.2
# Data File names - measurements
# each measurement needs its own file in its own row
testx.dat
#colscale1x.dat
# Data File names - stimuli
# each stimuli needs its own file in its own row
# No stimuli for this problem
# Boundary & initial conditions
# 0 for no initial data file, 1 for data file
# A data file must include values for all state variables at each time point.
0
# If above is 1, list name of data file next.  If 0, no entry needed.
#start.dat
# State Variable bounds:
# These are in the formats: lower bound, upper bound, initial guess, RF
# Boundary & initial conditions
# x
-100, 100, 20,1
# y
-100, 100, 10,1
# z
-100, 100, 1,1
# each coupling needs two entries, for u and du as there is no equation for u
# u00
# couplings are typically not needed in unconstrained problems
# but are essential in equality constraint problems
0,1,0
-10,10,0
# each parameter needs upper bound, lower bound, initial guess
# p1
0, 100, 0.016
# p2
0, 100, 0.14
```

```
# p3
0, 100, 1.26
```

References

[Fano, 1961] Fano, R. M., *Transmission of Information: A Statistical Theory of Communication*, Wiley, New York, (1961).

[Toth, et al., 2011] B. A. Toth, Kostuk, M., C. D. Meliza, D. Margoliash, and H. D. I. Abarbanel,"Dynamical Estimation of Neuron and Network Properties I: Variational Methods," *Biological Cybernetics*, Biol Cybern DOI 10.1007/s00422-011-0459-1, 1 - 21, (2011).

[Toth, 2010] B. A. Toth, "Python Scripting for Dynamical Parameter Estimation in IPOPT,"*SIAG/OPT Views-and-News***21:1**, 1-8 (2010).

[Wächter & Biegler, 2006] A. *Wä*chter and L. T. Biegler, "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming,"*Mathematical Programming* **106**, 25-57 (2006).