Running neuroDA code for Rhabdomys: A short walk through

Matthew Moye

October 29, 2021

1 Organizing Data for Estimation

The main piece of code utilized for massaging the data is contained in the file **'make_rhabdomys_neuroDA_details.m'** will isolate portions of the relevant data from the *rhabdomys* dataset. `for CASE = CASENUM` at the top illustrates which of the `CASE` from the corresponding switch to use. These relate to the different data files, apriori estimates for the capacitance, and which protocols (1:13) to use. The data is organized symmetrically with the current clamp protocol being $I_{app} = 0$ until index `StepOne=2.6641e4` where it will be held at a value -30:30 based on the protocol index until `StepTwo=5.1641e4`. `VSeriesStepOneinds` will contain data with the first step within its bounds, but not the release step, and the time window is declared as `timeWindow`. Similarly for the return step (`stepTwoinds`). `timeWindowSpontaneous` nonempty implies the index 7 for spontaneous activity. `dsf` is the down sampling factor which will be used to downsample data outside of preserved regions near the pulse and during the action potential. If using a period of data with both the intial and return step, declare `timeWindowAll` and `VSeriesStepOneStepTwoinds`.

The integer `CASE` numbers overall are only organized by order of when they were run, but dissimilar settups for the same cell are given decimals for distinction. These will generate unique names later for the output .mats. Only `CASE=2` is shown for brevity.

An example:

```
        case 2
filename=[pwd ...
    '\rawData\rhabdomys\Hyper-Depo-Pulses\190620_Pulses\matFiles\Cell10_0003.mat'];
[a,filenameshort,c]=fileparts(filename);
dateinfo='_190620_Pulses';
mDFN_prefix=[filenameshort,dateinfo];
CapEst=14.2;
VSeriesStepOneinds=[1 6 10 13];
VSeriesStepTwoinds=[1 5];
timeWindow=[800 1300];
timeWindow2=[2000 2500];
```

```
timeWindowSpontaneous=[500 2000];
```

The second section of code declares the parameters bounds which should correspond to the model selected in `modeltouse` . The parameter bounds (and a final column which normally goes unused) are fed into a variable `PDATA`.

The third section will use the information from the first section to create seperate .csv files connected into another .mat. The .mat will contain the information associated with the chosen model, parameter bounds, and which data files to use for a given cell. The .csv filenames themselves are relatively descriptive, but if working on the same cell in the same directory, files can easily be overwritten without recognizing it; best to only consider one permutation of the cell declaration at a time when running the neuroDA code and reproduce each of these steps whenever making adjustments.

In this section, the data is also downsampled according to a strategy of using all the data when exceeding a threshold arbitrarily set as -20 and using 30 ms of data around the point at which the threshold is achieved. The data is also preserved around the initial step and return step index.

The last section is a switch block associated with `modeltouse`. Here, the cases relate to different potential models. One must construct the RHS in the casadi syntax, akin to `ODEmodel=@casadi_Belle2009_new`.

Here is the default model used in the paper:

```
        case 6.5
Nstate=5;
modelDataFileName=[mDFN_prefix,'data_SCN_new_mtau_fastm_twoleaks','_case',num2str(CASE)];
modelDataFileName=strrep(modelDataFileName,'.','pt');
PDATA=PDATA_new_mtau_fastm_twoleaks;
ODEmodel=@casadi_Belle2009_new_mtau_fastm_twoleaks;

save(modelDataFileName,'PDATA','Nstate','compartments','nleaks','vseriesCollection','ODEmodel');
ODEmodel=@casadi_Belle2009_new_mtau_fastm_twoleaks;
```

The corresponding casadi file:

```
function [f,fu] = casadi_Belle2009_new_mtau_fastm_twoleaks()
% NaKL 4 kinetic parameter dynamics
%X = x(probinfo.xind);

%p = x(probinfo.pind);
%Iapp=probinfo.Iapp;
% Calculate h
%Iapp=interp1(Iappt,Iappdata,t);
timeScaleFactor=10;
import casadi.*

V= SX.sym('V');
h_na = SX.sym('h_na');
n = SX.sym('n');
m_ca = SX.sym('m_ca');
h_ca = SX.sym('h_ca');
```

```
17
18 x = [V,h_na,n,m_ca,h_ca];
19
20 C = SX.sym('C');
21 Ena = SX.sym('Ena');
22 Ek =   SX.sym('Ek');
23 Eca= SX.sym('Eca');
24 %Eleak   = SX.sym('Eleak');
25 Gna= SX.sym('Gna');
26 Gk= SX.sym('Gk');
27 Gca = SX.sym('Gca');
28 Gleak_na =  SX.sym('Gleak_na');
29 Gleak_k = SX.sym('Gleak_k');
30 %m_na
31 vm_na= SX.sym('vm_na');
32 dvm_na= SX.sym('dvm_na');
33
34 %h_na
35 vh_na= SX.sym('vh_na');
36 dvh_na= SX.sym('dvh_na');
37 th0_na = SX.sym('th0_na');
38 th1_na = SX.sym('th1_na');
39 vht_na = SX.sym('vht_na');
40 dvht_na = SX.sym('dvht_na');
41
42 %n%
43 vn= SX.sym('vn');
44 dvn= SX.sym('dvn');
45 tn0 = SX.sym('tn0');
46 tn1= SX.sym('tn1');
47 vnt = SX.sym('vnt');
48 dvnt = SX.sym('dvnt');
49
50 %m_ca
51 vm_ca= SX.sym('vm_ca');
52 dvm_ca= SX.sym('dvm_ca');
53 tm0_ca = SX.sym('tm0_ca');
54 tm1_ca = SX.sym('tm1_ca');
55 vmt_ca = SX.sym('vmt_ca');
56 dvmt_ca = SX.sym('dvmt_ca');
57 %h_ca
58 vh_ca= SX.sym('vh_ca');
59 dvh_ca= SX.sym('dvh_ca');
60 th0_ca = SX.sym('th0_ca');
61 th1_ca = SX.sym('th1_ca');
62 vht_ca = SX.sym('vht_ca');
63 dvht_ca = SX.sym('dvht_ca');
64
65
66
67 p = [C, Ena, Ek, Eca,  Gna, Gk, Gca, Gleak_na,Gleak_k]';
68 p = [p; vm_na; dvm_na;
69 vh_na; dvh_na; th0_na; th1_na; vht_na; dvht_na; ...
70 vn; dvn; tn0; tn1; vnt; dvnt; ...
71 vm_ca; dvm_ca; tm0_ca; tm1_ca; vmt_ca; dvmt_ca;
72 vh_ca; dvh_ca; th0_ca; th1_ca; vht_ca; dvht_ca;];
73
```

```
74
75  u = SX.sym('u');
76  ydat = SX.sym('ydat');
77  Iappx = SX.sym('Iappx');
78  m_na=ainf_fun(V,vm_na,dvm_na);
79
80  xdot = [ ...
        fV(V,m_na,h_na,n,m_ca,h_ca,C,Ena,Ek,Eca,Gna,Gk,Gca,Gleak_na,Gleak_k,Iappx);
81  fa(h_na,V,vh_na,dvh_na,th0_na,th1_na,vht_na,dvht_na);
82  fa(n,V,vn,dvn,tn0,tn1,vnt,dvnt);
83  fa(m_ca,V,vm_ca,dvm_ca,tm0_ca,tm1_ca,vmt_ca,dvmt_ca);
84  fa(h_ca,V,vh_ca,dvh_ca,timeScaleFactor*th0_ca,timeScaleFactor*th1_ca,vht_ca,dvht_ca); ...
        ];
85  xdotu = [ ...
        fVu(V,m_na,h_na,n,m_ca,h_ca,C,Ena,Ek,Eca,Gna,Gk,Gca,Gleak_na,Gleak_k,Iappx,u,ydat);
86  fa(h_na,V,vh_na,dvh_na,th0_na,th1_na,vht_na,dvht_na);
87  fa(n,V,vn,dvn,tn0,tn1,vnt,dvnt);
88  fa(m_ca,V,vm_ca,dvm_ca,tm0_ca,tm1_ca,vmt_ca,dvmt_ca);
89  fa(h_ca,V,vh_ca,dvh_ca,timeScaleFactor*th0_ca,timeScaleFactor*th1_ca,vht_ca,dvht_ca); ...
        ];
90
91
92  % xdot = [ ...
        fV(V,m_na,n,m_nap,k,Gna,Gk,Gks,Gnap,Gleak,Gtonic,Iappx,Ena,Ek,Eleak,Esyn,C);
93  %      fa(n,V,vn,dvn,tn1);
94  %      fa(k,V,vk,dvk,exp(tk1));];
95  %  xdotu = [ ...
        fVu(V,m_na,n,m_nap,k,Gna,Gk,Gks,Gnap,Gleak,Gtonic,Iappx,Ena,Ek,Eleak,Esyn,C,u,ydat);
96  %      fa(n,V,vn,dvn,tn1);
97  %      fa(k,V,vk,dvk,exp(tk1));];
98
99  f = Function('f', {x,p,Iappx}, {xdot});
100 fu = Function('fu', {x,p,Iappx,ydat,u}, {xdotu});
101 %f= Function('Ff',{x,p,Iappx}, {xdot}, ...
        char('x0','p0','Iapp'),char('xf'));
102 %fu = Function('Ffu', {x,p,Iappx,ydat,u}, {xdotu}, ...
        char('x0','p0','Iapp','ydat','u'),char('xf'));
103
104
105 function dVdt = ...
        fV(V,m_na,h_na,n,m_ca,h_ca,C,Ena,Ek,Eca,Gna,Gk,Gca,Gleak_na, ...
        Gleak_k,Iapp)
106 Ina = Gna*(m_na^3)*h_na*(V-Ena);
107 Ik = Gk*(n^4)*(V-Ek);
108 %Ileak = Gleak*(V-Eleak);
109 Ileak_na = Gleak_na*(V-Ena);
110 Ileak_k = Gleak_k*(V-Ek);
111 Ica = Gca*m_ca*h_ca*(V-Eca);
112 dVdt = (1/C)*(Iapp-Ina-Ik-Ileak_na-Ileak_k-Ica);
113 end
114
115 function dVdt = ...
        fVu(V,m_na,h_na,n,m_ca,h_ca,C,Ena,Ek,Eca,Gna,Gk,Gca,Gleak_na,Gleak_k,Iapp,u,ydat)
116 Ina = Gna*(m_na^3)*h_na*(V-Ena);
117 Ik = Gk*(n^4)*(V-Ek);
118 %Ileak = Gleak*(V-Eleak);
119 Ileak_na = Gleak_na*(V-Ena);
```

```
120 Ileak_k = Gleak_k*(V-Ek);
121 Ica = Gca*m_ca*h_ca*(V-Eca);
122 Iu= u*(V-ydat);
123 dVdt =  (1/C)*(Iapp-Ina-Ik-Ileak_na-Ileak_k-Ica)-Iu;
124 end
125
126 function dadt = fa_tau_constant(a,V,va,dva,ta0)
127 ainf =ainf_fun(V,va,dva);
128 %tau = ta1./cosh((V-va)./(dva));
129 tau = ta0;
130 dadt = (ainf-a)/tau;
131 end
132
133 function dadt = fa(a,V,va,dva,ta0,ta1,vat,dvat)
134 ainf =ainf_fun(V,va,dva);
135 tau = ta0 + ta1*(1-tanh((V-vat)/dvat)^2);
136 dadt = (ainf-a)/tau;
137 end
138
139 function ainf = ainf_fun(V,va,dva)
140 ainf =(1/2)*(1+tanh((V-va)/(dva)));
141 end
142
143 %  (.5*(1+tanh((vx-vm)/dvm)) - ...
        mx)/((tm0+tm1.*(1-tanh((vx-vm)./dvm)^2)));
144
145 end
```

Models which manifest in the Rhabdomys paper:

- `modeltouse=6.5` This is the default model,

$$C\frac{dV}{dt} = -\left(I_{Na} + I_K + I_{Ca} + I_{Leak,Na} + I_{Leak,K}\right) + I_{app}$$

  There are 6 distinct parameters representing the gating variables: $va, dva, ta0, ta1, vat, dvat$. There are a total of 5 dynamic variables as sodium activation is assumed to be instantaneous.

- `modeltouse=12` This is the model used to investigate delays,

$$C\frac{dV}{dt} = -\left(I_{Na} + I_K + I_{Ca} + I_{Leak,Na} + I_{Leak,K} + I_H + I_A\right) + I_{app}$$

  There are 5 distinct parameters, as we assume $vat = va$. There are 7 dynamic variables, as the activation of sodium and I_A are assumed to be instantaneous.

- `modeltouse=15` This model is used to investigate I_A influence on firing properties,

$$C\frac{dV}{dt} = -\left(I_{Na} + I_K + I_{Ca} + I_{Leak,Na} + I_{Leak,K} + I_A\right) + I_{app}$$

  There are 5 distinct parameters, as we assume $vat = va$. There are 6 dynamic variables, as the activation of sodium and I_A are assumed to be instantaneous.

A word of caution with the above models: within the ODE file, I often have made scalings of the time constants so as to try to condense the space of parameters within a few orders of magnitude. These may not be uniform across differing models as goal points fluctuated over the course of the project. If possible, please try to just play around with the parameter bounds prior to adjusting existing ODE models so as to not break any old results; if this is not possible, please make a history of the code or save a copy of it. We have also explored logarithmic scaling of parameters which have unique sign, and I encourage considering logarithmic scalings for these situations as well.

2 Running Data Assimilation

Once the casadi ODE file and the .csv and linking .mat are created, the next file to edit is
`~\code\Rhabdomys_strong_DA.m`.
This is file which will ultimately be run, potentially from command line, as it accepts at most 3 arguments. The first argument is the `SEEDNUM` for some seed for the random number generator; typically, we will use an array of seeds to initialize the system at different random starting points to hopefully find the global minimum using this multi-start strategy. The second argument is `CASE` which will be a number descriptor for the data/ model implementation being investigated. This need not be directly the same as a combination of the previous cell IDs and model IDs, but should be close enough for quick surveying. If creating a new combination of these, please use the following template:

```
case 2
modelDataFileName='Cell10_0003_190620_Pulsesdata_SCN_new_mtau_fastm_twoleaks_case2.mat';
ModelLoaded=1;
```

where the `modelDataFileName` is the output from running **'make_rhabdomys_neuroDA_details.m'**. If using controlled 4D-Var, don't worry about the settings at the end, aside from maybe `'SmoothControl',0` which could be set to 1 at the cost of $N/2$ additional variables to help smooth the control a bit. Turning this on will add additional $N/2$ constraints imposed on the control. Also `'ControlBound',1.0` which could be changed from 1 to say .1 to really prevent dependence on the control at the cost of potentially a highly irregular problem. The control, u is bounded between 0 and `ControlBound`.

3 Implementation Considerations

Note that the traditional way of solving problems of this nature demand computing clusters, as the code boils down to essentially a multi-start, nonlinear non-convex local optimization problem. I would advise casting a wide-net (we used often between 50-200 initial conditions) for initializations and SEEDS, and running the problem in parallel. Currently, issuing different `SEEDNUM` will draw

from a uniform distribution of states and parameters over there bounds. A more optimal implementation of initial guesses might use latin hypercube sampling (stay tuned for this!)

Warning: the RAM required can potentially exceed 8GB depending on problem size. My laptop clocked around 8.3 GB of max memory used.

If using the ColPack repository (recommended for the *rhabdomys* data fitting) some ".mtx" files may be generated and not removed during the hessian coloring process; ignore any warning issued and feel free to delete any extraneous files later.

Once the .mat file of the estimated model is generated, feed this into the file `'plot_estimated_model.m'`. The script `'main_example_Rhabdomys.m'` encloses these functions in a logical format.