IPOPT and Neural Dynamics
Tips, Tricks and Diagnostics

G. Hooker and L. Biegler

Abstract

We describe the process of fitting a phenomenological model for the spiking behavior of a zebra-fish neuron using the IPOPT routines described in Wächter and Biegler 2006. These routines provide a computationally efficient methods for estimating parameters in nonlinear differential equations. However, convergence to global solutions is not guaranteed and we describe a number of strategies that were necessary to obtain parameter estimates from these data. The intention of fitting these data was to improve on the phenomenological models used. The use of diagnostic techniques with IPOPT software is also described.

1 Introduction

There has been increasing interest in performing inference for systems governed by nonlinear ordinary differential equations. See, for example, Ramsay et al. 2007, Ionides et al. 2006 and Huang et al. 2006 for some of the many recent developments. Estimating parameters for such systems has been a computationally challenging exercise, made difficult by the lack of explicit descriptions of a system trajectory, the sensitivity of estimated trajectories to numerical errors and the complex dependence of trajectories on system parameters. Many of the methods developed to estimate parameters in such systems require careful tuning in order to converge on good estimates.

This paper presents a case study of the use of the IPOPT routines (Wächter and Biegler 2006) to solve a parameter estimation problem in neural dynamics. These routines use a collocation method to represent solutions to the system trajectories. They attempt to jointly optimize the system while solving collocation equations, leading to a constrained optimization problem. Here, we apply IPOPT to data collected from a single-neuron patch clamp experiment. The system used to describe the data is linear in the parameters. Nonetheless, finding optimal parameter values is a numerically challenging problem for which straightforward gradient descent methods failed. IPOPT, by contrast, succeeded in finding parameter estimates, but a number of modifications to the original problem were required before it converged. The purpose of this paper is to elucidate these modifications as strategies to achieve better performance from constrained optimization routines.

In the numerical difficulties of estimating parameters for nonlinear systems are exacerbated by a poor agreement between theoretical descriptions of a system and empirical measurements taken of it. This makes good data-driven techniques for suggesting improvements to such models an important area of research. This problem was considered in order to demonstrate the use of estimated forcing functions as diagnostic tools. A more detailed description of these ideas is given in (Hooker 2007). We demonstrate the application of IPOPT in implementing these methods to obtain an improved model for the zebrafish data.

2 Data and Models

A patch clamp experiment allows the study of neural behavior by measuring the potential along the axon membrane of a neuron. The result of such an experiment performed on a zebra-fish motor neuron are given in Figure 1. Here, an input current was stepped up from zero and then returned to zero. The neuron responds by firing a series of action potentials – the long series of spikes – and ceases when the current is shut off. Neurons can exhibit a wide range of behaviors, including hysteresis – or continuing to spike after the stimulus has been removed and bursts – or groups of spikes between longer periods of no activity. A great deal of work has been done in producing models to mimic these behaviors.

Broadly, models for neural behavior fall into two classes; complex descriptions based on physiological properties and simplifications that capture the qualitative features of physiological models. See Wilson (1999) for an introductory overview of the field. These simplifications are frequently desirable to work with in being more tractable for mathematical analysis, and computationally cheaper to simulate. Our task here will be to assess and improve the quantitative performance of these simplified models.

As a starting point, we use the following system of equations:

$$\frac{dV}{dt} = f_V(V, R, p) = p_1 + p_2V + p_3V^2 + p_4V^3 + p_5R + p_6VR \tag{1}$$

$$\frac{dR}{dt} = f_R(V, R, r) = r_1 + r_2V + r_3R \tag{2}$$

$$V(0) = V_0, \ R(0) = R_0 \tag{3}$$

Here V represents the transmembrane potential while R is a recovery variable representing a collection of ion currents. Equations of this form are derived in Wilson (1999) by inspection of the isoclines of the Rinzel equations Rinzel and Keller (1973), in turn a simplification of the Hodgkin-Huxley equations (Hodgkin and Huxley 1952). Equation (1) expands the products of polynomial terms given in Wilson (1999). The products are meant to represent approximations to elements of the original Rinzel system, but the forms above are considerably more tractable and do not affect the number of parameters in the model.

Initial parameter estimates for these equations were obtained by taking parameter values in Wilson (1999) and re-scaling so that the estimated solutions for V take the same range as those found in the data. This may be done through the observation that

$$\frac{d}{dt}(aV + b) = af\left(\frac{(aV + b) - b}{a}\right). \tag{4}$$

When f is polynomial in V and linear in its parameters as in (4), this transformation does not change its form. The parameters were then further transformed so that the solutions for V exhibited the same period as the observed data. This is achieved through:

$$\frac{d}{dt}V(kt) = k\frac{d}{ds}V(s)$$

for $s = kt$. This also leaves the form of f unchanged.

Finally, we observe that (4) can also be applied to R. Since no measurements are available for R, the transform is left unconstrained and was instead chosen so that R has approximately the same range as V. This has been done in order to improve the numerical stability of the estimation procedure. This observation also implies that only one of r_1, r_2 and r_3 may be estimated from our data.

3 Collocation Methods, Solution Trajectories and IPOPT

The methods chosen to fit the model and these data employ the IPOPT routines for constrained optimization described in Wächter and Biegler (2006) along with a collocation method. Here the system is broken into K small intervals; we used 1000 intervals of one unit each. Within each interval $[t_i \; t_{i+1}]$, the derivative of the the trajectory is interpolated by Lagrange polynomials, $L_{ij}(t)$:

$$\dot{V}(t) = \sum_{j=1}^{s} \dot{V}(c_{ij})L_{ij}(t) = \sum_{j=1}^{s} f_V(V(c_{ij}), R(c_{ij}), p)L_{ij}(t)$$
$$\dot{R}(t) = \sum_{j=1}^{s} \dot{R}(c_{ij})L_{ij}(t) = \sum_{j=1}^{s} f_R(V(c_{ij}), R(c_{ij}), r)L_{ij}(t)$$

where c_{ij} are chosen to be s Gauss-Radau quadrature points $c_{ij} = t_i + (t_{i+1} - t_i)\rho_j$ with $\rho_j \in [0, 1]$ and $\rho_s = 1$. For the system above, we selected $s = 3$. This leads to the representation of the solution at the collocation points:

$$\begin{aligned}
V(c_{ij}) &= V(t_i) + \sum_{k=1}^{s} f_V(V(c_{ik}), R(c_{ik}), p) \int_{t_i}^{c_{ij}} L_{ik}(t)dt \\
&= V(t_i) + \sum_{k=1}^{s} A_{ik} f_V(V(c_{ik}), R(c_{ik}), p) \qquad (5) \\
R(c_{ij}) &= R(t_i) + \sum_{k=1}^{s} f_R(V(c_{ik}), R(c_{ik}), r) \int_{t_i}^{c_{ij}} L_{ik}(t)dt \\
&= R(t_i) + \sum_{k=1}^{s} A_{ik} f_R(V(c_{ik}), R(c_{ik}), r) \qquad (6) \\
& V(0) = V_0,\ R(0) = R_0
\end{aligned}$$

for a collocation matrix A. Equations (5) and (6) now represent $2Ks$ equations in the $2Ks$ unknowns, $\{V(c_{ij}), R(c_{ij})\}$. Solving for these has been shown to be equivalent to using fully implicit Runge-Kutta methods for estimation of solutions to equations such as (1,2). See Deuflhard and Bornemann (2000).

Parameter estimation via the IPOPT routines now attempts to find a solution to (5,6) while at the same time minimizing the squared error criterion over parameters p and r.

$$\text{SSE}(p, r) = \sum_{i=1}^{N} (v_i - V(t_i))^2 \qquad (7)$$

for observations v_i given at times t_i. The resulting parameter estimation problems can be expressed as large-scale, structured NLP problems of the form,

$$\begin{aligned}
\min \quad & f(x) \\
\text{s.t.} \quad & c(x) = 0 \\
& x_L \le x \le x_U \qquad (8)
\end{aligned}$$

where $x \in \Re^{nx}$ represents all the variables in the system; the discretized estimates of the states, $V(c_{ik})$, $R(c_{ik})$ as well as any parameters in p, r, V_0 and R_0 which are to be estimated. $f(x)$ is then just (7) while $c(x)$ is given by (5)-(6). x_L and x_U represent lower and upper bounds on these variables which have not been imposed here, but will be useful later.

Using IPOPT, the NLP problem (8) is transformed by adding logarithmic barrier functions to the objective,

$$\begin{aligned}
\min \quad & \varphi_\mu(x) = f(x) - \mu \left[\sum_{i=1}^{nx} \ln\left(x^{(i)} - x_L^{(i)}\right) + \sum_{i=1}^{nx} \ln\left(x_U^{(i)} - x^{(i)}\right) \right] \\
\text{s.t.} \quad & c(x) = 0 \qquad (9)
\end{aligned}$$

where μ is a barrier parameter satisfying $\mu > 0$. Under mild regularity conditions, solutions of (9) converge to the solution of (8) as $\mu \to 0$. Moreover, the primal-dual optimality conditions of (9) resemble those of the original NLP problem and are defined by,

$$\begin{aligned}
\nabla_x f(x) + \nabla_x c(x)\lambda - \nu_L - \nu_U &= 0 \\
c(x) &= 0 \\
(X - X_L)V_L e - \mu\, e &= 0 \\
(X_U - X)V_U e - \mu\, e &= 0
\end{aligned} \tag{10}$$

where $X, X_L, X_U, V_L, V_U \in \Re^{nx \times nx}$ are diagonal matrices whose diagonal entries are the components of x, x_L, x_U, ν_L and ν_U, respectively; $e = [1, 1, \dots, 1]^T \in \Re^{nx}$, λ is the vector of Lagrange multipliers for the equality constraints; and ν_L, $\nu_U \in \Re^{nx}$ is an estimate of the multipliers for the bound constraints of the original NLP problem. The optimality conditions can be solved efficiently by applying Newton's method, which requires the solution of a large and sparse linear system at each iteration given by,

$$\begin{bmatrix} \nabla_{x,x} L_k + \Sigma_k & \nabla_x c(x_k) \\ \nabla_x c(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_u(x_k) + \nabla c(x_k)\lambda_k \\ c(x_k) \end{pmatrix} \tag{11}$$

where L_k is the Lagrangian function of the original NLP problem (8) evaluated at iteration k, Σ_k is defined by,

$$\Sigma_k = (X_k - X_L)^{-1} V_L^k - (X_U - X_k)^{-1} V_U^k \tag{12}$$

and the bound multipliers are updated at each iteration from,

$$\Delta \nu_L^k = (X_k - X_L)^{-1}(\mu\, e - V_L^k \Delta x_k) - \nu_L^k \tag{13}$$

$$\Delta \nu_U^k = (X_U - X_k)^{-1}(\mu\, e + V_U^k \Delta x_k) - \nu_U^k \tag{14}$$

The solution of linear system (11) is the core step of the optimization algorithm and requires most of the computational time. This linear system can be solved efficiently by sparse, symmetric, indefinite solvers.

Convergence to local optima, starting arbitrarily far away, is promoted using a novel filter line search strategy. Line search methods require the Hessian matrix $H_k = \nabla_{x,x} L_k + \Sigma_k$ to have strictly positive curvature in the null space of the linearized constraint gradients. Moreover, under the assumption that $\nabla_x c(x_k)$ has full rank, the projection of H_k onto the null space of $\nabla_x^T c(x_k)$ is positive definite if and only if the iteration matrix in (11) has n positive and m negative eigenvalues. However, due to severe nonlinearity on the problem or non-informative data, respectively, the linear independence and positive curvature conditions, may not hold at intermediate iterations, and the matrix in (11) becomes singular. To correct for this, IPOPT adds diagonal correction terms to the (so-called KKT) matrix in (11), leading to:

$$\begin{bmatrix} \nabla_{x,x} L_k + \Sigma_k + \delta_1 I & \nabla_x c(x_k) \\ \nabla_x c(x_k)^T & -\delta_2 I \end{bmatrix} \tag{15}$$

with $\delta_1, \delta_2 > 0$. In order to detect whether a modification of the Hessian is necessary, the inertia of this iteration matrix (i.e., the number of positive, negative and zero eigenvalues) can be calculated from the linear solver and monitored by the algorithm (Wächter and Biegler 2006). Moreover, if the diagonal terms δ_1, δ_2 are zero at the solution, then sufficient second order optimality conditions hold. From a statistical perspective, this gives the important result that the parameters have been uniquely determined, and the data are sufficiently informative.

IPOPT is open-source software and can be downloaded from:
`http://projects.coin-or.org/Ipopt`.

A set of AMPL templates that describes the collocation formulation for a small reactor optimization problem can be found on:
`http://www.andrew.cmu.edu/user/vzavala/dynopt.html`.

4 Estimating Parameters in Zebra-fish Data with IPOPT

Parameters were fit to the data via the collocation method combined with IPOPT as described above. Due to the nonlinearities in both the constraints and the parameters, the solution of the constrained problem may be slow to converge and runs some risk of failure, even with the compensating parameters δ_1 and δ_2 in (15). A number of strategies can be used to improve this performance, especially in preventing ill-conditioning of the matrix in (15):

- Add loose bounds to the state variables and parameters. Even though they are not active at the optimum, the bounds contribute positive entries to Σ in (15) and promote positive curvature of the reduced Hessian.
- For overparametrized problems, estimation with a reduced set of parameters will lead to a positive reduced Hessian in the neighborhood of the optimum. Such problems are much better conditioned and much easier to converge. Hence we use the following heuristic:
  - Fix a set of parameters, solve the NLP and solve a sequence of problems by systematically "unfixing" parameters. This procedure generates a monotonically decreasing sequence of $SSE((p, r)$ values through the solution of well-conditioned subproblems. Moreover, along with the regularization parameters δ_1, δ_2, these partial solutions are excellent initializations that promote faster convergence of the overall problem.

These techniques were employed for the equations above.

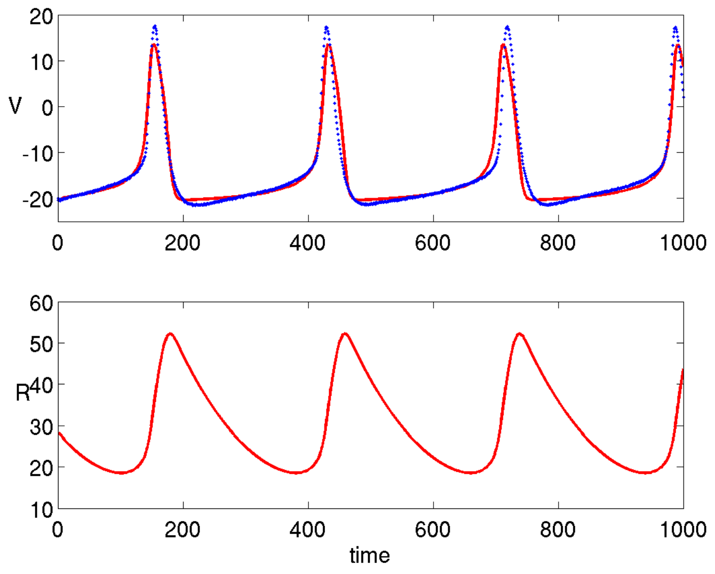

Figure 1: A least squares fit to the Rinzle equations estimated using the IPOPT constrained optimization routines.

- Constraints that $-50 \leq V \leq 50$ and $10 \leq R \leq 50$ were imposed. These bounds are comfortably outside the observed values of V and the values of R at the initial parameter estimates.
- The parameters were estimated by sequentially unfixing two parameters at a time. That is, first optimizing for V_0, R_0 with the remaining parameters held fixed. Then p_1 and p_2 were also relaxed, leading to a four-parameter optimization and so forth.

As noted in Section 2, r_1 and r_2 are unidentifiable in this system and were held constant throughout. Additional parameters representing the initial conditions for V and R were also estimated as parameters in this scheme.

Under this scheme, the system converged with a reduction in squared error from 17465.2 to 6710.4. A representation of the final estimated trajectories is given in Figure 1. While the general characteristics of the observed data are maintained, some of the dynamics of the system are not captured. The peaks are consistently underestimated and a strong elbow at the end of the peak is not present in the data. Details of the implementation of the estimation scheme using the AMPL modeling language are provided in the appendix and electronic supplement.

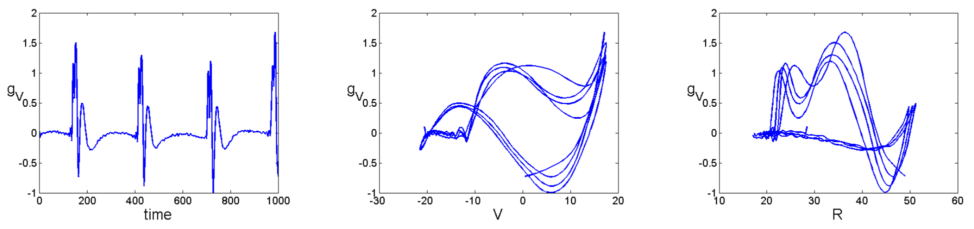

Figure 2: From left to right: lack of fit forcing functions for V, plotted against estimated trajectories for V and R.

5 Diagnostics

In order to improve this simple set of equations, a diagnostics forcing function was estimated for V. That is (1) was modified to

$$\dot{V} = p_1 + p_2 V + p_3 V^2 + p_4 V^3 + p_5 R + p_6 V R + g(t)$$
$$g(t) = \sum_{i=1}^{k} \phi_i(t) b_i$$

where the ϕ_i are 202 equi-spaced linear B-splines on the interval $[0, 1000]$ and the b_i were estimated using the IPOPT routines again with the parameters p, r, V_0, R_0 held fixed at their estimated values. The addition of forcing functions on R was also considered, but did not provide the interpretable results found below.

The resulting fit to the data and the estimated $g(t)$ are given in Figure 2. Here $g(t)$ demonstrates a consistent cycle. In order to represent it, we treat $g(t)$ as a residual and plot it against the estimated values of V and R. This plot provides a diagnostic for consistent functional relationships that are missing from the original model equations. In this case, we have chosen to divide the (V, R) phase space in two around the indicated dashed line. On either side of the line, an approximately cubic function provides a reasonable description for lack of fit and there are relatively fast transitions between the two.

Accordingly, we have modified (1) to allow different parameters on either side of a line $V = c_0 + c_1 R$, chosen by visual inspection of a three-dimensional plot. These are tied together by a logistic transition, resulting in:

$$\frac{dV}{dt} = \theta_1 + \theta_2 V + \theta_3 V^2 + \theta_4 V^3 + \theta_5 R + \theta_6 V R \quad (16)$$
$$\theta_i = p_i + \frac{q_i}{1 + \exp(c_3(V - c_1 + c_2 R))}. \quad (17)$$

Here the denominator in (17) controls the transition between parameters p_i and parameters $p_i + q_i$. The transition happens across the line

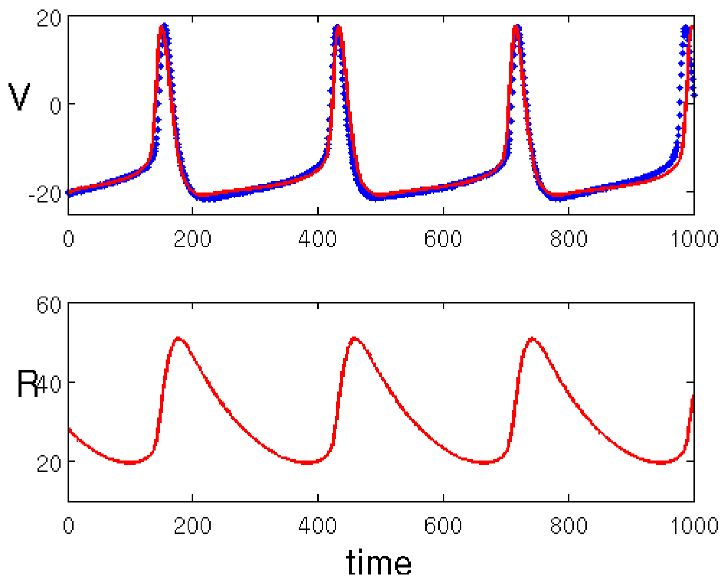

Figure 3: A least squares fit to the Rinzle equations estimated using the IPOPT constrained optimization routines.

$$V = c_1 + c_2 R$$

and c_3 controls the rate at which it occurs. These parameters were initially set naively via a least-squares fit between the estimated V, R and $g(t)$. The parameters p, q and r were then re-scaled to provide the same average period of the data using the Matlab function `ode45`. The resulting fit is plotted in Figure 3. Here we note that least squares fitting may lead to errors; the spikes in the data clearly do not occur at exactly equal intervals and this will tend to encourage wider peaks than desired. Although providing a good qualitative agreement with the data, these solutions are, in fact, worse in squared-error terms than those found for the original equations.

The modified model was re-estimated to data using the IPOPT routines and a certain amount of experimentation. Not all parameters could be estimated jointly within an evening of manually trying various sequences of fixing and unfixing parameters. The best results decreased squared error by more than a factor of 10 to 45.57. The resulting smooth is plotted in Figure 4, along with an estimated forcing function that demonstrates considerably less pattern. However, the resulting solutions represent unstable limit cycles that eventually decay; indicating that more analysis is needed to develop simple, tractable models.

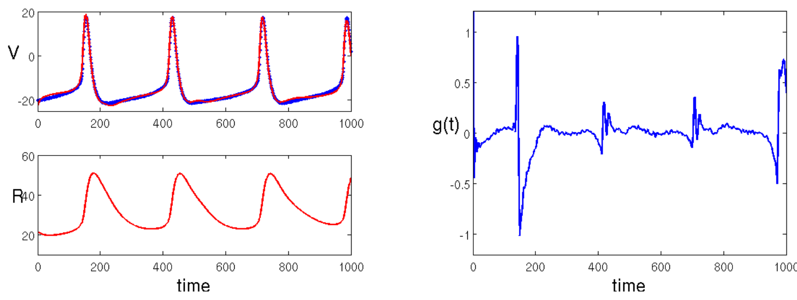

Figure 4: Left: modified Wilson equation fit with least squares. Right: lack of fit on the derivative scale for the modified Wilson equations; little pattern is evident.

6 Conclusion

We have examined the use of the IPOPT constrained nonlinear optimization routines for estimating parameters and performing diagnostics for a simple system of differential equations designed to describe the spiking behavior of zebra-fish motor-neurons. IPOPT provides an efficient means of performing this estimation; however, obtaining convergence can require some problem-specific fine-tuning. Particularly useful techniques inclued

- Imposing loose boundary conditions on the state variables. Although these are not active in the final solution, they help improve the conditioning of the system at intermediate steps.
- Successively increase the set of parameters to be optimized. This helps to keep the Hessian positive definite and provides a sequence of good initialization values for the next optimization problem.
- Consider transformations of the system. These have not been employed here, but the log transform

$$\frac{d}{dt}x = f(x), \;\; y = log(x) \Rightarrow \frac{d}{dt}y = e^{-y} f\left(e^{y}\right)$$

  is commonly found to make system trajectories easier to solve, particularly when they involve large values of $f(x)$.

Additionally, these routines proved highly efficient in providing diagnostics in the form of estimated forcing functions. The AMPL interface to IPOPT, described in the Appendix, allows parameter estimation, diagnostics and system modifications to be implemented in a straightforward manner.

Acknowledgements

The zebra fish data were kindly provided by Joe Fetcho's laboratory at the Biology Department of Cornell University.

References

Deuflhard, P. and F. Bornemann (2000). *Scientific Compuitng with Ordinary Differential Equations*. New York: Springer-Verlag.

Fourer, R., D. M. Gay, and B. W. Kernighan (2003). *AMPL: a Modeling Language for Mathematical Programming*. Pacific Grove: Brooks/Cole - Thomson Learning.

Hodgkin, A. L. and A. F. Huxley (1952). A quantitative descripotion of membrane current and its application to conduction and excitation in nerve. *J. Physiol. 133*, 444–479.

Hooker, G. (2007). Forcing function diagnostics for nonlinear dynamics. unpublished.

Huang, Y., D. Liu, and H. Wu (2006). Hierarchical bayesian methods for estimation of parameters in a longitudinal hiv dynamic system. *Biometrics*.

Ionides, E. L., C. Bretó, and A. A. King (2006). Inference for nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*.

Ramsay, J. ., G. Hooker, D. Campbell, and J. Cao (2007). Parameter estimation in differential equations: A generalized smoothing approach. *Journal of the Royal Statistical Society, Series B (with discussion)*.

Rinzel, J. and J. B. Keller (1973). Traveling wave solutions of a nerve conduction equation. *Biophysical Journal 13*(12), 13131337.

Wächter, A. and L. T. Biegler (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming* (106), 25–57.

Wilson, H. R. (1999). *Spikes, decisions and actions: the dynamical foundations of neuroscience*. Oxford: Oxford University Press.

A AMPL Model Code

The figures were fit using the AMPL interface to IPOPT. AMPL is mathematical modeling software available from

`http://www.ampl.com`.

A more complete description of AMPL is given in Fourer et al. 2003. The AMPL interface requires a model file to describe the parameters and relationships in the model; the model file for the initial parameter estimation problem is given in Appendix A.1. A data file defining constants and initialization values is given in Appendix A.2. Appendix A.3 provides AMPL output code to write the output of the model to a Matlab file for visualization. A set of commands for running this initial code, including the addition of constraints and sequentially unfixing parameters, is in Appendix A.4.

In addition to the code provided in the appendices, data files from the experiment are required. Further model estimation and diagnostics also require files defining the values of basis functions at the Gauss-Radau quadrature points. An archive providing all the files and code necessary to replicate the experiments described in this paper may be found at

`http://www.bscb.cornell.edu/~hooker`
or by e-mailing `giles.hooker@cornell.edu`.

A.1 Model File

```
#==================================================================
# wilsonfit.run
#
# This file provides the ampl commands to run parameter estimation
# and diagnostics on the Wilson equations.
#
# Giles Hooker, June 2007
#==================================================================

# First of all a model declaration

reset                                     ;  # Reset memory
option solver ipopt                       ;  # Select IPOPT as a solver
options ipopt_options "ma27_pivtol=1e-10" ;  # Change the tolerances


model wilsonfit.mod       ;   # Model declaration

data wilson_feas.dat      ;   # Read in feasible points
data zebrafish.dat        ;   # Read in observational data
data forcingbasis.dat     ;   # Basis expansion to estimate forcing functions
data scoefV.dat           ;   # Initial coefficients for the forcing functions
data spars.dat            ;   # Initial parameter estimates

data wilsonfit.dat        ;   # Initialize collocation matrix and variables


include exportbasis.inc   ;   # Gets the basis values into a matlab-readable data file
                              # this only needs to be run once

# We will put some constraints on the state space to help numerical
# stability
```

```
let rl :=  10 ;
let ru :=  50 ;
let vl := -50 ;
let vu :=  50 ;

# Now try to find a solution to the ODE at the current parameter values

fix p     ;
fix r     ;
fix y     ;
fix coefV ;
solve     ;

include wilsonfit1.inc ; # Export intial estimates to Matlab

# Sequentially unfix parameters to get an optimization.

unfix y[1]; unfix y[2]; solve;    # First let the initial conditions vary

unfix p[1]; unfix p[2]; solve;
unfix p[3]; unfix p[4]; solve;
unfix p[5]; unfix p[6]; solve;

unfix r[2]; solve;               # Only r2 is identifiable here

include wilsonfit2.inc;   # Export estimates with fitted parameters to Matlab

# Now fit a forcing function to the V component to try some diagnostics

fix p        ;
fix r        ;
fix y        ;
unfix coefV  ;
solve        ;

include wilsonfit3.inc;   # Export diagnostics to matlab


# ------------------- End of wilsonfit.run ---------------------- #
```

A.2 Data File

```
# ===================================================================
# dynamic optimization and diagnostics for the Wilson equations
# data declaration
# Giles Hooker, June 2007
# ===================================================================

# collocation matrix

param  a:        1                  2                  3 :=
      1  0.19681547722366   0.39442431473909   0.37640306270047
      2 -0.06553542585020   0.29207341166523   0.51248582618842
      3  0.02377097434822  -0.04154875212600   0.11111111111111;


# finite element parameters

let nfe       := 1000         ;
let ncp       := 3            ;

let time      :=  1           ;

let r1 := 0.15505102572168    ;
let r2 := 0.64494897427832    ;
let r3 := 1                   ;

# data points

let npts := 1001 ;
let nbase := 202 ;

# initial guesses for parameter values

for {i in parsp}{
    let p[i] := pstart[i];
}

for {i in parsr}{
    let r[i] := rstart[i];
}

for {i in parsy}{
    let y[i] := ystart[i];
}


# initial guesses of the decision variables

for {i in pts} {let  wvals[i] := 1 }         ;
for {i in base}
{
    let coefV[i] := ccoefV[i]                    ;
}


for {i in fe}
{
    for {j in cp}
    {
         let R[i,j]  := Rstartvals[i]        ;
         let V[i,j]  := Vstartvals[i]        ;
     }
let h[i] := tvals[i+1]-tvals[i]              ;
}


#-- end of the wilsonfit.dat file -
```

A.3 Output File

```
# =======================================================================
# diagnostics for the Wilson equations
# create results file for Matlab
#
# Giles Hooker, June 2007
# =======================================================================

# final parameter values

printf "% final co-efficient values" >>wilsonfit.m;
printf "\n\n" >>wilsonfit.m;

printf "coefV=[" >>wilsonfit.m;
printf {j in base}:
"%5.5f ",coefV[j]>>wilsonfit.m;
printf "]';\n\n" >>wilsonfit.m;

#display variables and parameters

printf "% display variables and parameters" >>wilsonfit.m;
printf "\n\n" >>wilsonfit.m;

printf "V3=[" >>wilsonfit.m;
printf "%5.5f ",y[1]>>wilsonfit.m;
printf {j in fe,k in cp}:
```

```
"%5.5f ",V[j,k]>>wilsonfit.m;
printf "]';\n\n" >>wilsonfit.m;

printf "R3=[" >>wilsonfit.m;
printf "%5.5f ",y[2]>>wilsonfit.m;
printf {j in fe,k in cp}:
"%5.5f ",R[j,k]>>wilsonfit.m;
printf "]';\n\n" >>wilsonfit.m;

printf "Vdot3=[" >>wilsonfit.m;
printf {j in fe,k in cp}:
"%5.5f ",Vdot[j,k]>>wilsonfit.m;
printf "]';\n\n" >>wilsonfit.m;

printf "Rdot3=[" >>wilsonfit.m;
printf {j in fe,k in cp}:
"%5.5f ",Rdot[j,k]>>wilsonfit.m;
printf "]';\n\n" >>wilsonfit.m;

# plot the results

printf "% plot the results" >>wilsonfit.m;
printf "\n\n" >>wilsonfit.m;

printf "figure(3)\n" >>wilsonfit.m;
printf "subplot(2,1,1)\n" >>wilsonfit.m;
printf "plot(time,V3,'r')\n" >>wilsonfit.m;
printf "hold on\n" >>wilsonfit.m;
printf "plot(tvals,Vobs,'.')\n" >>wilsonfit.m;
printf "hold off\n" >>wilsonfit.m;
printf "xlabel ('time')">>wilsonfit.m; printf ";\n">>wilsonfit.m;
printf "ylabel ('V')">>wilsonfit.m; printf ";\n">>wilsonfit.m;
printf "subplot(2,1,2)\n" >>wilsonfit.m;
printf "plot(time,R3,'r')\n" >>wilsonfit.m;
printf "xlabel ('time')">>wilsonfit.m; printf ";\n">>wilsonfit.m;
printf "ylabel ('R')">>wilsonfit.m; printf ";\n">>wilsonfit.m;
printf "\n" >>wilsonfit.m;

# Set up diagnostic plots based on forcing functions

printf "% Now lets look at forcing diagnostics" >>wilsonfit.m;
printf "\n\n" >>wilsonfit.m;

printf "% First of all, we need to find the values of the forcing functions" >>wilsonfit.m;
printf "\n\n" >>wilsonfit.m;
printf "% Load in basis values at the collocation points" >>wilsonfit.m;
printf "\n\n" >>wilsonfit.m;

printf "bvals = load('basisvals.dat');\n" >>wilsonfit.m;
printf "\n" >>wilsonfit.m;

printf "% Now turn these into forcing functions" >>wilsonfit.m;
printf "\n\n" >>wilsonfit.m;

printf "V_forcing = bvals*coefV;\n" >>wilsonfit.m;
printf "\n" >>wilsonfit.m;

printf "% Plot them" >>wilsonfit.m;
printf "\n\n" >>wilsonfit.m;

printf "figure(4)\n" >>wilsonfit.m;
printf "plot(time,V_forcing)\n" >>wilsonfit.m;
printf "\n" >>wilsonfit.m;

printf "% Now a diagnostic plot" >>wilsonfit.m;
printf "\n\n" >>wilsonfit.m;

printf "figure(5)\n" >>wilsonfit.m;
printf "plot3(V3,R3,V_forcing)\n" >>wilsonfit.m;
printf "xlabel('V','fontsize',20)\n" >>wilsonfit.m;
printf "ylabel('R','fontsize',20)\n" >>wilsonfit.m;
printf "title('Diagnostics for V forcing','fontsize',20)\n" >>wilsonfit.m;
printf "\n" >> wilsonfit.m;

printf "% End of wilsonfit3" >>wilsonfit.m;
printf "\n\n\n" >>wilsonfit.m;

close wilsonfit.m;
printf "output written to matlab m-file: wilsonfit.m ...\n";

#-- end of the wilsonfit3.inc file -
```

A.4 Running Code

```
#=================================================================
# wilsonfit.run
#
# This file provides the ampl commands to run parameter estimation
# and diagnostics on the Wilson equations.
#
# Giles Hooker, June 2007
```

```
#=================================================================

# First of all a model declaration

reset                                     ;  # Reset memory
option solver ipopt                       ;  # Select IPOPT as a solver
options ipopt_options "ma27_pivtol=1e-10" ;  # Change the tolerances


model wilsonfit.mod       ;   # Model declaration

data wilson_feas.dat      ;   # Read in feasible points
data zebrafish.dat        ;   # Read in observational data
data forcingbasis.dat     ;   # Basis expansion to estimate forcing functions
data scoefV.dat           ;   # Initial coefficients for the forcing functions
data spars.dat            ;   # Initial parameter estimates

data wilsonfit.dat        ;   # Initialize collocation matrix and variables


include exportbasis.inc   ;   # Gets the basis values into a matlab-readable data file
                              # this only needs to be run once

# We will put some constraints on the state space to help numerical
# stability

let rl :=  10 ;
let ru :=  50 ;
let vl := -50 ;
let vu :=  50 ;

# Now try to find a solution to the ODE at the current parameter values

fix p     ;
fix r     ;
fix y     ;
fix coefV ;
solve     ;

# Sequentially unfix parameters to get an optimization.

unfix y[1]; unfix y[2]; solve;    # First let the initial conditions vary

unfix p[1]; unfix p[2]; solve;
unfix p[3]; unfix p[4]; solve;
unfix p[5]; unfix p[6]; solve;
unfix r[2]; solve;               # Only r2 is identifiable here


# Now fit a forcing function to the V component to try some diagnostics

fix p        ;
fix r        ;
fix y        ;
unfix coefV  ;
solve        ;

include wilsonfit3.inc;   # Export diagnostics to matlab


# ------------------- End of wilsonfit.run ---------------------- #
```