Parameter Estimation of the Morris–Lecar Model from Noisy Data

Gefei Zhang

2025-07-05

Contents

1 Introduction

Epilepsy is a common neurological disorder characterized by recurrent, unprovoked seizures due to abnormal synchronous neuronal activity (S. 2010). A key challenge in epilepsy management is that nearly 30% of patients have medically refractory epilepsy, where seizures persist despite antiepileptic drugs (AEDs) (Remy S 2006). Understanding mechanisms behind drug-resistant seizures remains central to epilepsy research. Computational modeling provides a valuable approach to explore epileptiform dynamics, connecting experimental, clinical, and theoretical insights.

Among computational models, the Morris-Lecar (ML) model is widely used for studying neuronal excitability. Based on simplified biophysics, it describes membrane potential and ionic currents via a system of ordinary differential equations (ODEs) (Morris C 1981). Its computational efficiency and ability to reproduce key neuronal behaviors—such as spiking, bursting, and transitions between quiescent and active states—make it particularly useful for modeling epileptic seizures (Stefanescu and Talathi 2012). For example, it has been applied to simulate seizure-like activity and investigate how ion channel dysregulation may lower seizure thresholds (Stefanescu and Talathi 2012). However, accurate parameterization is critical, as the model's predictions depend sensitively on its biophysical parameters.

A central difficulty in applying the ML model is the challenge of estimating parameters from noisy experimental data. Electrophysiological recordings are typically sparse, noisy, and available only at discrete time points, complicating direct parameter estimation. The conventional method, maximum likelihood estimation (MLE), aims to optimize the probability of observed data given the model by solving a nonlinear least squares problem. However, in high-dimensional ODE models, the likelihood surface is often irregular due to non-convexity and parameter correlations, making optimization sensitive to initial values and prone to local minima (Remy S 2006). These issues hinder reliable inference in complex systems like the ML model.

To overcome these challenges, this paper explores functional data analysis (FDA) as an alternative for estimating ML model parameters. FDA focuses on data that vary over continuous domains (e.g., time series) and provides tools for smoothing noisy data and leveraging functional derivatives in parameter optimization (B. W. Ramsay J. O. Silverman 2005; Ramsay J. O. Hooker G. and S. 2009). By treating observations as smooth functional processes, FDA can help regularize estimation, improve parameter identifiability, and mitigate sensitivity to noise.

This paper proceeds as follows: We first review the Morris-Lecar model and outline the MLE and parameter cascading methods. We then describe our simulation design to generate noisy synthetic data mimicking experimental recordings. Finally, we compare MLE and FDA-based approaches for parameter estimation, emphasizing their differences in handling noise and their relative performance.

2 Background

2.1 Ordinary Differential Equation

Solutions to ordinary differential equations (ODEs) fall into analytical and numerical approaches. Analytical methods derive explicit formulas using techniques like separation of variables, integrating factors, and Laplace transforms but are often infeasible for complex systems. Numerical methods approximate solutions at discrete points and include schemes like forward and backward Euler, essential for nonlinear systems lacking closed-form solutions. In this project, we use R's `deSolve` package and the `lsoda()` solver, which adaptively handles stiff and non-stiff systems (Soetaert K. and W. 2010), enabling efficient simulation of the Morris-Lecar model even with oscillatory or rapidly changing dynamics.

2.2 The Morris–Lecar Model

The Morris–Lecar model is a two-dimensional system of nonlinear ordinary differential equations originally developed to describe the membrane voltage dynamics in barnacle muscle fibers. Despite its simplicity compared to full conductance-based models like Hodgkin–Huxley, it captures a wide range of excitability types and serves as a useful tool for understanding neuronal spiking and bursting behavior.

The Morris-Lecar model is a two-dimensional nonlinear system used to describe the electrical activity of excitable cells, particularly neurons. It captures the dynamics of membrane voltage v and a recovery variable w, representing the gating of potassium ion channels. Calcium channel activation is assumed to follow its steady-state value, reducing the full biophysical model to a pair of coupled ODEs.

$$\frac{dv}{dt} = \frac{1}{20}\left[90 - g_{Ca} \cdot m_\infty(v) \cdot (v - 120) - 8w(v + 84) - 2(v + 60)\right]$$

$$\frac{dw}{dt} = \phi \cdot (w_\infty(v) - w) \cdot \cosh\left(\frac{v - 2}{60}\right)$$

Here, $m_\infty(v)$ and $w_\infty(v)$ are steady-state activation functions:

$$m_\infty(v) = \frac{1}{2}\left(1 + \tanh\left(\frac{v + 1.2}{18}\right)\right), \quad w_\infty(v) = \frac{1}{2}\left(1 + \tanh\left(\frac{v - 2}{30}\right)\right)$$

Using the custom-defined `morrislecar` function, we first generate phase portraits and nullclines for the system. The phase plane provides a geometric representation of how the state (v, w) evolves over time. With phasearrows() and `nullclines()`, we overlay vector fields and the loci where $\frac{dv}{dt} = 0$ and $\frac{dw}{dt} = 0$. respectively. This helps identify equilibrium points and analyze local behavior.

For parameters $(g_{Ca}, \phi) = (4.4, 0.04)$, we find a single stable fixed point. Solving numerically with `lsoda()` from the `deSolve` package, we simulate two trajectories starting from nearly identical initial conditions: $(-26, 0.1135)$ and $(-26, 0.1135)$ (Soetaert K. and W. 2010). Both converge to the same stable equilibrium, as evidenced by negative eigenvalues computed using `newton()`. This behavior is associated with persistent neural activity but not seizure-like activity. In neuroscience, such steady, confined patterns of firing may resemble localized post-stimulus responses often seen in epilepsy models, without generalization to full-blown seizures (Kerkhoff 2017).

```
MorrisLecar <- function(t, state, parameters) {
  gca <- parameters[1]; phi <- parameters[2]
  v <- state[1]; w <- state[2]
  m_inf <- 0.5 * (1 + tanh((v + 1.2) / 18))
  w_inf <- 0.5 * (1 + tanh((v - 2) / 30))
```

```
  tau_w <- 1 / cosh((v - 2) / 60)
  dv <- (1 / 20) * (90 - gca * m_inf * (v - 120) - 8 * w * (v + 84) - 2 * (v + 60))
  dw <- phi * (w_inf - w) / tau_w
  return(list(c(dv, dw)))
}
```

```
morrislecar=function(v,w,parms) {
  gca=parms[1]; phi=parms[2];
  c((1/20)* (90-gca*0.5*(1+tanh((v+1.2)/18))*(v-120)-8*w*(v+84)-2*(v+60)),
    phi*(0.5*(1+tanh((v-2)/30))-w)*cosh((v-2)/60))
}
```

```
#blank plot to add vectors and clines
plot(1,xlim=c(-60,40), ylim=c(0,0.6), type='n',xlab="v",ylab="w")

# gca = 4.4, phi = 0.04
phasearrows(fun=morrislecar,xlim=c(-60,40),ylim=c(0,0.6),resol=25,parms=c(4.4,0.04), add=T)
```

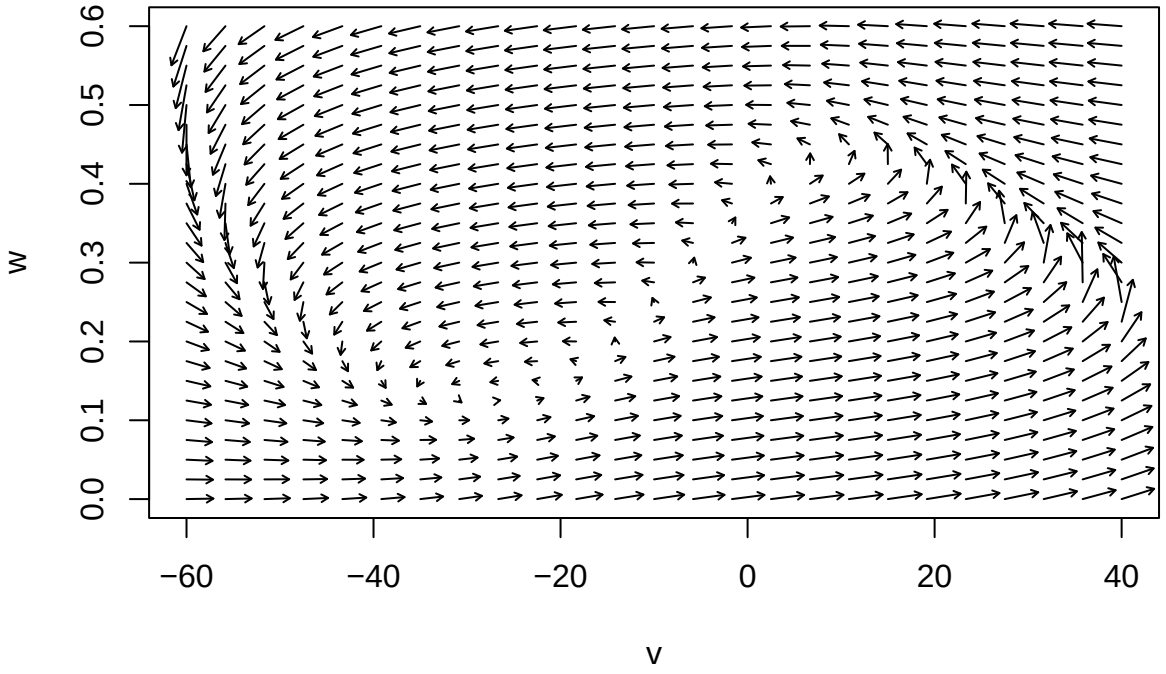

Figure 1: Phase-plane diagram of the Morris-Lecar model showing vector field arrows.

```
plot(1,xlim=c(-60,40), ylim=c(0,0.6), type='n',xlab="v",ylab="w")
nullclines(fun=morrislecar,xlim=c(-60,40),ylim=c(0,0.6),resol=250,parms=c(4.4,0.04), add=T)
newton(MorrisLecar,x0=c(-30,0.1),parms=c(4.4,0.04))
```

```
## 1 -26.53938 0.1275774
## 2 -26.59739 0.1293757
## 3 -26.59687 0.1293793
## Fixed point (x,y) =  -26.59687 0.1293793
## Jacobian Df=
```

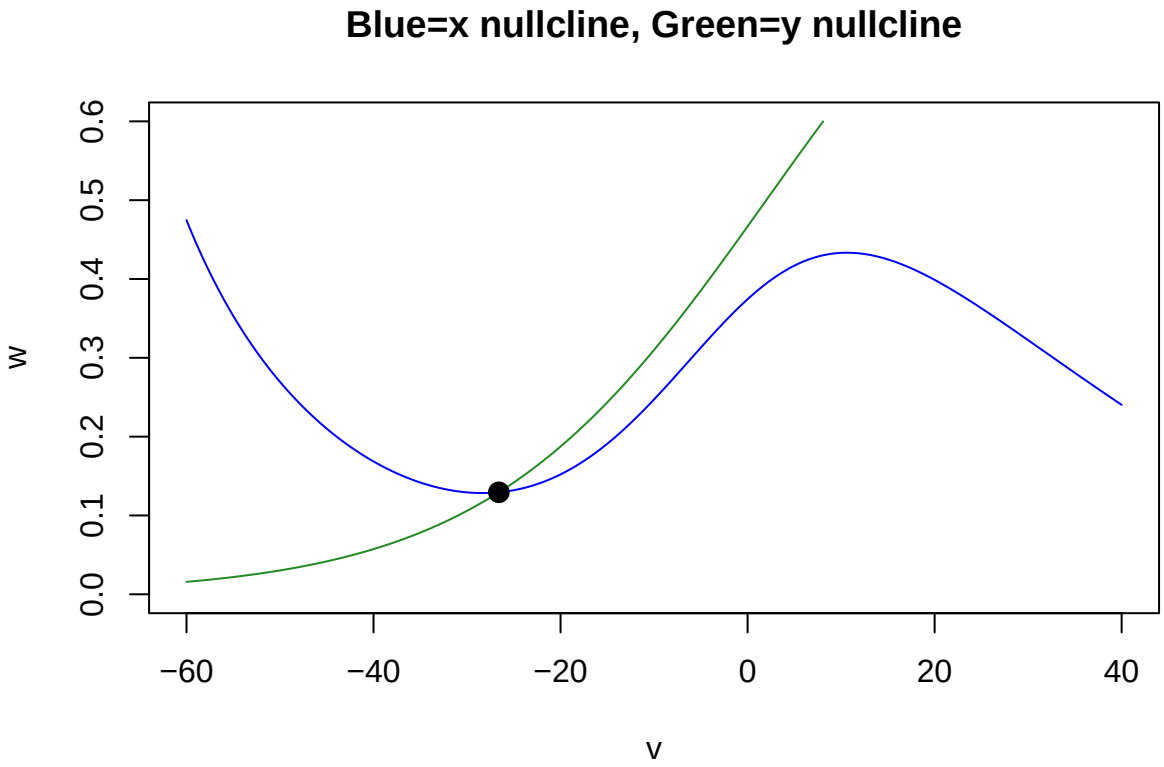

Figure 2: Phase-plane diagram of the Morris-Lecar model displaying nullclines for v and w.

```
##              [,1]         [,2]
## [1,] 0.0258199730 -22.96125321
## [2,] 0.0003351416  -0.04462988
## Eigenvalues
## [1] -0.00940496+0.08033975i -0.00940496-0.08033975i

## $x
## [1] -26.5968670   0.1293793
##
## $df
##              [,1]         [,2]
## [1,] 0.0258199730 -22.96125321
## [2,] 0.0003351416  -0.04462988
```

```
# solve the Morris Lecar model, given initial conditions (-60,40) using lsoda
times<-seq(0,300,0.2)
ML_1<-lsoda(y=c(-26,0.1135),times=times, MorrisLecar, parms=c(4.4,0.04))
ML_2<-lsoda(y=c(-26,0.1134),times=times, MorrisLecar, parms=c(4.4,0.04))
```

```
# plot in phase space x=voltage, y=w
plot(1,xlim=c(-60,40), ylim=c(0,0.6), type='n',xlab="v",ylab="w")
nullclines(fun=morrislecar,xlim=c(-60,40),ylim=c(0,0.6),resol=250,parms=c(4.4,0.04), add=T)
lines(ML_2[,2], ML_2[,3], xlim=c(-60,40),ylim=c(0.1,0.6), type="l", col="black", add=T)
lines(ML_1[,2], ML_1[,3], xlim=c(-60,40),ylim=c(0.1,0.6), type="l", col="red", add=T)
```

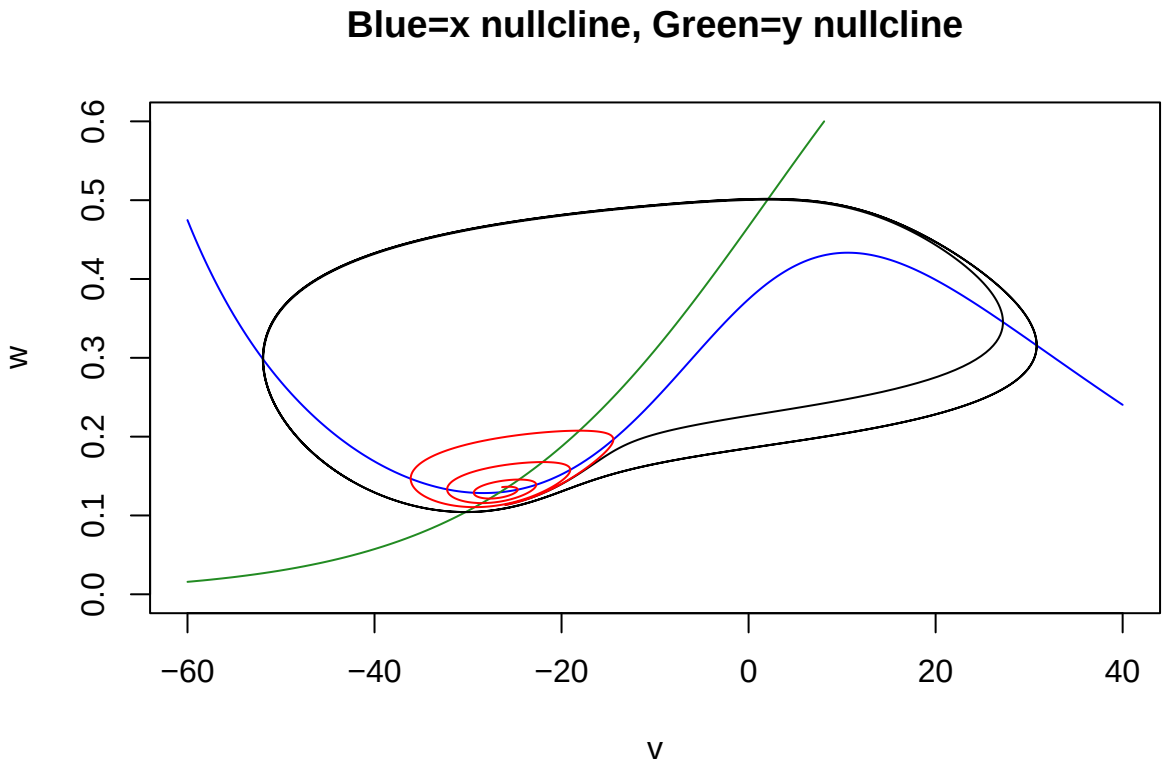

Figure 3: Phase-plane trajectories for two nearly identical initial conditions, w = 0.1134 (black) and w = 0.1135 (red), starting from v = -26 mV.

```
plot(ML_2[,1],ML_2[,2], type="l", xlab="time", ylab="v", col="blue")
par(new=TRUE)
plot(ML_2[,1],ML_2[,3], xlab="", ylab="", type="l", axes=FALSE, col="forestgreen")
axis(side=4)
mtext(side=4, line=2.5, 'w')
```

For parameters $(g_{Ca}, \phi) = (5.5, 0.22)$, The phase plane now shows three equilibrium points (Kerkhoff 2017):

- A stable focus at $(-21.1, 0.177)$, attracting nearby trajectories;
- A saddle point at $(-11.5, 0.289)$, which repels along one eigen-direction and attracts along another;
- An unstable node at $(2.97, 0.516)$, where trajectories diverge.

This arrangement suggests multistability and sensitivity to initial conditions. To illustrate this, we simulate two trajectories with slightly different initial voltages: $(-12, 0.3)$ and $(-10, 0.3)$. The first (black line) is attracted to the stable focus and settles into steady activity, while the second (red line) escapes to higher voltage states—indicative of a transition into seizure-like behavior. This aligns with experimental and theoretical observations: small perturbations in initial membrane potential can push the system over a bifurcation threshold, leading to qualitatively different neural outcomes.

```
# gca = 5.5, phi = 0.22
plot(1,xlim=c(-60,40), ylim=c(0,0.6), type='n',xlab="v",ylab="w")
nullclines(fun=morrislecar,xlim=c(-60,40),ylim=c(0,0.6),resol=250,parms=c(5.5,0.22), add=T)
newton(MorrisLecar,x0=c(-30,0.1),parms=c(5.5,0.22))
```

```
## 1 -22.70985 0.1515668
```

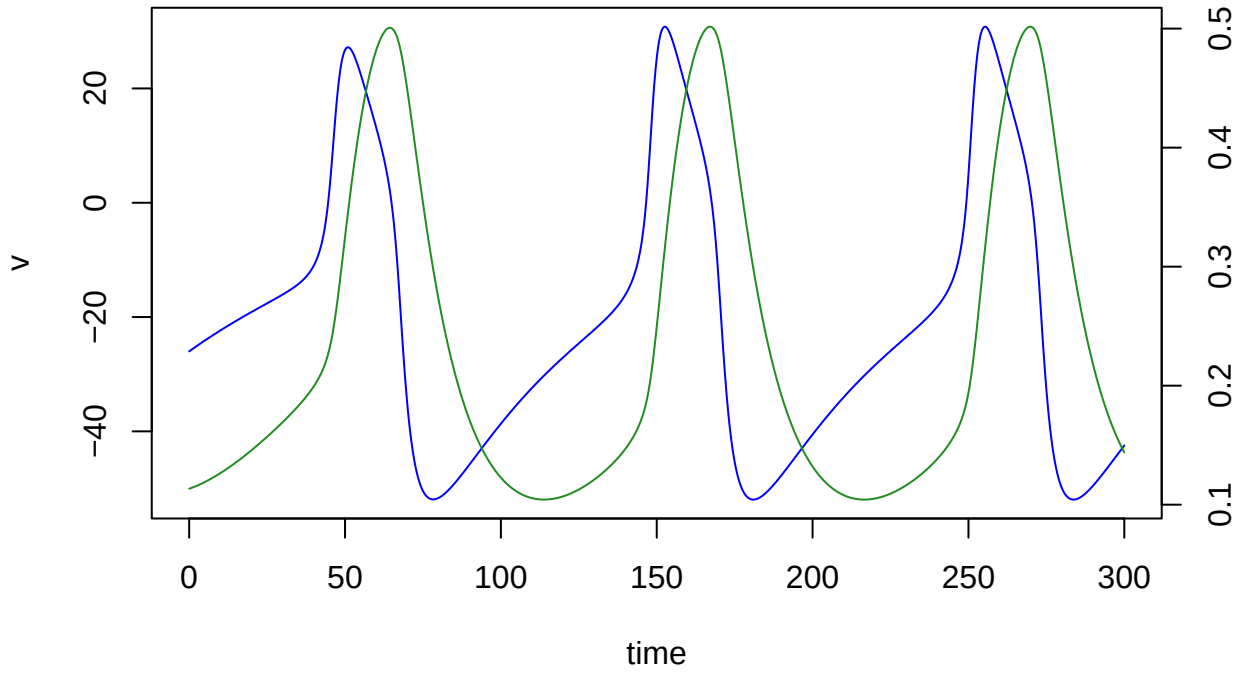

Figure 4: Time evolution of membrane voltage v(t) (blue) and recovery variable w(t) (green) for initial condition (-26, 0.1134).

```
## 2 -21.14534 0.1754925
## 3 -21.09309 0.1766014
## 4 -21.09315 0.1766017
## Fixed point (x,y) =  -21.09315 0.1766017
## Jacobian Df=
##            [,1]        [,2]
## [1,] 0.18612985 -25.1627403
## [2,] 0.00229266  -0.2364972
## Eigenvalues
## [1] -0.0251837+0.1141761i -0.0251837-0.1141761i

## $x
## [1] -21.0931494   0.1766017
##
## $df
##            [,1]        [,2]
## [1,] 0.18612985 -25.1627403
## [2,] 0.00229266  -0.2364972
```

```
newton(MorrisLecar,x0=c(4,0.5),parms=c(5.5,0.22))
```

```
## 1 3.028265 0.5171423
## 2 2.975248 0.5162485
## 3 2.97492 0.5162429
## Fixed point (x,y) =  2.97492 0.5162429
## Jacobian Df=
##             [,1]       [,2]
```

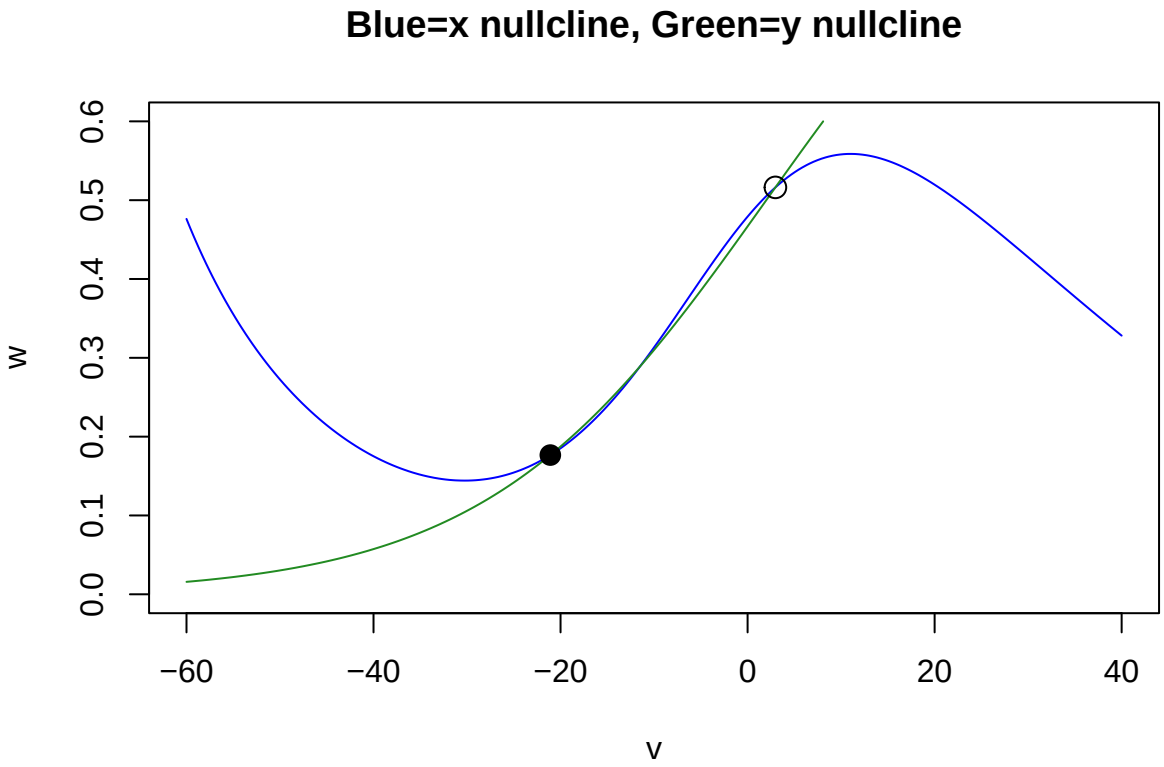

Figure 5: Phase-plane diagram of the Morris-Lecar model showing nullclines and equilibrium points.

```
## [1,] 0.372195451 -34.789968
## [2,] 0.003663281  -0.220029
## Eigenvalues
## [1] 0.0760832+0.1994065i 0.0760832-0.1994065i
```

```
## $x
## [1] 2.9749200 0.5162429
##
## $df
##              [,1]       [,2]
## [1,] 0.372195451 -34.789968
## [2,] 0.003663281  -0.220029
```

```
# solve the Morris Lecar model, given initial conditions (-12,0.3) using lsoda
times<-seq(0,300,0.2)
outMLa1<-lsoda(y=c(-12,0.3),times=times, MorrisLecar, parms=c(5.5,0.22))
outMLa2<-lsoda(y=c(-10,0.3),times=times, MorrisLecar, parms=c(5.5,0.22))
```

```
# plot in phase space x=voltage, y=w
plot(1,xlim=c(-60,40), ylim=c(0,0.7), type='n',xlab="v",ylab="w")
nullclines(fun=morrislecar,xlim=c(-60,40),ylim=c(0,0.7),resol=250,parms=c(5.5,0.22), add=T)
lines(outMLa1[,2], outMLa1[,3], xlim=c(-60,40),ylim=c(0.1,0.6), type="l", col="black", add=T)
lines(outMLa2[,2], outMLa2[,3], xlim=c(-60,40),ylim=c(0.1,0.6), type="l", col="red", add=T)
```

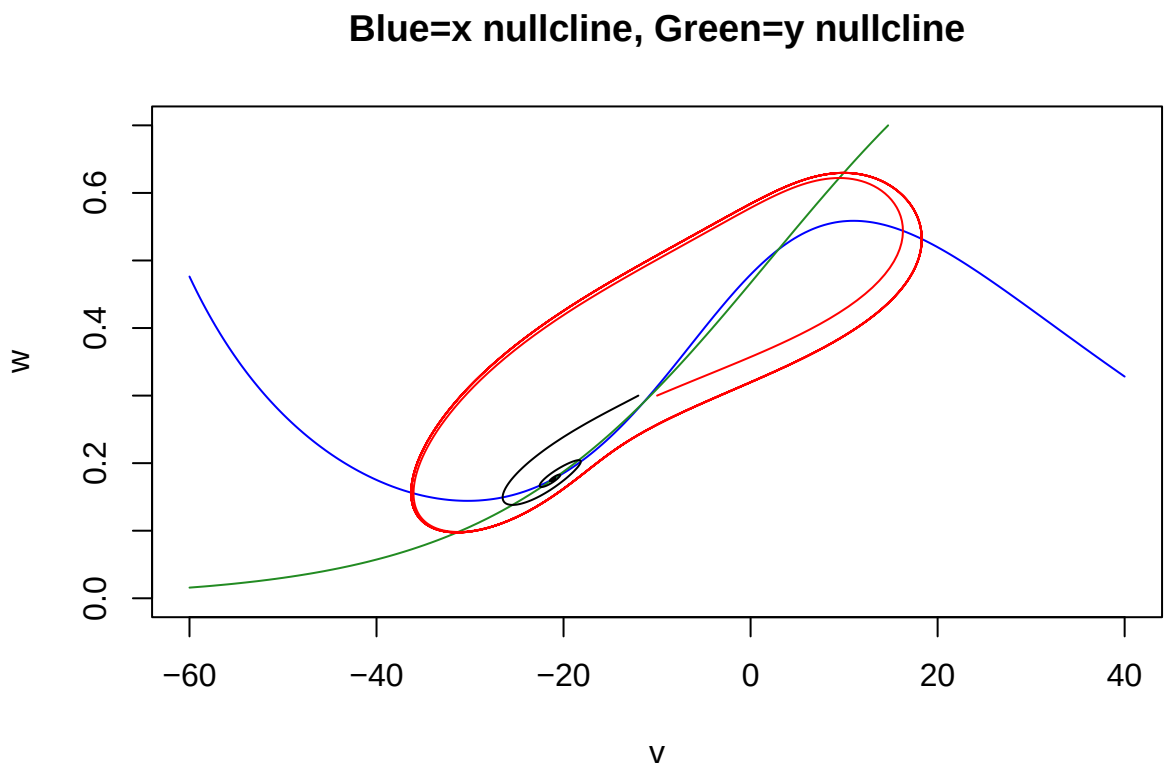

Figure 6: Phase-plane trajectories of the Morris-Lecar model for two different initial conditions.

```
plot(outMLa1[,1],outMLa1[,3], ylim=c(0,0.8), type="l", xlab="time", ylab="w", col="black")
lines(outMLa1[,1],outMLa2[,3], col="red")
```

2.3 Maximum likelihood

Maximum likelihood estimation (MLE) is a classical approach for fitting models to data by maximizing the likelihood—the probability of observing the data given model parameters. For ODEs, the likelihood assumes observation noise and compares simulated system trajectories to observed data (Phillips 2015). MLE involves three main steps:

- **Obtain or Simulate Data**: Either collect empirical data or simulate synthetic data from a known model;
- **Loss Function**: Quantify discrepancies between predicted and observed values; lower deviance indicates better fit;
- **Optimization**: Find parameters minimizing the loss (or equivalently maximizing the log-likelihood) using numerical optimizers such as R's `optim()`.

2.4 FDA cascading parameter approach

The **pCODE** package offers an alternative framework for ODE parameter estimation based on **functional data analysis (FDA)**, specifically utilizing a **parameter cascading method** (Wang and Cao 2023). This nested optimization framework estimates both smooth state trajectories and structural parameters simultaneously, with a smoothing penalty λ regulating their interaction.

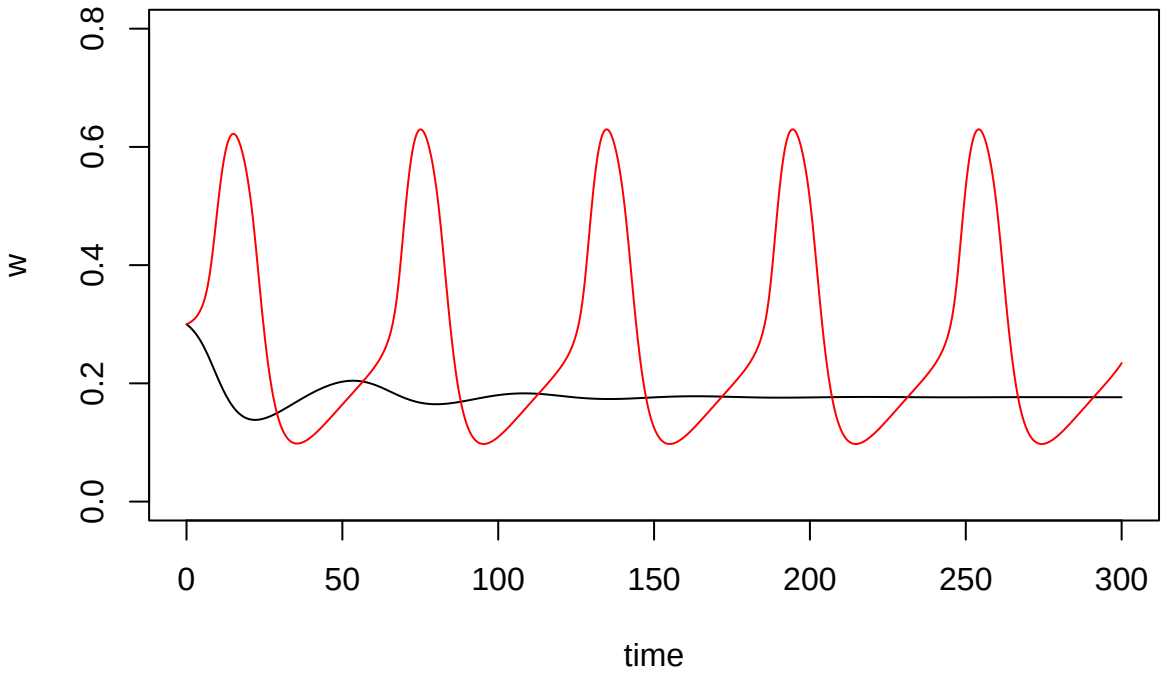

Figure 7: Time evolution of the recovery variable w(t) for two initial conditions. The black curve shows the trajectory starting from (-12, 0.3) and the red curve from (-10, 0.3).

2.4.1 Lower Stream: Estimating Smooth Trajectories

The lower stream focuses on recovering smooth estimates of the latent state variables from noisy measurements. Each state trajectory $x_i(t)$ is represented as a linear combination of B-spline basis functions:

$$x_i(t) = \sum_{k=1}^{K_i} c_{ik}\,\phi_{ik}(t) = \mathbf{c}_i^\top \phi_i(t)$$

where $\phi_{ik}(t)$ is the k-th basis function, and c_{ik} is its coefficient. The coefficients $\mathbf{c}_i$ are determined by minimizing an inner-level objective function:

$$J(\mathbf{c} \mid \theta) = \sum_{i=1}^{I} \left[-\ln g(\mathbf{e}_i) + \lambda_i \int \left(\dot{x}_i(t) - f_i(\mathbf{x}, t \mid \theta)\right)^2 dt \right]$$

The first term $-\ln g(\mathbf{e}_i)$ captures the log-likelihood of the residuals $\mathbf{e}_i$, and the second term penalizes the deviation between the estimated derivatives $\dot{x}_i(t)$ and the dynamics prescribed by the ODE system $f_i(\cdot)$. The scalar λ_i controls the trade-off between data fidelity and model adherence. This smoothing spline formulation enables accurate recovery of the latent trajectory under noisy conditions.

2.4.2 Upper Stream: Estimating Structural ODE Parameters

The upper stream addresses the estimation of the structural parameters θ (e.g., reaction rates, conductances). For a given set of structural parameters, the lower-level optimization returns the optimal basis coefficients $\mathbf{c}_i^*(\theta)$, yielding fitted state trajectories

$$\hat{x}_i(t \mid \theta) = \mathbf{c}_i^*(\theta)^\top \phi_i(t) \quad , \quad \hat{\mathbf{e}}_i = \mathbf{y}_i - \hat{x}_i(t_i \mid \theta)$$

The outer objective function—depending on the distributional assumption of the error—is defined as:

$$H(\theta) = -\sum_{i=1}^{I} \ln g(\hat{\mathbf{e}}_i \mid \theta)$$

In the case of i.i.d. normal noise, this simplifies to a nonlinear least squares (NLS) problem. When non-Gaussian errors are involved, a custom log-likelihood function $g(\cdot)$ can be supplied.

3 Simulation Design

3.1 Generation of Noisy Observations

To investigate the performance of parameter estimation under various noise conditions, we implement the add_noise_experiment function. This function adds Gaussian noise to the true state trajectories, independently for v and w, across a predefined grid of noise standard deviations (`v_sd_seq` and `w_sd_seq`). For each noise configuration, we repeat the perturbation process 10 times with different seeds to generate replicates. The result is a nested list structure containing simulated observations of v and w for each combination of noise level and trial.

Four noise settings are tested:

- (10, 0.001): High noise on v, low noise on w;
- (0.1, 0.001): Low noise on both;
- (10, 0.1): High noise on both;
- (0.1, 0.1): Low noise on v, moderate noise on w

This design allows us to disentangle the effect of noise on each state variable.

```
gca_true <- 4.4
phi_true <- 0.04
model.par <- c(gca = gca_true, phi = phi_true)
y0 <- c(v = -26, w = 0.1135)
times <- seq(0, 300, 0.2)
out_true <- lsoda(y = y0, times = times, func = MorrisLecar, parms = model.par)
true.v <- out_true[, 2]
true.w <- out_true[, 3]
```

```
v_sd_seq <- c(10,0.1,10,0.1)
w_sd_seq <- c(0.001,0.001,0.1,0.1)

add_noise_experiment <- function(true.v, true.w, v_sd_seq, w_sd_seq, n_repeat = 10, seed_base = 1) {
  results <- list()

  for (k in seq_along(v_sd_seq)) {
    v_sd <- v_sd_seq[k]
    w_sd <- w_sd_seq[k]
    repeat_results <- list()

    for (i in 1:n_repeat) {
      set.seed(seed_base + i)
      v.noisy <- true.v + rnorm(length(true.v), mean = 0, sd = v_sd)
      w.noisy <- true.w + rnorm(length(true.w), mean = 0, sd = w_sd)
      repeat_results[[i]] <- list(v = v.noisy, w = w.noisy)
    }

    results[[k]] <- list(v_sd = v_sd, w_sd = w_sd, repeats = repeat_results)
  }

  return(results)
}
noisy_data_list <- add_noise_experiment(true.v, true.w, v_sd_seq, w_sd_seq, n_repeat = 10)
```

3.2 Maximum Likelihood Estimation for the Morris–Lecar Model

We implement Maximum Likeliohood Estimation to infer the two structural parameters g_{Ca} and ϕ of the Morris–Lecar model using synthetic data generated with additive Gaussian noise.

$$\frac{dv}{dt} = \frac{1}{20}\left[90 - g_{Ca} \cdot m_\infty(v) \cdot (v - 120) - 8w(v + 84) - 2(v + 60)\right]$$

$$\frac{dw}{dt} = \phi \cdot \frac{w_\infty(v) - w}{\tau_w(v)} \quad \text{where} \quad \tau_w(v) = \frac{1}{\cosh\left(\frac{v-2}{60}\right)}$$

Here, $m_\infty(v)$ and $w_\infty(v)$ are steady-state activation functions:

$$m_\infty(v) = \frac{1}{2}\left(1 + \tanh\left(\frac{v + 1.2}{18}\right)\right), \quad w_\infty(v) = \frac{1}{2}\left(1 + \tanh\left(\frac{v - 2}{30}\right)\right)$$

3.2.1 Step 1: Define the Model

We first define the Morris–Lecar system as an R function compatible with `deSolve::lsoda`. This function computes the rate of change for membrane voltage v and recovery variable w, given the parameters g_{Ca} and ϕ, based on voltage-dependent activation functions. The model equations are:

```
MorrisLecar <- function(t, state, parameters) {
  gca <- parameters[1]; phi <- parameters[2]
  v <- state[1]; w <- state[2]
  m_inf <- 0.5 * (1 + tanh((v + 1.2) / 18))
  w_inf <- 0.5 * (1 + tanh((v - 2) / 30))
  tau_w <- 1 / cosh((v - 2) / 60)
  dv <- (1 / 20) * (90 - gca * m_inf * (v - 120) - 8 * w * (v + 84) - 2 * (v + 60))
  dw <- phi * (w_inf - w) / tau_w
  return(list(c(dv, dw)))
}
```

3.2.2 Step 2: Loss function

The core of the ML framework is the loss function. A loss function receives model parameters and observed data as input and produces a numerical value that reflects how well the model, under those parameters, aligns with the data. In our implementation, the loss is the negative log-likelihood assuming independent Gaussian noise on v and w. For each candidate parameter set, the function simulates the system dynamics using `lsoda()` from the `deSolve` package over a time interval from 0 to 300 with a time step of 0.2, starting from fixed initial conditions.

```
ml.loss.gen <- function(par, v.obs, w.obs) {
    gca <- par[1]; phi <- par[2]
    err.v <- par[3]; err.w <- par[4]
    if (err.v <= 0 || err.w <= 0) return(1e10)
    out_true <- lsoda(y = y0, times = times,
                      func = MorrisLecar, parms = c(4.4, 0.04))
    true.v <- out_true[, 2]
    true.w <- out_true[, 3]
    loglik.v <- dnorm(v.obs, mean = true.v, sd = err.v, log = TRUE)
    loglik.w <- dnorm(w.obs, mean = true.w, sd = err.w, log = TRUE)
```

```
    deviance <- -2 * sum(loglik.v + loglik.w)
    return(deviance)
}
```

3.2.3 Step 3: Parameter estimation via optimization:

We used the `optim()` function to minimize the loss function and recover the parameters. In each run, we jointly estimated the values of g_{Ca} and ϕ, and the standard deviations of the noise for v and w. To assess stability and variability, we repeated the estimation process 10 times using different random noise realizations. The estimated parameters from each run were stored and summarized to evaluate the method's robustness.

```
mle_results <- list()

for (k in seq_along(noisy_data_list)) {
  v_sd <- noisy_data_list[[k]]$v_sd
  w_sd <- noisy_data_list[[k]]$w_sd
  repeat_data <- noisy_data_list[[k]]$repeats

  result_mat <- matrix(NA, nrow = length(repeat_data), ncol = 6)
  colnames(result_mat) <- c("v_sd", "w_sd", "gca", "phi", "err.v", "err.w")

  for (i in seq_along(repeat_data)) {
    v.noisy <- repeat_data[[i]]$v
    w.noisy <- repeat_data[[i]]$w

    fit <- optim(par = c(4.4, 0.04, v_sd, w_sd),
                 fn = ml.loss.gen,
                 w.obs = w.noisy,
                 v.obs = v.noisy)

    result_mat[i, ] <- c(v_sd, w_sd, fit$par)
  }

  mle_results[[k]] <- as.data.frame(result_mat)
}

mle_df <- do.call(rbind, mle_results)
```

3.3 Simulation and Parameter Estimation of a Noisy Morris-Lecar Model Using pCODE

We constructed a common B-spline basis for both dimensions with order 4 (i.e., cubic splines), over the domain $[0, 300]$, using 10 equally spaced interior knots. This basis representation allowed us to smooth the noisy data before estimation. The basis list was passed to the `pcode` function, which estimates ODE parameters by minimizing a penalized criterion that balances data fidelity with model adherence. The regularization penalty is controlled by the parameter λ, which we varied across two values 0.01 and 1, representing low and moderate smoothing strengths.

For each combination of noise level and lambda, we applied `pcode` to the smoothed observations, specifying the known ODE system, initial values, and the parameter names. The pcode function then produced structural parameter estimates for each repetition.

```r
# B-spline basis
knots <- seq(0, 300, length.out = 10)
norder <- 4
nbasis <- length(knots) + norder - 2
basis <- create.bspline.basis(rangeval = c(0, 300), nbasis = nbasis, norder = norder, breaks = knots)
basis.list <- list(basis, basis)

# pcode
lambda_vec <- c(0.01, 1)
pcode_results_all <- list()
```

```r
add_noise_experiment <- function(true.v, true.w, v_sd_seq, w_sd_seq, n_repeat = 10, seed_base = 1) {
  results <- list()

  for (k in seq_along(v_sd_seq)) {
    v_sd <- v_sd_seq[k]
    w_sd <- w_sd_seq[k]
    repeat_results <- list()

    for (i in 1:n_repeat) {
      set.seed(seed_base + i)
      v.noisy <- true.v + rnorm(length(true.v), mean = 0, sd = v_sd)
      w.noisy <- true.w + rnorm(length(true.w), mean = 0, sd = w_sd)
      repeat_results[[i]] <- list(v = v.noisy, w = w.noisy)
    }

    results[[k]] <- list(v_sd = v_sd, w_sd = w_sd, repeats = repeat_results)
  }

  return(results)
}
noisy_data_list_2 <- add_noise_experiment(true.v, true.w, v_sd_seq, w_sd_seq, n_repeat = 10)
```

```r
for (l in seq_along(lambda_vec)) {
  lambda_val <- lambda_vec[l]
  pcode_results <- list()

  for (k in seq_along(noisy_data_list_2)) {
    group <- noisy_data_list_2[[k]]
    group_fits <- list()

    for (i in seq_along(group$repeats)) {
      obs_pair <- group$repeats[[i]]
      observ <- cbind(obs_pair$v, obs_pair$w)

      fit <- pcode(data = observ,
                   time = times,
                   ode.model = MorrisLecar,
                   par.names = c("gca", "phi"),
                   state.names = c("v", "w"),
                   par.initial = c(gca_true, phi_true),
                   lambda = lambda_val,
                   basis.list = basis.list)
```

```
      group_fits[[i]] <- fit
    }

    pcode_results[[k]] <- list(
      v_sd = group$v_sd,
      w_sd = group$w_sd,
      fits = group_fits
    )
  }

  pcode_results_all[[l]] <- list(
    lambda = lambda_val,
    results = pcode_results
  )
}

# save pcode results as tibble
extract_structural_params_all <- function(pcode_results_all) {
  rows <- list()

  for (l in seq_along(pcode_results_all)) {
    lambda_val <- pcode_results_all[[l]]$lambda
    pcode_results <- pcode_results_all[[l]]$results
    for (k in seq_along(pcode_results)) {
      for (i in seq_along(pcode_results[[k]]$fits)) {
        fit <- pcode_results[[k]]$fits[[i]]
        rows[[length(rows) + 1]] <- tibble(
          lambda = lambda_val,
          group = k,
          v_sd = pcode_results[[k]]$v_sd,
          w_sd = pcode_results[[k]]$w_sd,
          n_repeat = i,
          gca = fit$structural.par[1],
          phi = fit$structural.par[2]
        )
      }
    }
  }
  return(bind_rows(rows))
}

param_tibble <- extract_structural_params_all(pcode_results_all)
```

4 Results

```
mle_summary <- mle_df %>%
  group_by(v_sd, w_sd) %>%
  summarise(
    gca_mean = mean(gca),
    gca_sd   = sd(gca),
    phi_mean = mean(phi),
    phi_sd   = sd(phi),
    .groups = "drop"
  )
mle_summary
```

```
## # A tibble: 4 x 6
##    v_sd  w_sd gca_mean gca_sd phi_mean phi_sd
##   <dbl> <dbl>    <dbl>  <dbl>    <dbl>  <dbl>
## 1   0.1 0.001     4.64 0.131     0.399 0.137
## 2   0.1 0.1       4.61 0.0690    0.326 0.0691
## 3  10   0.001     4.93 0.198     0.794 0.114
## 4  10   0.1       4.95 0.0984    0.809 0.158
```

```
# data visulization for MLE method
mle_long<- mle_df %>%
  pivot_longer(cols = c(gca, phi), names_to = "parameter", values_to = "estimate") %>%
  mutate(noise_label = paste0("v=", v_sd, ", w=", w_sd))

ggplot(mle_long %>% filter(parameter == "gca"),
       aes(x = noise_label, y = estimate, fill = noise_label)) +
  geom_boxplot(outlier.alpha = 0.3) +
  geom_jitter(width = 0.2, alpha = 0.5, color = "black") +
  theme_minimal() +
  labs(x = "Noise Level", y = "gca Estimate",
       title = "MLE Estimated gca under Different Noise Levels") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
ggplot(mle_long %>% filter(parameter == "phi"),
       aes(x = noise_label, y = estimate, fill = noise_label)) +
  geom_boxplot(outlier.alpha = 0.3) +
  geom_jitter(width = 0.2, alpha = 0.5, color = "black") +
  theme_minimal() +
  labs(x = "Noise Level", y = "phi Estimate",
       title = "MLE Estimated phi under Different Noise Levels") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

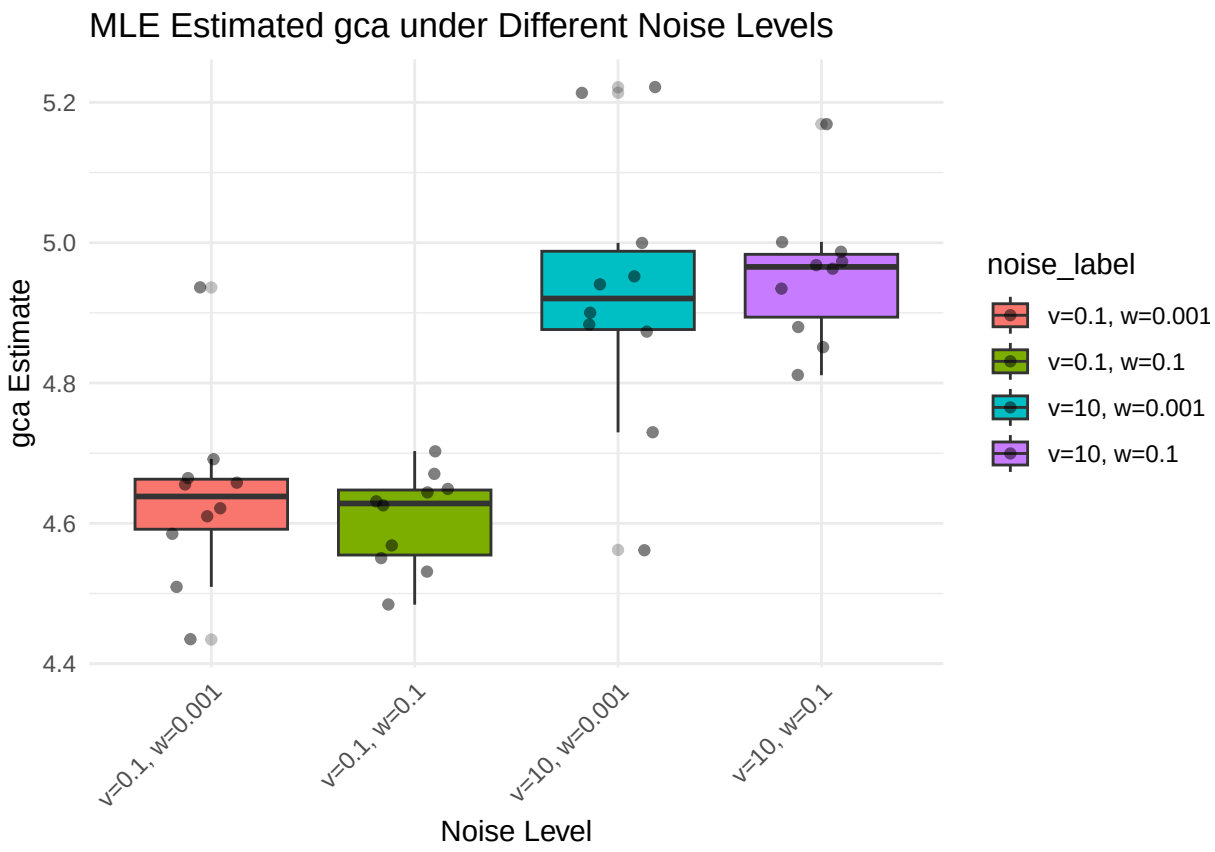

Figure 8: Distribution of MLE-based estimates under different noise levels.

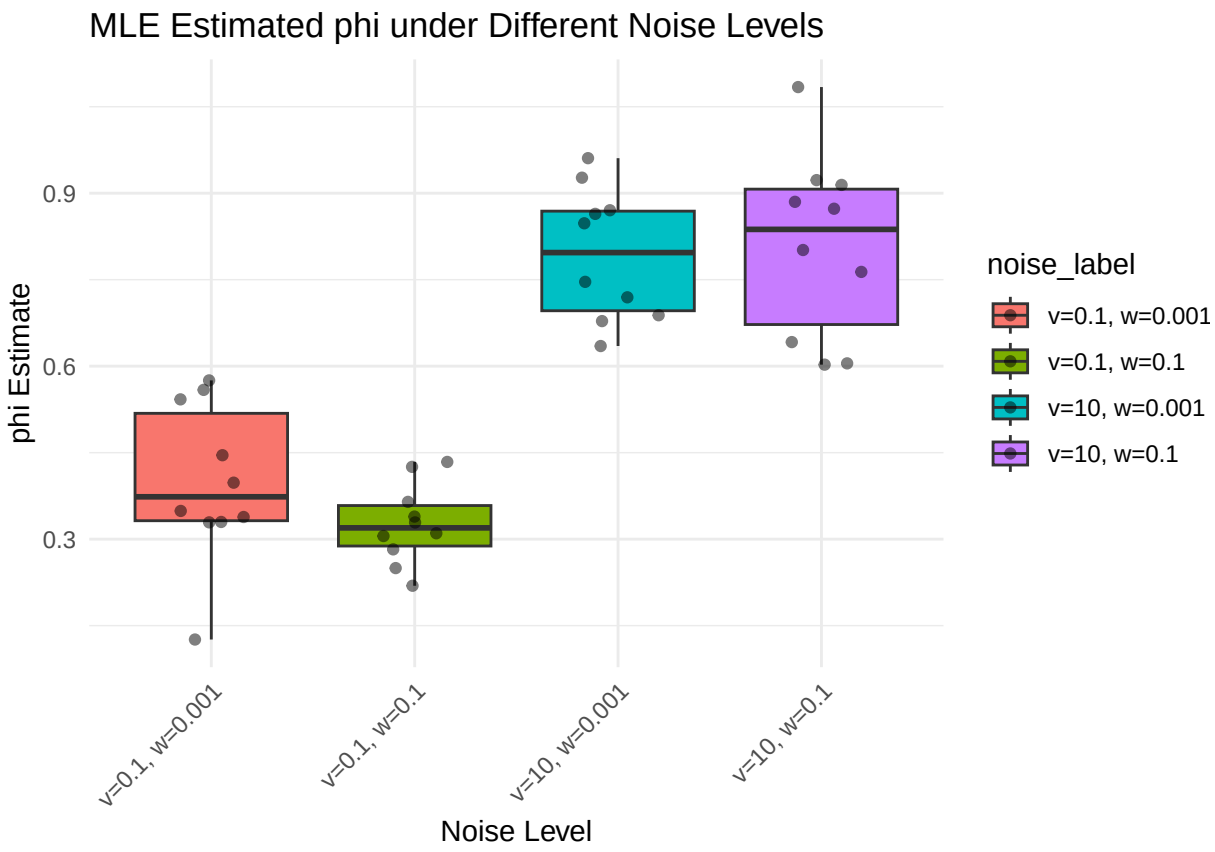

Figure 9: Distribution of MLE-based estimates under different noise levels.

```r
fda_summary <- param_tibble %>%
  group_by(v_sd, w_sd) %>%
  summarise(
    gca_mean = mean(gca),
    gca_sd   = sd(gca),
    phi_mean = mean(phi),
    phi_sd   = sd(phi),
    .groups = "drop"
  ) %>%
  mutate(
    gca_mean = signif(gca_mean, 5),
    gca_sd   = signif(gca_sd, 5),
    phi_mean = signif(phi_mean, 5),
    phi_sd   = signif(phi_sd, 5)
  )
fda_summary
```

```
## # A tibble: 4 x 6
##    v_sd  w_sd gca_mean  gca_sd phi_mean  phi_sd
##   <dbl> <dbl>    <dbl>   <dbl>    <dbl>   <dbl>
## 1   0.1 0.001     4.37 0.00241   0.0267 0.00368
## 2   0.1 0.1       4.44 0.0977    0.0288 0.0455
## 3  10   0.001     4.37 0.0686    0.0259 0.0162
## 4  10   0.1       4.44 0.0991    0.0167 0.0466
```

```r
# data visualization for FDA method
param_long<- param_tibble %>%
  pivot_longer(cols = c(gca, phi), names_to = "parameter", values_to = "estimate") %>%
  mutate(noise_label = paste0("v=", v_sd, ", w=", w_sd))

ggplot(param_long %>% filter(parameter == "gca"),
       aes(x = as.factor(lambda), y = estimate, color = as.factor(lambda))) +
  geom_boxplot(outlier.alpha = 0.3) +
  geom_jitter(width = 0.1, alpha = 0.5) +
  facet_wrap(~ noise_label, nrow = 1, scales = "free_y") +
  coord_cartesian(ylim = c(4.1, 4.7)) +
  theme_minimal() +
  labs(x = "Lambda", y = "gca Estimate",
       color = "Lambda",
       title = "Estimated gca under Different Noise Levels (v_sd, w_sd)")
```

```r
ggplot(param_long %>% filter(parameter == "phi"),
       aes(x = as.factor(lambda), y = estimate, color = as.factor(lambda))) +
  geom_boxplot(outlier.alpha = 0.3) +
  geom_jitter(width = 0.1, alpha = 0.5) +
  facet_wrap(~ noise_label, nrow = 1, scales = "free_y") +
  coord_cartesian(ylim = c(-0.1, 0.21)) +
  theme_minimal() +
  labs(x = "Lambda", y = "phi Estimate",
       color = "Lambda",
       title = "Estimated phi under Different Noise Levels (v_sd, w_sd)")
```

For the MLE method

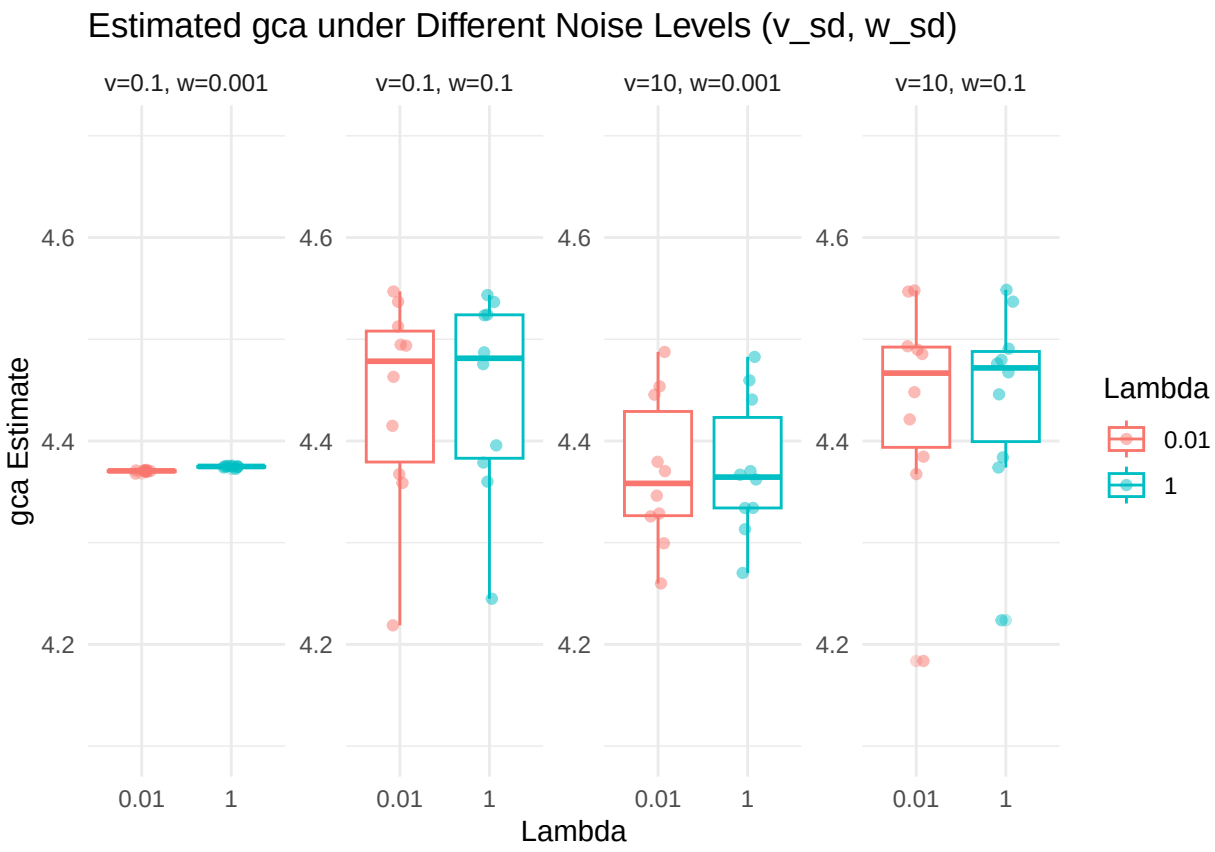

Figure 10: Estimated parameter values under different noise conditions and smoothing penalties.

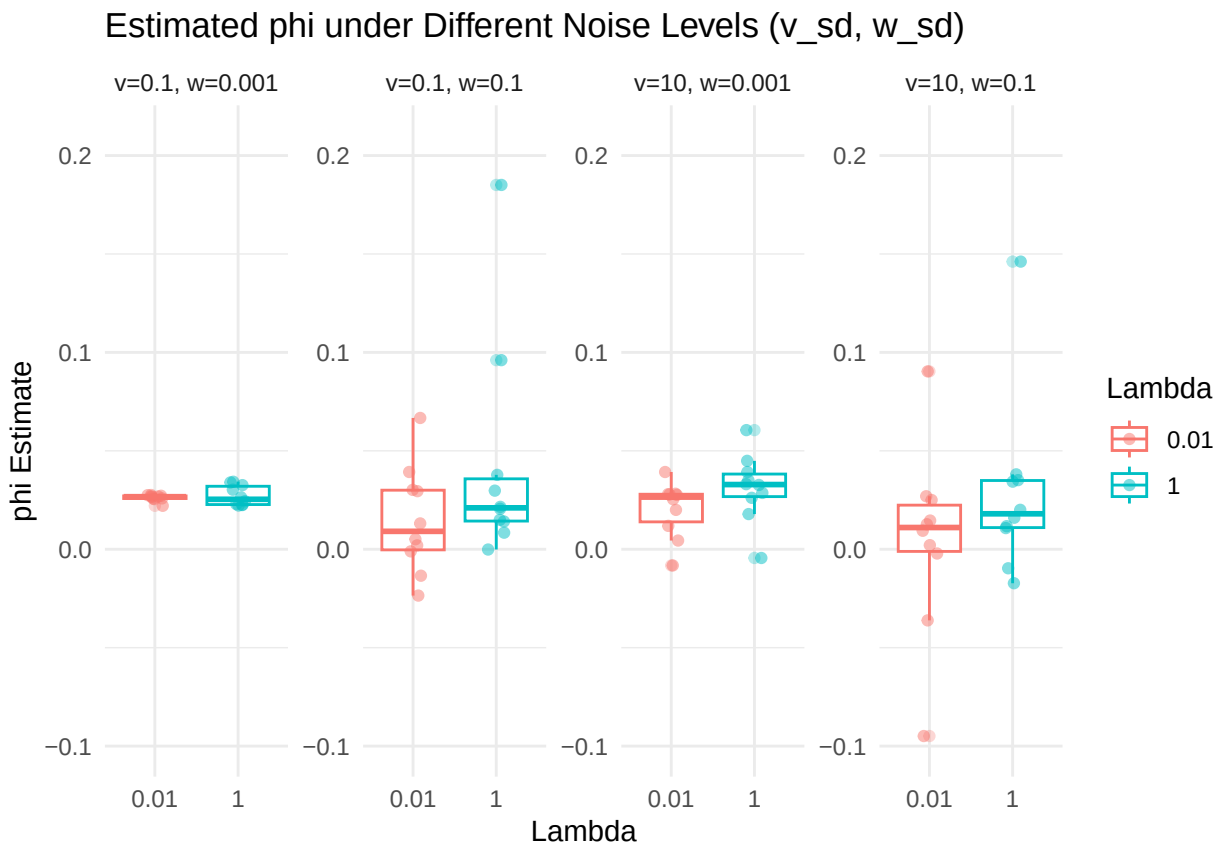

Figure 11: Estimated parameter values under different noise conditions and smoothing penalties.

- When `v_sd` was held constant:
  - At `v_sd` = 0.1, increasing `w_sd` from 0.001 to 0.1 caused the estimated mean of `gca` to shift further away from the true value (4.4), aligning with our expectation that increasing noise in the `w` variable impairs parameter accuracy.
  - However, at `v_sd` = 10, increasing `w_sd` from 0.001 to 0.1 led to the mean of estimated `gca` becoming closer to the true value, contrary to our prior hypothesis. This inconsistency might be attributed to random sampling variability in the simulated datasets.
- When `w_sd` was held constant:
  - Increasing `v_sd` from 0.1 to 10 consistently led to greater deviation of `gca_mean` from the true value, which aligns with our intuition that higher noise in the voltage variable reduces parameter estimation accuracy.

For the pCODE method

- When `v_sd` was fixed:
  - Increasing `w_sd` consistently pushed estimates of mean value of gca further away from the ground truth, which supports our expectation.
- However, when `w_sd` was held constant:
  - As `v_sd` increased from 0.1 to 10, `phi_mean` changed from 0.026676 to 0.025891, moving further away from its true value (`phi_true` = 0.04), while `gca_mean` changed from 4.37249 to 4.37153, unexpectedly moving closer to the true value (`gca_true` = 4.4). This may reflect stochastic variation or model-specific sensitivity to voltage noise under pCODE's basis decomposition framework.

Comparative accuracy between the two methods also revealed consistent patterns. Across all noise levels:

- The mean estimates of `gca` and `phi` obtained via MLE were systematically further from the ground truth values than those obtained via `pCODE`.
- The standard deviations of parameter estimates (`gca_sd` and `phi_sd`) were also substantially higher under maximum likelihood than parameter cascading method, suggesting that the pCODE method yields more precise and robust parameter estimates.

5 Discussion

During the MATH 410 Majors Project, I explored parameter estimation in dynamical systems using the Morris-Lecar model, a two-dimensional system of ordinary differential equations (ODEs) describing neuronal excitability. I implemented the model in R, simulated noisy data, and applied two estimation methods—Maximum Likelihood Estimation (MLE) and the Parameter Cascade Method (via the `pcode` package)—to recover key parameters under different noise conditions. This project strengthened my theoretical understanding of ODE-based modeling and my practical skills in programming, simulation, and statistical optimization.

The first part of the project involved translating the Morris-Lecar model into R code for numerical integration using the `deSolve` package, allowing simulation of trajectories for membrane voltage (`v`) and recovery variable (`w`). These "true" trajectories served as a baseline for generating noisy observations by adding Gaussian noise.

The MLE approach proceeded in three main steps: (1) generating noisy data, (2) defining a loss function as the negative log-likelihood of the observed data given parameter values, and (3) minimizing this loss using R's `optim()` function to estimate parameters (`gca` and `phi`) and noise levels. Repeating this process under different noise conditions allowed assessment of MLE's stability and accuracy.

I then applied the Parameter Cascade Method implemented in `pcode`, which differs from MLE by separating trajectory smoothing and parameter estimation using functional data analysis (FDA). Noisy data were smoothed using B-spline basis expansions, and parameters were estimated by penalizing deviations between the smoothed derivatives and ODE dynamics, regulated by a smoothing penalty (λ). This approach can produce more stable estimates than MLE, particularly when noise is substantial or derivative information is unreliable.

Critically, this project revealed key differences between MLE and FDA-based estimation. MLE directly fits model trajectories to noisy data but is sensitive to noise, initial guesses, and irregular likelihood surfaces, leading to unstable estimates under moderate-to-high noise. FDA, by contrast, smooths data before parameter estimation, improving robustness when noise is large and reducing sensitivity to local minima. However, FDA depends on careful tuning of smoothing parameters, and in settings where true dynamics exhibit sharp transitions or when noise structure is non-standard (e.g., heteroscedastic or multimodal), FDA may struggle or require additional adaptations.

If I had more time, I would extend this work in several directions. First, our experiments only examined the parameter combination gca = 4.4 and phi = 0.04, which corresponds to a specific dynamical regime of the Morris-Lecar model. It would be insightful to investigate other combinations of gca and phi, particularly those that lead to different stability behaviors—for example, where the Jacobian matrix at equilibrium has one positive and one negative eigenvalue (indicating a saddle point) or two positive eigenvalues (indicating an unstable node). Exploring how parameter estimation behaves in different dynamical regimes could improve the robustness of estimation techniques and provide insight into identifiability under various system behaviors.

Second, we assumed that the noise in the observations followed a normal distribution, which is a common but sometimes simplistic assumption. In biological recordings, noise can be heteroscedastic, heavy-tailed, or even multimodal depending on the measurement apparatus and biological context. Therefore, another potential extension would be to explore other noise distributions or to apply robust loss functions that are less sensitive to outliers. In addition, applying these methods to real-world experimental data, such as membrane voltage recordings from neurons, would provide a more challenging and practical test of both MLE and pcode approaches. Accessing and preprocessing such datasets would be a significant step toward validating these techniques beyond simulated scenarios.

In conclusion, this summer project provided me with a deepened understanding of statistical estimation in dynamical systems and hands-on experience with modern tools in R for simulation, optimization, and functional data analysis. It also highlighted the importance of model assumptions, noise structure, and regularization in the recovery of biological parameters. These insights will be valuable as I continue to explore statistical modeling in neuroscience and systems biology.

References

Kerkhoff, D. 2017. "The Morris Lecar Model." https://www.rpubs.com/kerkhoffa/morrislecar.

Morris C, Lecar H. 1981. "Voltage Oscillations in the Barnacle Giant Muscle Fiber." *Biophysical Journal* 35 (1): 193–213.

Phillips, Nathaniel. 2015. "Maximum-Likelihood Fitting in r." https://rpubs.com/yarrr/mltutorial.

Ramsay J. O., B. W., Silverman. 2005. *Functional Data Analysis*. 2nd ed. Springer.

Ramsay J. O., Hooker G., and Graves S. 2009. *Functional Data Analysis with r and MATLAB*. Springer.

Remy S, Beck H. 2006. "Molecular and Cellular Mechanisms of Pharmacoresistance in Epilepsy." *Brain* 129 (1): 18–35.

S., Shorvon. 2010. *Handbook of Epilepsy Treatment*. Wiley-Blackwell.

Soetaert K., Petzoldt T., and Setzer R. W. 2010. "Package deSolve: Solving Initial Value Differential Equations in r." *The R Journal*.

Stefanescu, R. G. Shivakeshavan, R. A., and S. S. Talathi. 2012. "Computational Models of Epilepsy." *Seizure* 21 (10): 748–59.

Wang, Haixu, and Jiguo Cao. 2023. "pCODE: Estimating Parameters of ODE Models by Functional Parameter Cascade." *The R Journal* 14 (4): 291–304.