# pCODE: Estimating Parameters of ODE Models

*by Haixu Wang and Jiguo Cao*

**Abstract** The ordinary differential equation (ODE) models are prominent to characterize the mechanism of dynamical systems with various applications in biology, engineering, and many other areas. While the form of ODE models is often proposed based on the understanding or assumption of the dynamical systems, the values of ODE model parameters are often unknown. Hence, it is of great interest to estimate the ODE parameters once the observations of dynamic systems become available. The parameter cascade method initially proposed by Ramsay et al. (2007) is shown to provide an accurate estimation of ODE parameters from the noisy observations at a low computational cost. This method is further promoted with the implementation in the R package **CollocInfer** by Hooker et al. (2016). However, one bottleneck in using **CollocInfer** to implement the parameter cascade method is the tedious derivations and coding of the Jacobian and Hessian matrices required by the objective functions for doing estimation. We develop an R package **pCODE** to implement the parameter cascade method, which has the advantage that the users are not required to provide any Jacobian or Hessian matrices. Functions in the **pCODE** package accommodate users for estimating ODE parameters along with their variances and tuning the smoothing parameters. The package is demonstrated and assessed with four simulation examples with various settings. We show that **pCODE** offers a derivative-free procedure to estimate any ODE models where its functions are easy to understand and apply. Furthermore, the package has an online Shiny app at `https://pcode.shinyapps.io/pcode/`.

## 1 Introduction

The evolution of a dynamic system in time can be represented by ordinary differential equations (ODE) models consisting of a set of state variables and their derivatives. For example, the state variables can be the population of multiple species in an ecological system, the velocities, and locations of particles in a physics system, the concentration of chemicals in a reactor, or the energy of objects in a heating process. Usually, an ODE model is defined as a set of differential equations in the following form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t|\boldsymbol{\theta}), \tag{1}$$

where $\boldsymbol{x}(t)$ denotes the fully or partially observed states of the system at time $t$. The ODE model links the first derivative, the temporal change, of the states with $\boldsymbol{x}$ itself and other influential components $\boldsymbol{u}$ through some function $\boldsymbol{f}$ defined by the parameter vector $\boldsymbol{\theta}$.

Conventionally, parameter estimation of an ODE model combines numerically solving the system and updating the ODE parameters given the solution. The updating step complies with the nonlinear regression framework as suggested in Bard (1974) and Biegler et al. (1986). That is, the optimal $\boldsymbol{\theta}$ minimizes the sum of squared errors between the observations and the solutions of ODE models. It is not very typical for an ODE model to have analytical solutions unless the dynamic system is simple and linear. In more realistic situations where analytical solutions are not available, the ODE solutions must be numerically obtained prior to any estimations of the ODE parameters. The initial condition, which are values of state variables at the initial time, dictates the unique solution obtained by the numeric solver. However, the initial condition is often unavailable in real practices and must be estimated before obtaining the ODE solutions. If the initial conditions are unknown, then they are augmented to the parameter vector and estimated simultaneously with the ODE parameters. As a result, the complexity of optimization dramatically increases due to the fact that the solution is sensitive to ODE parameters and initial conditions. Also, the augmentation increases the dimension and complexity of the optimization surface hence hindering the convergence of optimization algorithms.

Being an improvement in the stability of estimation, the multiple shooting methods, presented in Bock (1981), Peifer and Timmer (2007), and Voss et al. (2004), expands the dimension of optimization surface by introducing additional initial conditions. The extra initial conditions are a result of partitioning the original time interval into multiple subintervals, and numeric solutions are obtained within each subinterval. Ideally, the introduction of these initial conditions will facilitate the optimization process for $\boldsymbol{\theta}$ to avoid local minima in the optimization surface and provide a faster convergence rate. Additionally, Huang et al. (2006) and Gelman et al. (1996) present Bayesian frameworks of estimating ODE parameters with a hierarchical model.

The aforementioned strategies depend on the numeric solutions of ODE models and can be

burdened with significant computational challenges given complex dynamic systems. That is, the system has to be numerically solved given each update or candidate of ODE parameters. To overcome these challenges, Varah (1982) proposes a two-step procedure to use nonparametric techniques for interpolating the time derivatives of state variables. Subsequently, the parameters are estimated through a nonlinear regression framework by treating the time derivatives as the response. This methodology is further developed in Chen and Wu (2008), Ramsay and Silverman (2005) , and Brunel (2008). Replacing numeric solutions with nonparametric estimations drastically reduces the computational cost, but estimation errors are introduced in the process of interpolating the time derivatives. The errors are due to the fact that there are no direct observations of the derivatives. Furthermore, Cao et al. (2012) shows that the estimation errors of derivatives will lead to bias in the estimation of ODE parameters.

As contrary to the two-step procedure, Ramsay et al. (2007) presents a new approximation strategy, called the parameter cascade method, in the family of collocation methods. The parameter cascade method integrates data smoothing and estimation of ODE parameters into a single estimation scheme. The parameter cascades method provides a much more efficient way of approximating the ODE solution without numerically solving the system. Additionally, it ensures that the ODE parameters are more accurately estimated than the two-step approach (Cao et al., 2012). Various well-known ODE models are examined in Ramsay et al. (2007). Qi and Zhao (2010) has discussed the asymptotic properties of the parameter cascades estimates for both ODE parameters and basis coefficients. The methodology has also been validated with many applications. Zhang et al. (2015) investigates the model selection problem in ODE models based on parameters estimated from the cascading procedure. When the system is measured with replications, mixed effects need to be incorporated into the model. This mixed-effect ODE model is explored in Wang et al. (2014).

The cascading estimation strategy is further promoted as in the R package **CollocInfer** by Hooker et al. (2016). **CollocInfer** offers a variety of functions regarding the estimation of ODE models. It has included the implementation of the parameter cascade method, the tuning criterion of smoothing parameter, and the sample variance of parameter estimates through the Newey-West method in Newey and West (1987). However, one bottleneck of using **CollocInfer** is the requirement of manually providing all the partial derivatives for constructing the Jacobian and Hessian matrices in the optimization process. For complex ODE models, the calculation and coding of those derivatives can be tedious and easy to make mistakes. As an improvement to the **CollocInfer** package, we propose a new package, named **pCODE**, which offers more user-friendly functions for estimating ODE models without specifying any derivatives. That is, an analytic Jacobian and Hessian matrix is required for the optimization routine in **CollocInfer**, whereas **pCODE** automatically calculates the numeric approximations to both matrices. Furthermore, **pCODE** also adds a bootstrap variance estimator besides the variance estimator obtained by the Delta method in Ramsay et al. (2007) and **CollocInfer**. **pCODE** uses the k-fold cross-validation for finding the optimal smoothing parameter while **CollocInfer** selects the smoothing parameters based on the forward prediction errors. We have also developed an online Shiny app at `https://pcode.shinyapps.io/pcode/`.

The remainder of this paper is organized as follows. Next section will introduce the parameter cascade methodology for simultaneous estimation of both ODE solutions $\boldsymbol{x}(t)$ and parameters $\boldsymbol{\theta}$. The third section explains the core functions of the package **pCODE** along with their usages and syntaxes. Subsequently, The fourth section illustrates the application of **pCODE** on several datasets with simulations for assessing the performance. At last, The fifth section gives a conclusion of the paper.

## 2 The parameter cascade method

An ODE model is build upon a set of differential equations $\dot{x}_i(t) = f_i(\boldsymbol{x}, t|\boldsymbol{\theta})$ for $i = 1, ..., I$, corresponding to the index of state variables. Each differential equation represents how the time derivative of one state variable depends on $\boldsymbol{x} = (x_1, ..., x_I)$. Furthermore, $f_i(\boldsymbol{x}, t|\boldsymbol{\theta})$'s are assumed to be parametrized by a vector of parameters $\boldsymbol{\theta}$. Let $y_{ij} = x_i(t_{ij}) + e_{ij}$ be the $j$-th noisy observation of $i$-th state variable at time $t_{ij}$, where $e_{ij}$ is the measurement error or noise, $j = 1, ..., n_i$. We assume the vector of random errors $\boldsymbol{e}_i = (e_{i1}, ..., e_{in_i})$ has a distribution function $g(\boldsymbol{e}_i)$ with the distribution parameter $\boldsymbol{\sigma}_i$. Our goal is to estimate $\boldsymbol{\theta}$ from the noisy observations from the system.

The lower stream estimation of the parameter cascade method is to provide smooth interpolations of the observations. As suggested in Ramsay et al. (2007), the data interpolation part is done by the smoothing splines. The goal of smoothing is to diminish the random errors in the observations of all dimensions, hence the underlying solutions $x_i(t)$'s can be recovered from the noisy observations.

That is, each $x_i(t)$ is expressed as a linear combination of B-spline basis functions

$$x_i(t) = \sum_{i}^{K_i} c_{ik}\phi_{ik}(t) = \boldsymbol{c}_i'\boldsymbol{\phi}_i(t),$$

where $K_i$ is the number of basis functions, and $\boldsymbol{\phi}_i(t)$ is the vector of basis functions evaluated at a time point $t$. Data smoothing is then equivalent to the estimation of these basis coefficients $\boldsymbol{c}_i$'s.

To comply the lower level estimation, a simple objective function $J(\boldsymbol{c})$, following the smoothing spline routine, is considered for obtaining basis coefficients $\boldsymbol{c} = (\boldsymbol{c}_1', \ldots, \boldsymbol{c}_I')'$. That is, the negative log-likelihood function is chosen for non-normal errors, whereas the least square criterion is naturally suitable for i.i.d. normal errors $e_{ij} \sim \mathrm{N}(0, \sigma_i)$. The summation $J(\boldsymbol{c}|\boldsymbol{\sigma}) = \sum_i J(\boldsymbol{c}_i|\boldsymbol{\sigma}_i)$ of individual fitting criterion over $i$ defines a composite objective function for all $I$ dimensions. The data interpolation will use a saturated number of basis functions which requires a penalty term in the objective function to prevent overfitting problems. Ramsay and Silverman (2005) suggests to use a linear differential operator for introducing a smoothness penalty in the objective functions. Subsequently, Poyton et al. (2006) utilizes the smoothness penalty in estimating parameters and solutions of ODE models. Following the same technique, the smoothness penalty is defined based on the discrepancy between the time derivatives and the ODE model, i.e.,

$$\int (\dot{x}_i(t) - f_i(\boldsymbol{x}, t|\boldsymbol{\theta}))^2 dt, \tag{2}$$

for each state variable $x_i$. Multiplying the penalty with a smoothing parameter $\lambda_i$ and combining the penalties over all dimensions, the complete objective function is defined to be

$$J(\boldsymbol{c}|\boldsymbol{\theta}) = \sum_{i=1}^{I} \big[ -\ln g(\boldsymbol{e}_i) + \lambda_i \int (\dot{x}_i(t) - f(\boldsymbol{x}, t|\boldsymbol{\theta}))^2 dt \big]. \tag{3}$$

For each given set of ODE parameters $\boldsymbol{\theta}$, the basis coefficients $\boldsymbol{c}$ is estimated by minimizing $J(\boldsymbol{c}|\boldsymbol{\theta})$. Hence, the estimate for the basis coefficients $\boldsymbol{c}$ become an implicit function of the ODE parameters $\boldsymbol{\theta}$ as $\boldsymbol{c}(\boldsymbol{\theta})$. The objective function (3), referred to as the inner objective function, needs to be optimized for estimating $\boldsymbol{c}$ whenever $\boldsymbol{\theta}$ is updated.

To clearly distinguish two sets of parameters $\boldsymbol{c}$ and $\boldsymbol{\theta}$, $\boldsymbol{c}$ will be referred to as the nuisance parameters since they are not essential for estimating the ODE model rather interpolate the observations. On the other hand, $\boldsymbol{\theta}$ is responsible for defining the structure of the ODE model and will be denoted as structural parameters. Often, $\boldsymbol{\theta}$ will be the primary concern given any ODE models.

The upper stream estimation of the parameter cascade method focuses on the structural parameter $\boldsymbol{\theta}$. To comply this estimation, the objective function $H(\boldsymbol{\theta})$, referred to as the outer objective function, is optimized with respect only to the structural parameters $\boldsymbol{\theta}$. It is usually defined to be the negative log-likelihood function or the sum of squared errors given the distributions of random errors, that is,

$$H(\boldsymbol{\theta}) = -\sum_{i=1}^{I} \ln\, g(\widehat{\boldsymbol{e}}_i|\boldsymbol{\theta}), \tag{4}$$

where

$$\widehat{\boldsymbol{e}}_i = \boldsymbol{y}_i - \hat{x}_i(\boldsymbol{t}_i|\boldsymbol{\theta}).$$

Here $\hat{x}_i(\boldsymbol{t}_i|\boldsymbol{\theta}) = \hat{\boldsymbol{c}}_i'(\boldsymbol{\theta})\boldsymbol{\phi}_i(\boldsymbol{t}_i)$, where $\hat{\boldsymbol{c}}_i'(\boldsymbol{\theta})$ is the optimizer of $J(\boldsymbol{c}|\boldsymbol{\theta})$ given current $\boldsymbol{\theta}$.

The parameter cascade method nests the estimation of basis coefficients in that of the structural parameters, where both estimations depend on the choice of smoothing parameters $\boldsymbol{\lambda}$. As a result, the interpolation of data points is not separated from estimating ODE parameters. The stream of dependencies defined a parameter cascade which offers great accuracy and efficiency in estimating ODE models. To take full advantage of the parameter cascade method, the **pCODE** package provides several R functions that are able to apply the aforementioned methodology for estimating ODE models.

## 3 Main functions in the pCODE package

### Parameter estimation: `pcode`

The main function to perform the parameter cascade method is `pcode` in the package. This is a generic wrapper function for handling several scenarios given different objective functions, normal or non-normal errors, and missing state variables.

First, we focus on the situation when we have the normal random errors for all dimensions where $e_{ij} \sim \mathrm{N}(0, \sigma_i^2)$. In fact, both the inner optimization function (3) and the outer optimization function (4) are calculated based on a vector of residuals. The first part of (3) is easily recognized as the residuals from fitting the observation with basic functions. If the second part, the integral, is approximated by the composite Simpson's rule, then the approximation can be written in a quadratic form based on a vector of residual-like quantities. Stringing the aforementioned two vectors into one, and the concatenation of these vectors from all dimensions will produce a single vector of residuals. Hence, the optimization of the inner objective function adheres to the non-linear least square (NLS) framework. As a result, we can use the popular Levenberg-Marquart (LM) algorithm to obtain an estimate $\hat{\boldsymbol{c}}_i'(\boldsymbol{\theta})$. Our package `pcode` employs the function `lsqnonlin` from the package **PRACMA** (Borchers, 2019) for applying the LM algorithm.

The optimization of the outer objective function appears to be exactly an NLS problem. However, the parameter cascade strategy appends an extra layer of optimization to each update of the outer estimation, which characterizes a nested optimization problem. It is applicable to apply the Levenberg-Marquart algorithm again to the outer objective, however, the computational effort is much greater than that of inner optimization. The reason is that a Jacobian matrix needs to be calculated for each iteration of the update on the structural parameters, and each entry of the Jacobian for the outer objective involves a complete optimization of the inner one. This occurs to be the major delay in computation time.

Most commonly, the random errors $e_{ij}$'s are i.i.d from the normal distribution. However, some dynamic systems may produce non-normal errors which make the NLS approach not applicable. One solution is to perform a transformation on the observations and state variables, which forces the errors of transformed variables to have a normal distribution. As an alternative, `pcode` allows one to input a likelihood (or log-likelihood) function as a fitting criterion for both inner and outer objectives. In such a case, `optim` will be used for both levels of objective functions.

The following demonstrates the syntax of `pcode` with its argument list:

```
pcode(
  data, time, ode.model, par.names, state.names, likelihood.fun,
  par.initial, basis.list, lambda, controls
)
```

where `data`, a matrix, contains the observations of state variables as columns. The names of state variables are stored in `state.names` as a vector. `time` includes the observation time points in a vector. The ODE model to be estimated is provided to `pcode` as `ode.model` and defined in a similar way as that in package **deSolve** (Soetaert et al., 2010):

```
ode.model <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dV <- c * (V - (V^3) / 3 + R)
    dR <- -(1 / c) * (V - a + b * R)
    return(list(c(dV, dR)))
  })
}
```

For this example, the state names are passed to the function `PCODE` as a vector `c('V','R')`, and the structural parameter names `c('a','b','c')` are given to `par.names` argument of the function. Optimization for the structural parameters requires a initial value input as `par.initial`, and there is no need for an input of `likelihood.fun` when the error distributions are Normal. `basis.list` contains the list of basis objects defined by the package **fda**, and the default `basis` has 9 interior knots with B-spline basis functions of order of 4. `lambda` corresponds to the penalty parameter $\boldsymbol{\lambda}$ which controls the fitness of estimated state variables to the differential equations. A scalar input of `lambda` means that all dimensions will subject to the same penalty, or a vector input differentiates the penalty calculated for each dimension. `controls` contains addition parameters to adjust optimizations for both inner and outer objective functions and obtain initial value for basis coefficients $\boldsymbol{c}$. Two scenarios are discussed depending on the different distribution of errors. All the mentioned scenarios will be demonstrated in the fourth section with examples.

### The bootstrap variance estimator: `bootsvar`

All the functions of this package are derivative-free, hence the variance of structural parameters is numerically approximated. The first option is to use the bootstrap variance estimator. Given the estimation of both parameters $\boldsymbol{\theta}$ and $\boldsymbol{c}$, we are able to solve the ODE directly with estimated initial value $\hat{\boldsymbol{x}}(t_0)$. Hence, we can simulate bootstrap samples of the ODE model based on the estimated distributions of errors $\boldsymbol{e}_i$ for all $I$ dimensions. The detailed algorithm for obtaining a bootstrap variance estimate of $\boldsymbol{\theta}$ is as follows

**Algorithm 1** Bootstrap variance estimator

**Initialization:**

- $\boldsymbol{\theta}_0$: initial value for structural parameters
- $B$: the number of bootstrap samples
- $\boldsymbol{y}$: observations

**function** BOOTSVAR($\boldsymbol{\theta}_0, B, \boldsymbol{y}$...)
  $\boldsymbol{\theta}_\star, \boldsymbol{c}_\star \leftarrow$ results from `pcode(`$\boldsymbol{\theta}_0, \boldsymbol{y}$...`)` ▷ Estimation from the original data
  $\sigma_i^2 \leftarrow \text{var}(\boldsymbol{y}_i - \boldsymbol{x}_i)$ ▷ Estimate the variance of observation errors
  $\boldsymbol{x}_\star \leftarrow$ Solution to the ODE given $\boldsymbol{\theta}_\star$ and $\boldsymbol{x}_\star(t_0)$
  **for** $b \leftarrow 1$ to $B$ **do**
    Obtain $\boldsymbol{z}_i^{(b)} = \boldsymbol{x}_{i,\star} + \boldsymbol{\epsilon}_i^{(b)}$. ▷ $\boldsymbol{\epsilon}_i^{(b)} \sim N(0, \sigma_i^2)$
    $(\boldsymbol{\theta}_\star^{(b)}, \boldsymbol{c}_\star^{(b)}) \leftarrow$ results from `pcode(`$\boldsymbol{\theta}_\star, \boldsymbol{z}_i^{(b)}$...`)`
  **end for**
  **return** $Var(\boldsymbol{\theta}_\star^{(b)}), Var(\boldsymbol{c}_\star^{(b)})$
**end function**

The syntax of `bootsvar` is illustrated as the following

```
bootsvar(
  data, time, ode.model, par.names, state.names, par.initial,
  lambda, basis.list, bootsrep, controls
)
```

Most of the arguments are the same as the function `pcode` with only one addition of argument `bootsrep` which indicates the number of bootstrap samples to be taken for obtaining the variance estimator.

### The Delta variance estimator: `deltavar`

As an alternative to the bootstrap variance estimator, the package **PCODE** also offers another numeric estimator for the variance of structural parameters. Ramsay et al. (2007) has developed the approximation to $Var(\hat{\boldsymbol{\theta}}(\boldsymbol{y}))$ via the delta-method. The resulting approximation is of the form

$$Var(\hat{\boldsymbol{\theta}}(\boldsymbol{y})) \approx \left[\frac{d\hat{\boldsymbol{\theta}}}{d\boldsymbol{y}}\right] \boldsymbol{\Sigma} \left[\frac{d\hat{\boldsymbol{\theta}}}{d\boldsymbol{y}}\right]'$$

where $\frac{d\hat{\boldsymbol{\theta}}}{d\boldsymbol{y}}$ is obtained as

$$\frac{d\hat{\boldsymbol{\theta}}}{d\boldsymbol{y}} = -\left[\frac{\partial^2 H}{\partial \boldsymbol{\theta}^2}\bigg|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}\right]^{-1} \left[\frac{\partial^2 H}{\partial \boldsymbol{\theta} \partial \boldsymbol{y}}\bigg|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}\right] \tag{5}$$

and $\boldsymbol{\Sigma}$ is a $N \times N$ variance-covariance matrix for observations put into a vector. $N$ is the total number of observation summing over all dimensions of $\boldsymbol{y}$ . Then the estimation of variance relies on the computation of (5). The partial derivatives are approximated by the finite difference method, i.e.,

$$\left[\frac{\partial^2 H}{\partial \boldsymbol{\theta}_u^2}\bigg|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}\right] \approx \frac{H(\hat{\boldsymbol{\theta}}_{u+}(\boldsymbol{y})|\boldsymbol{\lambda}) - 2H(\hat{\boldsymbol{\theta}}(\boldsymbol{y})|\boldsymbol{\lambda}) + H(\hat{\boldsymbol{\theta}}_{u-}(\boldsymbol{y})|\boldsymbol{\lambda})}{\Delta^2},$$

where $\hat{\boldsymbol{\theta}}_{u+}(\boldsymbol{y})$ and $\hat{\boldsymbol{\theta}}_{u-}(\boldsymbol{y})$ indicate the addition and subtraction of stepsize $\Delta$ to the u-th structural parameter estimate $\hat{\boldsymbol{\theta}}(\boldsymbol{y})$. The mixed partial derivatives are approximated as the following

$$\left[\left.\frac{\partial^2 H}{\partial \boldsymbol{\theta}_u \partial \boldsymbol{\theta}_v}\right|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}\right] \approx \frac{H(\hat{\boldsymbol{\theta}}_{u+,v+}(\boldsymbol{y})) - H(\hat{\boldsymbol{\theta}}_{u-,v+}(\boldsymbol{y})) - H(\hat{\boldsymbol{\theta}}_{u+,v-}(\boldsymbol{y})) + H(\hat{\boldsymbol{\theta}}_{u-,v-}(\boldsymbol{y}))}{4\Delta^2}.$$

Given any fixed argument $\boldsymbol{\theta}$ for the outer objective function $H(\boldsymbol{\theta}, \boldsymbol{\sigma}|\boldsymbol{\lambda})$, its evaluation involves the profiled estimation of $\boldsymbol{c}(\boldsymbol{\theta}, \boldsymbol{\sigma}; |\boldsymbol{\lambda})$ and returns solely the likelihood or the sum of squared errors. Hence, individual evaluations of $H(\boldsymbol{\theta}, \boldsymbol{\sigma}|\boldsymbol{\lambda})$ used in numeric approximation is obtained in the following steps:

- Step 1: Optimize $J(\boldsymbol{c}|\boldsymbol{\theta}, \boldsymbol{\lambda})$ given $\boldsymbol{\theta}$.
- Step 2: Obtain $\hat{\boldsymbol{x}}(\boldsymbol{t})$ based on the estimated $\hat{\boldsymbol{c}}$.
- Step 3: Return $\sum_{i \in I} ||\boldsymbol{y}_i - \hat{x}_i(\boldsymbol{t}_i)||^2$ or $-\sum_{i \in I} g(\boldsymbol{y}_i - \hat{x}_i(\boldsymbol{t}_i)|\boldsymbol{\theta}, \boldsymbol{\lambda})$.

Approximation to the second term $\left.\frac{\partial^2 H}{\partial \boldsymbol{\theta} \partial \boldsymbol{y}}\right|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}$ utilizes the finite difference method as well. After evaluating $H(\boldsymbol{\theta}, \boldsymbol{\sigma}|\boldsymbol{\lambda})$ given $\boldsymbol{\theta}$, the mixed partial derivative is calculated by moving the particular observation up or down by some stepsize $\Delta_y$. That is,

$$\left[\left.\frac{\partial^2 H}{\partial \boldsymbol{\theta}_u \partial y_v}\right|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}\right] \approx \frac{H(\hat{\boldsymbol{\theta}}_{u+}, \boldsymbol{y}_{v+}) - H(\hat{\boldsymbol{\theta}}_{u+}, \boldsymbol{y}_{v-}) - H(\hat{\boldsymbol{\theta}}_{u-}, \boldsymbol{y}_{v+}) + H(\hat{\boldsymbol{\theta}}_{u-}, \boldsymbol{y}_{v-})}{4\Delta\Delta_y},$$

where $\boldsymbol{y}_{v+}$ and $\boldsymbol{y}_{v-}$ represent moving up and down v-th observation by stepsize $\Delta_y$ in the last step of evaluating $H(\boldsymbol{\theta}, \boldsymbol{\sigma}|\boldsymbol{\lambda})$. The syntax of `numericvar` is based on that of `pcode`

```
numericvar(
  data, time, ode.model, par.names, state.names,
  par.initial, lambda, basis.list, stepsize, y_stepsize, controls
)
```

with addition of `stepsize` and `y_stepsize`. `stepsize` can be specified by a single number for which finite difference method will use it for all parameters or a vector where derivative of each parameter is estimated based on its own stepsize. `y_stepsize` allows an input of a vector where each element indicates the stepsize for each dimension of the ODE model.

## 4 Illustrations

### A simple ODE model

A simple illustration uses an one-dimensional ODE model

$$\dot{X} = \theta X(1 - \frac{X}{10}). \tag{6}$$

The following code defines the aforementioned model that will be provided for `pcode` for estimating parameters:

```
model <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dX <- theta * X * (1 - X / 10)
    return(list(dX))
  })
}
```

Given an observation period of $[0, 100]$, random noise errors are added to the ODE solution with a Normal distribution $\mathrm{N}(0, 0.5^2)$. Observations of the system are generated as follows:

```
times  <- seq(0, 100, length.out = 101)
mod    <- ode(y = state, times = times, func = ode.model, parms = model.par)
nobs   <- length(times)
scale  <- 0.5
noise  <- scale * rnorm(n = nobs, mean = 0, sd = 1)
observ <- mod[, 2] + noise
```

Subsequently, we can visualize the observations along the true solution of this simple ODE model in Figure 1.

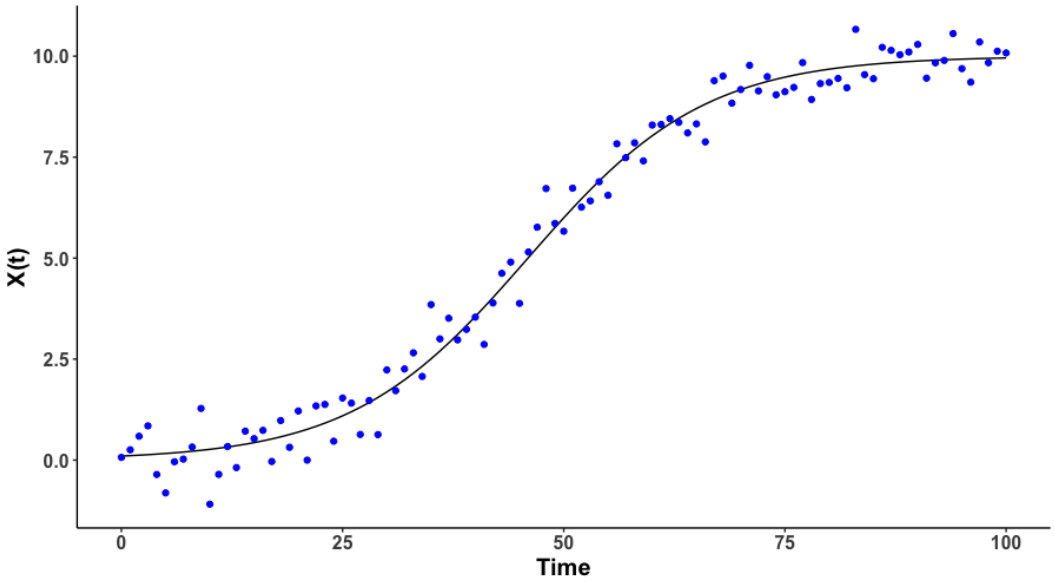



**Figure 1:** This plot demonstrates the solution of an ODE system. The black solid line represents the true ODE solution given a known initial value. The estimation of the solution and parameters of the ODE system usually depends on some noisy observations of the system. The noisy observations, the blue points, are used for estimation.

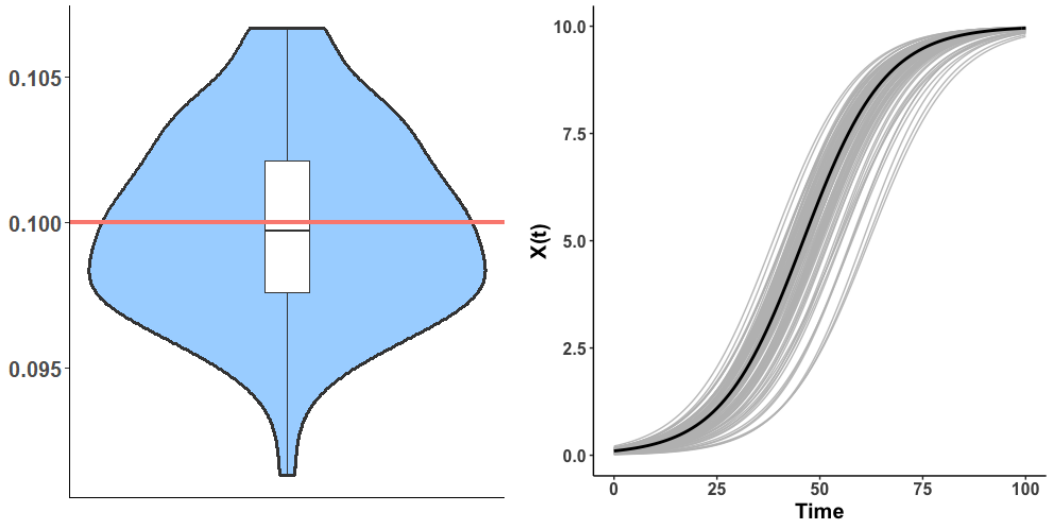
First, a basis object needs to be defined by `create.bspline.basis` from **fda** package

```
knots  <- seq(0, 100, length.out = 21)
norder <- 4
nbasis <- length(knots) + norder - 2
basis  <- create.bspline.basis(c(0, 100), nbasis, norder, breaks = knots)
```

A B-spline basis is created given 21 konts including both interior and boundary knots acorss observation interval $(0, 100)$. The order of basis functions is 4, so there is a total of 23 basis functions. To perform the parameter cascade method for estimating both structural and nuisance parameters, one can use `pcode` in the following way

```
pcode.result <- pcode(
  data = observ, time = times, ode.model = model,
  par.initial = 0.3, par.names = "theta", state.names = "X",
  basis.list = basis, lambda = 1e2
)
pcode.result["structural.par"]
theta
0.09995229
```

To validate the methodology of this function, data simulation and parameter estimation are repeated for 100 times.




**(a)** The violin and boxplot of the estimated $\theta$ over 100 replications.

**(b)** Solution of the ODE model with $\hat{\theta}$ and $\hat{X}(0)$.

**Figure 2:** The violin plot demonstrates the reliability of the estimation procedure. It summarizes the estimates of parameters over 100 replications. Each replication generates a entirely new set of noisy observations and gives a parameter estimate of the simple ODE model (6). The true parameter is indicated by the redline as $\theta = 0.1$. The right plot shows the estimation of the solution from the simple ODE model over 100 replications. Each grey line is obtained from solving the ODE system after obtaining the parameter estimate from one replication. The black curve is the ODE solution with the true value of $\theta$.

Given that $\theta = 0.1$ is used for generating data, the parameter cascade method performs impressively well. The violin plot in Figure 2 shows that the distribution of estimates of $\theta$ covers 0.1 (indicated by the red horizontal line) and does not fall far away from the true value. Also, the state variable `X` (the bold black curve of the plot on the right) is well predicted by solving the ODE model with estimated initial value $\hat{x}(0)$ and structural parameters $\hat{\theta}$.

The true variance for data generating parameter $\theta$ is $1.003 \times 10^{-5}$. We can compare the performance of two functions `bootsvar` and `deltavar` for estimating $\text{var}(\theta)$.

```
bootsvar(
  data = observation, time = times, ode.model = ode.model,
  par.initial = 0.3, par.names = "theta", state.names = "X",
  basis.list = basis, lambda = 1e2, bootsrep = 20
)
theta
1.042763e-05

deltavar(
  data = observation, time = times, ode.model = ode.model,
  par.initial = 0.3, par.names = "theta", state.names = "X",
  basis.list = basis, lambda = 1e2,
  stepsize = 1e-5, y_stepsize = 1e-5
)
theta
9.923318e-06
```

Both variance estimator give excellent estimates of the true variance of structural parameter. Subsequently, the consistency and reliability of those estimator can be inspected by simulation. In 100 simulation replicates, the results of two variance estimators are summarized in Figure 3.

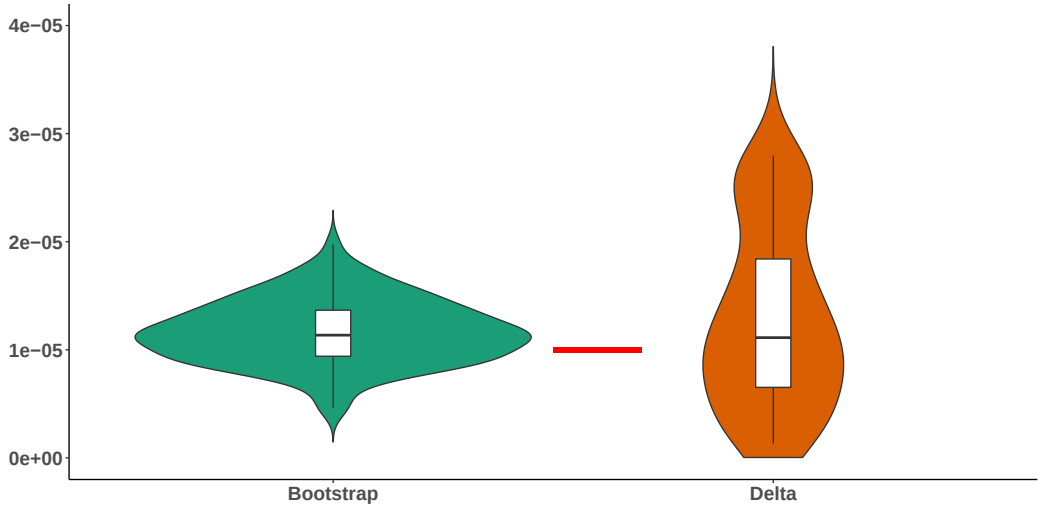



**Figure 3:** This plot demonstrates the estimation of variance of parameters over 100 replications. Two violin plots are results from two methods, the bootstrap and Delta method, introduced in the article. The red line segment indicates the true variance. Green plot includes the estimates from the bootstrap method, whereas the origin plot includes those from the Delta method.

The bootstrap variance estimator has a smaller variation but at the cost of higher computational effort in comparison with the Delta variance estimator. On the other hand, the Delta variance estimator is faster but sensitive to the choice of the step size in estimating all the derivatives and the subsequent var($\theta$).

### The FitzHugh-Nagumo ODE model

The FitzHugh-Nagumo ODE model is well known for capturing the dynamic system of neuronal firings based on the membrane potential V and recovery variable R. This two-dimensional ODE contains 3 positive parameters $\boldsymbol{\theta} = (a, b, c)$, and it can be defined as

$$\dot{V} = c(V - \frac{V^3}{3} + R),$$
$$\dot{R} = -\frac{1}{c}(V - a + bR).$$

The parameters are assigned with values $(a, b, c) = (0.2, 0.2, 3)$ the same as in Ramsay et al. (2007). The state variables V and R should behave as the black solid line plotted in Figure 4. Following the routine of specifying an ODE model, the FitzHugh-Nagumo model is defined in the following function with simulated observations

```
ode.model <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dV <- c * (V - (V^3) / 3 + R)
    dR <- -(1 / c) * (V - a + b * R)
    return(list(c(dV, dR)))
  })
}
model.par   <- c(a = 0.2, b = 0.2, c = 3)
desolve.mod <- ode(y = state, times = times, func = ode.model, parms = c(0.2, 0.2, 3))
nobs    <- length(times)
scale   <- 0.1
noise_v <- scale * rnorm(n = nobs, mean = 0, sd = 1)
noise_r <- scale * rnorm(n = nobs, mean = 0, sd = 1)
observ  <- matrix(NA, nrow = length(times), ncol = 3)
observ[, 1] <- times
observ[, 2] <- desolve.mod[, 2] + noise_v
observ[, 3] <- desolve.mod[, 3] + noise_r
```

The data generating functions and simulated data are illustrated in Figure 4.

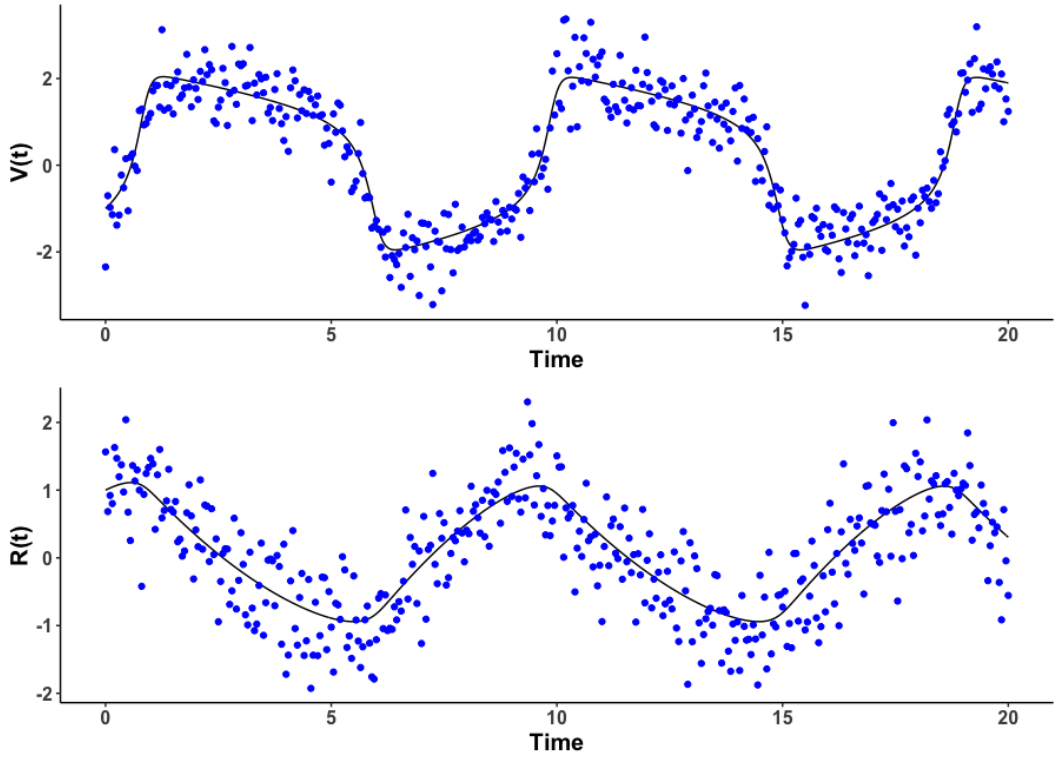



**Figure 4:** Given the known FitzHugh-Nagumo model, we are able to generate the true solutions. In this case, we set the parameters to be $(a, b, c) = (0.2, 0.2, 3)$ and are able to generate the true solution of the system, $V(t)$ and $R(t)$. The top graph demonstrates the solution $V(t)$, the black line, whereas the bottom graph shows the solution $R(t)$. For parameter estimations, we have added random noises to the true solution and use observations as the blue points in both graphs.

First step of estimating parameters is to declare basis object for each state variable. For observation period from time 0 to time 20, the basis is defined on 101 knots with B-spline basis function of order of 4. The total number of nuisance parameters is 103. For this example, the same basis will be used for both dimensions $V$ and $R$:

```
knots  <- seq(0, 20, length.out = 101)
norder <- 4
nbasis <- length(knots) + norder - 2
basis  <- create.bspline.basis(c(0, 20), nbasis, norder, breaks = knots)
basis.list <- list(basis, basis)
```

Then, `pcode` can be executed as follows:

```
pcode.result <- pcode(
  data = observ[, 2:3], time = times, ode.model = Dmodel,
  par.names = c("a", "b", "c"), state.names = c("V", "R"),
  par.initial = rnorm(3), lambda = 1e2,
  basis.list = basis.list
)

pcode.result['structural.par']
0.1953324 0.2203495 2.9269980
```

In addition, we can also compare the estimation performance of the proposed package **pCODE** with the existing package **CollocInfer**. Figure 5 summarizes the comparison between two packages on the parameter estimation of the Fitz-Hugh Nagumo model.

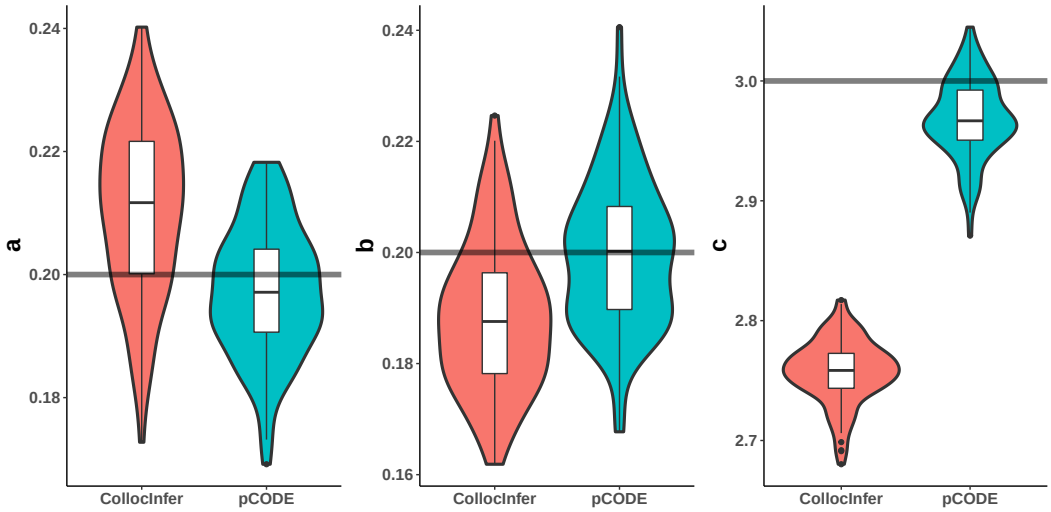



**Figure 5:** We have repeated the data generation and parameter estimation for 100 times given the FitzHugh-Nagumo model. The parameter estimates of $(a, b, c)$ are summarized in the violin plots ordered from left to right. Within each replication, we also use the `CollocInfer` package for parameters estimation. In each plot, the blue violion plot contains estimates from `pCODE` whereas the red one contains those from `CollocInfer`. The black horizontal lines correspond to the true model parameters used for simulating data sets.

Package **CollocInfer** requires the Jacobian and Hessian matrices for estimating the parameters of the ODE models, whereas the proposed **pCODE** package is a derivative-free method. Both packages are estimating parameters a and b with satisfaction, and the estimates from both methods cover the true values. However, we can see that **pCODE** produced better estimations for the third parameter c in the model. Our package does not depend on users to provide tedious details but still performs equally well in parameter estimations.

As a part of the estimation routine, we can use two methods for estimating the variance of structural parameters. First, the usage of bootstrap variance estimator is

```
bootsvar.res <- bootsvar(
  data = data, time = observ[, 1], ode.model = ode.model,
  par.names = c("a", "b", "c"), state.names = c("V", "R"),
  par.initial = c(0.15, 0.25, 5), lambda = 1e2, basis.list = basis.list,
  controls = list(smooth.lambda = 10, verbal = 1, maxeval = 20), bootsrep = 20
)
```

based on a bootstrap sample with size of 20. Delta variance estimator is much faster in approximating the variance through the following execution

```
deltavar.res <- deltavar(
  data = observ[, 2:3], time = observ[, 1], ode.model = Dmodel,
  par.names = c("a", "b", "c"), state.names = c("V", "R"), par.initial = c(0.1, 0.3, 4),
  lambda = 1e2, basis.list = basis.list,
  stepsize = 0.001, y_stepsize = 0.001
)
```

### Non-normal errors

The previous examples assume that the random errors are following Normal distributions, hence the optimization utilizes the routines of NLS problems. In some cases, if the state variables are bounded or the errors have a non-normal distribution, a likelihood (or log-likelihood) function can be passed to the function `pcode` in order to evaluate the objective functions. Subsequently, the function `pcode` will be using `optim` for optimizations of both inner and outter objective functions.

The first example would reuse the simple ODE model, i.e.,

$$\dot{X} = \theta X(1 - \frac{X}{10}).$$

In this case, the observations are simulated where log-normal errors are added to the solution of the ODE model in the following way:

$$log(y_i) = log(x_i) + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$. One simulated data is illustrated in the following Figure 6:

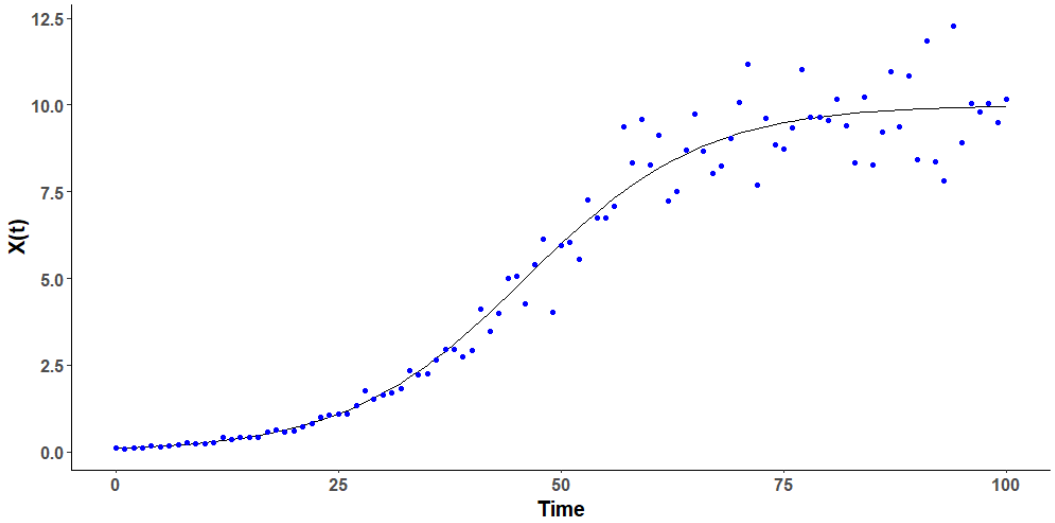



**Figure 6:** This plots illustrates that observations from ODE systems can follow a non-normal distribution. Given the previous simple ODE model, we generate random noises from a log-normal distribution and add them to the true solution of the system. Black line represents the solution of the simple ODE system, and simulated observations are indicated by blue points.

While keeping all the original settings for `pcode`, a log-likelihood function, as $g(\boldsymbol{e}_i|\cdot)$ in both 3 and 4, is additionally specified as follows

```
likfun <- function(x) {
  res <- lapply(x, function(t)
    dnorm(t,
      mean = 0,
      sd = 0.1, log = TRUE
    )
  )
  return(sum(unlist(res)))
}
```

Instead of defining the likelihood only as a function of residuals, `pcode` allows full flexibility in constructing a likelihood as long as it returns the evaluation. Then, the likelihood function can be passed to `pcode` through the argument `likelihood.fun` with specification of a basis object for interpolation

```
knots <- seq(0, 100, length.out = 21)
norder <- 4
nbasis <- length(knots) + norder - 2
basis <- create.bspline.basis(c(0, 100), nbasis, norder, breaks = knots)
lkh.result <- pcode(
  data = observ, time = times, likelihood.fun = likfun,
  par.initial = 0.1, ode.model = ode.model,
  basis.list = basis, par.names = "theta", state.names = "X", lambda = 1e2
)
lkh.result["structural.par"]
0.105
```

## 5 Summary

In this article, we have reviewed the parameter cascade method for estimating ordinary differential equations and introduced the new package **pCODE** for implementing this method. **pCODE** offers a

derivative-free procedure to estimate any ODE models where its functions are easily understood and to apply. Several examples of ODE models are considered for assessing the performance of functions from **pCODE** involving estimating parameters, producing variability measures, and tuning hyper-parameters. Subsequently, we can observe that the implemented functions provide satisfactory results presented with details in the fourth section. Type of examples differs in both model complexity and error distribution. Furthermore, a special case of predator-prey model is studied when some state variables are completely missing from observations. The package is able to simplifies the application procedure and reproduce the estimation results as in Cao et al. (2008).

One of the future works is to expand the flexibility of this package. Even though **pCODE** can provide satisfactory parameter estimates, current functions do not allow users to input the Jacobian and Hessian matrices to speed up the optimization process. In future package developments, we will implement the functionality to allow the input of these matrices. One limitation of the package is the time performance of procedures, especially for the calculation of the bootstrap variance estimator. For that matter, we are implementing this function with parallel computation ability to speed up the calculation. Moreover, we plan to expand the functionalities of the package to include data visualizations and generations of diagnostic plots and summaries.

## Bibliography


Y. Bard. *Nonlinear parameter estimation*. Academic Press, New York, 1974. [p291]

L. T. Biegler, J. J. Damiano, and G. E. Blau. Nonlinear parameter estimation: A case study comparison. *AIChE Journal*, 32(1):29–45, 1986. [p291]

H. G. Bock. Numerical treatment of inverse problems in chemical reaction kinetics. In K. H. Ebert, P. Deuflhard, and W. Jäger, editors, *Modelling of Chemical Reaction Systems*, pages 102–125, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg. [p291]

H. W. Borchers. *pracma: Practical Numerical Math Functions*, 2019. URL https://CRAN.R-project.org/package=pracma. R package version 2.2.5. [p294]

N. J.-B. Brunel. Parameter estimation of ODE's via nonparametric estimators. *Electron. J. Statist.*, 2:1242–1267, 2008. doi: 10.1214/07-EJS132. URL https://doi.org/10.1214/07-EJS132. [p292]

J. Cao, G. F. Fussmann, and J. Ramsay. Estimating a predator-prey dynamical model with the parameter cascades method. *Biometrics*, 64(3):959–967, 2008. [p303]

J. Cao, J. Huang, and H. Wu. Penalized nonlinear least squares estimation of time-varying parameters in ordinary differential equations. *Journal of Computational and Graphical Statistics*, 21(1):42–56, 2012. doi: 10.1198/jcgs.2011.10021. PMID: 23155351. [p292]

J. Chen and H. Wu. Efficient local estimation for time-varying coefficients in deterministic dynamic models with applications to HIV-1 dynamics. *Journal of the American Statistical Association*, 103 (481):369–384, 2008. [p292]

A. Gelman, F. Bois, and J. Jiang. Physiological pharmacokinetic analysis using population modeling and informative prior distributions. *Journal of the American Statistical Association*, 91(436): 1400–1412, 1996. [p291]

G. Hooker, J. Ramsay, and L. Xiao. Collocinfer: Collocation inference in differential equation models. *Journal of Statistical Software, Articles*, 75(2):1–52, 2016. ISSN 1548-7660. doi: 10.18637/jss.v075.i02. URL https://www.jstatsoft.org/v075/i02. [p291, 292]

Y. Huang, D. Liu, and H. Wu. Hierarchical bayesian methods for estimation of parameters in a longitudinal HIV dynamic system. *Biometrics*, 62(2):413–423, 2006. [p291]

W. K. Newey and K. D. West. A Simple, Positive Semi-definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix. *Econometrica*, 55(3):703–708, May 1987. [p292]

M. Peifer and J. Timmer. Parameter estimation in ordinary differential equations for biochemical processes using the method of multiple shooting. *IET systems biology*, 1(2):78–88, 2007. [p291]

A. Poyton, M. Varziri, K. Mcauley, P. Mclellan, and J. Ramsay. Parameter estimation in continuous-time dynamic models using principal differential analysis. *Computers & Chemical Engineering*, 30 (4):698–708, 2006. [p293]

X. Qi and H. Zhao. Asymptotic efficiency and finite-sample properties of the generalized profiling estimation of parameters in ordinary differential equations. *The Annals of Statistics*, 38(1):435–481, 02 2010. doi: 10.1214/09-AOS724. URL https://doi.org/10.1214/09-AOS724. [p292]

J. Ramsay and B. W. Silverman. *Functional data analysis*. New York: Springer, New York, 2nd edition, 2005. [p292, 293]

J. Ramsay, G. Hooker, D. Campbell, and J. Cao. Parameter estimation for differential equations: A generalized smoothing approach. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 69(5):741–796, 2007. [p291, 292, 295, 299]

K. Soetaert, T. Petzoldt, and R. W. Setzer. Solving differential equations in R: Package deSolve. *Journal of Statistical Software*, 33(9):1–25, 2010. ISSN 1548-7660. doi: 10.18637/jss.v033.i09. URL http://www.jstatsoft.org/v33/i09. [p294]

J. Varah. A spline least squares method for numerical parameter estimation in differential equations. *SIAM Journal on Scientific and Statistical Computing*, 3(1):28–46, 1982. doi: 10.1137/0903003. [p292]

H. U. Voss, J. Timmer, and J. Kurths. Nonlinear dynamic system identification from uncertain and indirect measurements. *International Journal of Bifurcation and Chaos*, 14(06):1905–1933, 2004. [p291]

L. Wang, J. Cao, J. Ramsay, D. M. Burger, C. J. Laporte, and J. K. Rockstroh. Estimating mixed-effects differential equation models. *Statistics and Computing*, 24(1):111–121, Jan. 2014. ISSN 0960-3174. doi: 10.1007/s11222-012-9357-1. URL http://dx.doi.org/10.1007/s11222-012-9357-1. [p292]

X. Zhang, J. Cao, and R. J. Carroll. On the selection of ordinary differential equation models with application to predator-prey dynamical models. *Biometrics*, 71(1):131–138, 2015. doi: 10.1111/biom.12243. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/biom.12243. [p292]

*Haixu Wang*
*Simon Fraser University*
*8888 University Dr, Burnaby, BC V5A 1S6*
*Canada*
haixuw@sfu.ca

*Jiguo Cao*
*Simon Fraser University*
*8888 University Dr, Burnaby, BC V5A 1S6*
*Canada*
jiguoc@sfu.ca