2016

Statistical Inference on Dynamical Systems

Hongyuan Wang
University of Kentucky, why8806@gmail.com
Digital Object Identifier: https://doi.org/10.13023/ETD.2016.445

STATISTICAL INFERENCE ON DYNAMICAL SYSTEMS

DISSERTATION

A dissertation submitted in partial fulfillment of

the requirements for the degree of Doctor of

Philosophy in the College of Arts and Sciences

at the University of Kentucky

By

Hongyuan Wang

Lexington, Kentucky

Directors: Dr. David M. Allen and Dr. Arnold J. Stromberg, Professor of Statistics

Lexington, Kentucky

2016

ABSTRACT OF DISSERTATION

STATISTICAL INFERENCE ON DYNAMICAL SYSTEMS

The ordinary differential equation (ODE) is one representative and popular tool in modeling dynamical systems, which are widely implemented in physics, biology, economics, chemistry and biomedical sciences, etc. Because of the importance of dynamical systems in scientific studies, they are the main focuses of my dissertation.

The first chapter of the dissertation is introduction and literature review, which mainly focuses on numerical integration algorithms of ODEs that are difficult to solve analytically, as well as derivative-free optimization algorithms for the so-called inverse problem.

The second chapter is on the estimation method based on numerical solvers of differential equations. We start by reviewing the state-of-the-art Gauss-Newton algorithm based method, with the derivation of approximate confidence intervals. Furthermore, we propose and illustrate a method using Differential Evolution along with numerical ODE integration algorithms, as well as a hybrid method to improve the convergence issue for Gauss-Newton algorithm. A numerical comparison study shows the hybrid method is more numerically stable than the traditional Gauss-Newton algorithm based estimation method.

In Chapter 3 we propose a novel two-step estimation method based on Fourier basis smoothing and pseudo least square estimator. It is less computationally intensive than methods using numerical ODE integration algorithms, and it works better on periodic or near periodic ODE model functions.

In Chapter 4 we expand our study to a population-based hierarchical model to study the correlation between individual features and certain parameter values. Both ML and REML estimation are studied, with more emphasis on REML. An iterative estimation method that incorporates numerical ODE solver into the stochastic approximation EM algorithm for the hierarchical model is proposed and illustrated. Several simulation studies are presented, and a parallel version of the algorithm is implemented as well.

KEYWORDS: Dynamical System, ODE, Computational Statistics, EM Algorithm

Author's signature: Hongyuan Wang

Date: December 6, 2016

STATISTICAL INFERENCE ON DYNAMICAL SYSTEMS

By
Hongyuan Wang

Director of Dissertation: Dr. David M. Allen

Co-Director of Dissertation: Dr. Arnold J. Stromberg

Director of Graduate Studies: Dr. Constance L. Wood

Date: December 6, 2016

To my loving and supportive parents.

ACKNOWLEDGMENTS

The following dissertation, while an individual work, benefited from the insights and direction of several people. First, I would like to express my sincere gratitude to my advisor, Dr. David Allen for his continuous support of my PhD study, and for his patience, motivation, and immense knowledge. He exemplifies the high quality scholarship to which I aspire. In addition, my advisor Dr. Arnold Stromberg provided timely and instructive comments and evaluation at every stage of the dissertation process. Moreover, I would like to thank the complete Dissertation Committee, and outside reader, respectively: Dr. William Griffith, Dr. Chi Wang, Dr. Katherine Thompson and Dr. David Fardo. It's a great honor to have their guidance and help in improving my work.

In addition, I would like to thank my parents, Xueyou Wang and Meiyu Wang, for their love and support. I also would like to express my sincere gratitude to my classmates in my PhD study. I will always cherish our friendship. In the end, I would like to express my appreciation to all the faculty and staff members in the Department of Statistics at University of Kentucky.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 Introduction and Literature Review

The ordinary differential equation (ODE) is one representative and popular tool in modeling dynamical systems, which are widely implemented in physics, biology, economics, chemistry and biomedical sciences, etc. There are relatively few published studies on estimation methods of complicated dynamical systems. Because of the importance of dynamical systems in scientific studies, they are the main focuses of my dissertation. Here, some of the methodology used is reviewed with research and review paper cited.

1.1 Dynamical System

A dynamical system is about evolution of something over time. Usually there are a finite set of states and a set of variables associated with each state. There are rules governing how the state variable change over time. A system of ordinary differential equations is an example of a dynamical system and is one of the focuses of this dissertation.

McGoff et al. (2015) provides a survey on statistical inference of dynamical systems, in which a dynamical system is defined by a stochastic process of the form $(X_n, Y_n)_n$, where X_{n+1} depends only on X_n and possibly some noise, and Y_n depends only on X_n plus some possible noise. Different model structures, such as differential equation model, hidden Markov model and general state space model have been described, and cases with or without observational and dynamical noises have been discussed and summarized in the paper.

Differential equations describe mathematical relations between some unknown function and its derivatives. In applications, the functions usually represent physical quantities, the derivatives represent their rates of change, and the equation defines a relationship between the two. Since these types of relations are quite common, differential equations are widely used in physics, biology, economics, chemistry and biomedical sciences, etc.

Differential equations can be classified into two categories: ordinary differential equation (ODE) and partial differential equation (PDE). ODEs involve one or more functions of a single variable while PDEs allow functions of several variables and partial derivatives of the unknown functions with respect to those variables. Solving a differential equation means to find a function that satisfies the relation, as well as some additional conditions, like initial value condition or boundary value condition. Some ODEs can be solved explicitly while most of the complicated ODEs are difficult or impossible to be solved in exact form.

1.2 Compartmental Models

Compartmental model is a representative tool in modeling pharmacokinetics and pharmacodynamics. A compartmental model can simulate the biologic process involved in the kinetic behavior of a drug after it has been introduced into the body, leading to a better understanding of its pharmacodynamics effects. The objectives of modeling include to make inference about: the uptake rate and steady state level of a heavy metal in animal tissue; the average time of certain drug stays at its site of action; the relative bioavailability of two drugs, etc. Based on Matis et al. (1983), the basic definition for deterministic compartmental model without measurement errors is:

1. Let $X_{ij}(t), i, j = 1, \ldots, m$ denote the amount of drug that originated in compartment i at time 0 that is in compartment j at time t. Let $X_{\cdot j}(t) = \sum_{i=1}^{m} X_{ij}(t)$ be the total amount of drug in j at time t.

2. Let $\boldsymbol{X}(t) = [X_{ij}(t)]$ be the $m \times m$ matrix.

3. Let A_{ij} for $i = 1, \ldots, m; j = 0, \ldots, m$ and $i \neq j$ denote the constant nonnegative and time independent transfer rate at time t of drug from compartment i to j. The units of the transfer rates are time^{-1} and 0 represents the system exterior.

4. Let $a_{ii} = -\sum_{j=0, j \neq i}^{m} a_{ij}$ be the total output transfer rate from compartment i.

5. Let $\boldsymbol{A} = [a_{ij}]_{m \times m}, i, j = 1, \ldots, m$ be the matrix of transfer coefficients.

6. Let $\lambda_1, \ldots, \lambda_m$ be the eigenvalues of matrix A.

7. Let $\boldsymbol{T}_1, \ldots, \boldsymbol{T}_m$ be the corresponding eigenvectors of matrix A.

Then it follows now by definition of compartment model

$$\dot{\boldsymbol{X}}(t) = \boldsymbol{X}(t)\boldsymbol{A} \tag{1.1}$$

where $\dot{\boldsymbol{X}}(t)$ is the $m \times m$ matrix of derivatives with $\dot{X}_{ij} = \mathrm{d}X_{ij}(t)/\,\mathrm{d}t$.

Generally, based on this definition the drug amount in each compartment j is partitioned into the amounts from all the compartments, which gives the $\boldsymbol{X}$ matrix.

A common simplification of the equation is to set $\boldsymbol{X}_{\cdot}(t) = [\boldsymbol{X}_{\cdot j}(t)]$, where $\boldsymbol{X}_{\cdot}(t)$ is an m dimension vector representing the amount of drug in all the states at time t. The solution to Equation 1.1 under some regularity conditions is

$$\boldsymbol{X}(t) = \boldsymbol{T}\boldsymbol{e}^{\boldsymbol{\Lambda}t}\boldsymbol{T}^{-1}\boldsymbol{X}(0) \tag{1.2}$$

where $\boldsymbol{e}^{\boldsymbol{\Lambda}t}$ is the diagonal matrix with elements $e^{\lambda_i t}$ and $\boldsymbol{X}(0)$ is the initial condition diagonal matrix. In application, this usually leads to an explicit 'sum of exponential' form, i.e.,

$$\boldsymbol{X}_{ij}(t) = \sum c_{ij} e^{\lambda_g t} \tag{1.3}$$

where c_{ij} and λ_g are functions that involve the coefficients a_{ij}. This is the general form of explicit solutions for linear deterministic compartmental models. In applications, the first step is to set up the compartment diagram, then system of ODEs could be set up based on the diagram. After the model is built, some additional parameters could be estimated, such as bioavailability: the percentage of administered dose which reaches the systemic circulation of the patient; AUC: area under the plasma curve; mean residence time: the average time that the drug stays at the site of action.

1.3 Numerical Solution to Differential Equations

Some ODEs can be solved explicitly while most of the complicated ODEs are difficult to solve in exact form. The theory of dynamical systems puts emphasis on qualitative analysis of systems described by differential equations. Many numerical methods have been developed to determine solutions with a given degree of accuracy.

For initial value problems of ODEs, the general model can be written as:

$$\dot{\boldsymbol{X}}(t) = f(\boldsymbol{X}(t), \theta), \qquad \boldsymbol{X}(t_0) = \boldsymbol{X}_0 \tag{1.4}$$

Where f is a known function and $\boldsymbol{X}$ is often called state variable. When the equation is unsolvable, numerical methods could be used to approximate the ODE solutions. Before numerically solving an ODE, we would like to know if a unique solution exists or not for a specific initial value problem. The Picard Lindelof theorem (Coddington and Levinson, 1955) guarantees a unique solution on some interval containing t_0 if f is continuous on a region containing t_0 and X_0 and satisfies the Lipschitz condition on the variable X: given two metric spaces (X, d_X) and (Y, d_Y), where d_X denotes the metric on the set X and d_Y is the metric on set Y (for example, Y might be the set of real numbers $I\!R$ with the metric $d_Y(x, y) = |x - y|$, and X might be a subset of $I\!R$), a function $f : X \rightarrow Y$ is called Lipschitz continuous if there exists a real constant $K \geq 0$ such that, for all x_1 and x_2 in X,

$$d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2) \tag{1.5}$$

Lipschitz condition is a strong form of uniform continuity for functions, basically saying that the function is limited in how fast it can change. In general, ODE numerical integration methods can be classified into two categories: one-step methods and multi-step methods. Basic one-step methods include Euler's method, midpoint method, Runge-Kutta method and Bulirsch-Stoer method. Three commonly used multi-step methods are Adams-Bashforth methods, Adams-Moulton methods, and the backward differentiation formulas (BDFs) (Bulirsch and Stoer, 2002). Every integration method is different in terms of orders of accuracy, computation efficiency and stability, etc. Different methods may be appropriate depending on characteristic of the system.

Here, several representative numerical methods are introduced. Euler's method is the most simple and basic explicit method for numerically solving ODEs with a given initial value. The basic formula for the Euler's method is:

$$X_{n+1} = X_n + hf(X_n, t_n) \tag{1.6}$$

which advances a solution from t_n to $t_{n+1} = t_n + h$. Even though it has limited use since it has second order local error and first order global error, which would cause larger error that is accumulated with each successive step, it has serves as an illustration of the concepts involves in the advanced methods.

Runge-Kutta methods propagate a solution over an interval by combining the information from several Euler-style steps (each involving one evaluation of the right-hand f), and then using the information obtained to match a Taylor series expansion up to some higher order. Runge-Kutta methods are among the most popular ODE solvers. They were first studied by Carle Runge and Martin Kutta around 1900. Modern developments are mostly due to Butcher (1963). According to Bulirsch and Stoer (2002), by far the most often used is the classical fourth-order Runge-Kutta formula. The classical Runge-Kutta method has the advantage of self starting, simple to handle, fixed local error order $O(h^5)$, and easy to conduct automatic step size control, even though the computation cost of four functions per step is relatively high. The original Runge-Kutta methods are explicit, however since the problem of stiff equations arises, implicit Runge-Kutta methods have been developed to handle the stability issue.

The methods discussed above are all one-step methods, which only require information about the solution at one time say $t = t_{n-1}$ to compute the solution at an advanced time $t = t_n$. After several points have been found, it is feasible to use several prior points in the calculation. The Adams-Bashforth-Moulton method (Brown et al., 1965) uses $X_{n-3}, X_{n-2}, X_{n-1}$ and X_n to compute X_{n+1}. A desirable feature of a multi-step method is that the local truncation error could be determined and a correction term can be included, which improves the accuracy of the solution at each step. So the Adams-Bashforth-Moulton methods are also known as the most common predictor corrector algorithm, which has the scheme:

$$\text{Predictor :} \quad X_{n+1} = X_n + \frac{h}{12}(23X'_n - 16X'_{n-1} + 5X'_{n-2}) + O(h^4) \tag{1.7}$$

$$\text{Corrector :} \quad X_{n+1} = X_n + \frac{h}{12}(5X'_{n+1} + 8X'_n - X'_{n-1}) + O(h^4) \tag{1.8}$$

Multi-step methods usually suffer from two difficulties in implementation: one is that adjusting the step size is difficult since the formulas require results from equally spaced steps, the other is that it has issue in starting and ending period of the algorithm.

A differential equation of the form $\dot{X}(t) = f(X, t)$ is said to be stiff if its exact solution $X(t)$ includes a term that decays exponentially to zero as t increases, but its derivatives are much greater in magnitude than the term itself. An example is e^{-ct} ,where c is large, positive constant. A large derivative would cause problem unless the step size is sufficiently small. Implicit multi-step methods are commonly used for stiff systems, like implicit Trapezoidal method, etc.

All those methods depend on the choice of step size and other tuning constants. There are recommendations on what method to use for different situations. For example, Petzold (1983) proposes a automatic scheme for determining whether a problem can be solved more efficiently using a class of methods suited for non-stiff problems or a class of methods designed for stiff problems. Liu et al. (2010) provides a numerical simulation of four popular ODE solvers for different problems with continuous, stiff, and hybrid behavior. Some important features, such as number of steps, accuracy, CPU time and the event handling capability, are examined and advice for solver selection is given. An excellent overview of computer methods for numerical ODE solvers is given by Ascher and Petzold (1998). In the implementation, the R package *desolve* (Soetaert et al., 2010) is mainly used for numerical study and real data analysis purposes.

1.4 Derivative Free Optimizations

In general, estimation involves optimizing an objective function with respect to the parameters of the model. In many cases the objective function is the likelihood function. In some cases the objective is to minimize a sum of squares function.

When the ODE above could be solved analytically, the parameter estimation of this model becomes a regular nonlinear least squares optimization problem. However, when the ODE is not solvable, some proper derivative-free optimization algorithms need to be utilized along with the ODE numerical integrators to do model inference.

Based on Rios and Sahinidis (2013), the development of derivative-free algorithms dates back to the works of Spendley et al. (1962) and Nelder and Mead (1965) with their simplex-based algorithms. Other derivative-free optimization techniques like genetic algorithm, hit-and-run algorithm, implicit filtering were proposed later. Significant progress has been made on the algorithmic and theoretical aspects of derivative-free optimization over the past two decades.

The Nelder-Mead uses the concept of a simplex, which is a special polytope of $N+1$ vertices in N dimensions. It begins with a set of $n+1$ points $x_0, \ldots, x_n \in IR^n$ that are considered as the vertices of a working simplex S. The method then performs a sequence of transformations of the working simplex S determined by computing one or more test points, together with their function values, and by comparison of these function values with those at the vertices, to decrease the target function values at its vertices. The Nelder-Mead method frequently gives significant improvements in the first few iterations and quickly produces quite satisfactory results. Also, the method typically requires only one or two function evaluations per iteration. On the other hand, the lack of convergence theory is often reflected in practice as a numerical breakdown of the algorithm, even for smooth and well-behaved functions.

Differential evolution (DE), originally proposed by Storn and Price (1997), is an algorithm that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It is developed to optimize real parameter, real valued functions which can be non-differentiable, non-continuous, non-linear or multi-dimensional. Differential evolution is crudely mimicking the evolution of population members from generation to generation. Instead of one starting value, we need to generate a population of candidates as starting values and can search very large spaces of candidate solutions. It has the advantage of simple structure, easy to use, high speed and robust.

Some selected previous work include: Chen et al. (1999) proposed a method to model gene expression with differential equations. A linear differential equation system was explicitly solved and model was reconstructed for further inference. Soetaert et al. (2010) created an R package *desolve* to solve initial value problems of ODEs using methods like Runge-Kutta. Iba (2008) presented a method to acquire the structure of differential equations by genetic programming from the observed time series. However the statistical inference for identified model is not discussed in the paper. Haario et al. (2013) presented a general scheme for reduction and identification of dynamic models using MCMC and asymptotic model reduction techniques when experimental data from field measurements is noisy and incomplete.

1.5 Population Models

In Chapter 4 population dynamical system models are discussed. Adapting mixed model into dynamical systems enables us to model and make inferences on dynamical systems in a population setting. We could not only characterize the typical parameter values in the population and the extent of their variation, but also study the correlation between individual features and certain parameter values. For instance, if one would like to study if there is a correlation between the eliminating rate of a certain drug and patients' clinical information, such as age, gender, weight or treatment effect, the population dynamical system model would be built since those assumptions could be well represented in the model settings. Therefore population dynamical system model is an practical tool quite routinely used to assess the drug efficacy and safety during the early phase of clinical trials, as well as to aid in the design of pharmaceuticals with desired properties.

A popular tool in longitudinal data analysis of biometric studies is mixed effect modeling. During the development process a lot of different methodologies have been proposed and implemented in the linear mixed model framework (Cnaan et al., 1997). Furthermore, nonlinear mixed effect models has been developed that focuses on features or mechanisms that underlie individual profiles of repeated measurements of the response and how these vary in the population. Some representative estimation algorithms include the Laplace's approximation for nonlinear mixed models by Wolfinger (1993), Gaussian quadrature method by Pinheiro and Bates (1995), etc. Population dynamical system model could be seen as an extension of nonlinear mixed effect models. A lot of methodologies and schema in nonlinear mixed modeled could be adapted and applied in population dynamical system models.

Restricted maximum likelihood (REML) approach is a particular form of maximum likelihood estimation which uses a likelihood of a set of residual contrasts in order to reduce the bias in the estimation of variance components (Patterson and Thompson, 1975). Recently, REML estimation algorithms for generalized linear mixed models and nonlinear mixed effects models have been developed by Liao and Lipsitz (2002) and Meza et al. (2007). An Algorithm for MLE and REML estimation for population dynamical system models is studied and implemented in Chapter 4.

Chapter 2 Estimation Methods Based on Numerical Integration of Differential Equations

2.1 Review: Gauss-Newton Algorithm Based Method

When using software for numerically solving differential equations, one supplies the set of times for which the system is observed, t_0, t_1, $\cdots$, t_m. The software returns $X(t_0)$, $X(t_1)$, $\cdots$, $X(t_m)$. Rarely would every compartment be observed at each time point. A notation for the observational model is required. Let y_{ij} represent the observation on the ith compartment at time t_j. The combination of times and compartments observed must be such the parameters are identifiable. Mock-up notation for a small data set is

$$y = \begin{bmatrix} y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{21} \\ y_{25} \end{bmatrix} \text{ and } \eta(\theta) = \begin{bmatrix} x_1(t_0) \\ x_1(t_1) \\ x_1(t_2) \\ x_1(t_3) \\ x_1(t_4) \\ x_2(t_1) \\ x_2(t_5) \end{bmatrix}. \tag{2.1}$$

Here θ is a vector containing all the unknown parameters of the system of differential equations including the initial values.

For many cases the solution of the differential equations represent the expected value of a set of observations. In this case, the observation model may be written as:

$$\boldsymbol{y} = \boldsymbol{\eta}(\boldsymbol{\theta}, \boldsymbol{t}) + \boldsymbol{\epsilon}. \tag{2.2}$$

The vector $\boldsymbol{\eta}(\boldsymbol{\theta}, \boldsymbol{t})$ is defined by selected elements of the solution of the differential equation

$$\dot{\boldsymbol{X}}(t) = f(\boldsymbol{X}, t, \boldsymbol{\theta}), \qquad \boldsymbol{X}(t_0) = \boldsymbol{X}_0 \tag{2.3}$$

evaluated at times where observations are made. we refer (2.3) as differential equation model.

One of the well-known parameter estimation methods is based on Gauss-Newton algorithm and numerical differential equation approximations (Englezos and Kalogerakis, 2000). Newton's method is a descent method with a specific choice of a descent direction by iteratively adjusting itself to the local geometry of the function to be minimized. The Gauss-Newton method is an approximation of Newton's method for minimizing the nonlinear least squares problem. It has the property of relatively high convergence rate and easy implementation. Moreover, explicitly solving the differential equations prior to model fitting is not necessary, therefore it could be applied to circumstances when analytical differential equation solutions are difficult or impossible to achieve.

In the estimation of the maximum likelihood estimates of the length m vector $\boldsymbol{\theta}$, Newton's method sets up some initial values for $\boldsymbol{\theta}$ and then takes iterative steps to find the estimates that minimize the target function $d(\boldsymbol{\theta}) = -l(\boldsymbol{\theta}; \boldsymbol{y})$. The gradient function and Hessian matrix of the target function is defined as:

$$g(\boldsymbol{\theta}) = \left[\frac{\partial}{\partial \theta_i} d(\boldsymbol{\theta})\right]_{m\times 1} \tag{2.4}$$

$$H(\boldsymbol{\theta}) = \left[\frac{\partial^2}{\partial \theta_i \partial \theta_j} d(\boldsymbol{\theta})\right]_{m\times m} \tag{2.5}$$

Then the quadratic function is obtained from a truncated Taylor's series expansion of the function $d(\boldsymbol{\theta})$ at $\boldsymbol{\theta}^{(i)}$.

$$d(\boldsymbol{\theta}^{(i)} + \boldsymbol{\delta}) \approx q(\boldsymbol{\delta}) = d(\boldsymbol{\theta}^{(i)}) + g(\boldsymbol{\theta}^{(i)})^T \boldsymbol{\delta} + \frac{1}{2}\boldsymbol{\delta}^T H(\boldsymbol{\theta}^{(i)})\boldsymbol{\delta} \tag{2.6}$$

If $H(\boldsymbol{\theta}^{(i)})$ is positive definite, then $\boldsymbol{\delta}^T H(\boldsymbol{\theta}^{(i)})\boldsymbol{\delta} > 0$. If we could find a direction $\boldsymbol{\delta}^{(i)}$ that satisfies $g(\boldsymbol{\theta}^{(i)})^T\boldsymbol{\delta} + \frac{1}{2}\boldsymbol{\delta}^T H(\boldsymbol{\theta}^{(i)})\boldsymbol{\delta} < 0$, then $q(\boldsymbol{\delta}) - d(\boldsymbol{\theta}^{(i)}) < 0$, such $\boldsymbol{\delta}$ is called the descent direction. If $H(\boldsymbol{\theta}^{(i)})$ is not positive definite, it is replaced by a 'nearby' positive definite matrix. To take the first order derivative of $q(\boldsymbol{\delta})$ with respect to $\boldsymbol{\delta}$

$$\frac{\partial}{\partial \boldsymbol{\delta}} q(\boldsymbol{\delta}) = H(\boldsymbol{\theta}^{(i)})\boldsymbol{\delta} + g(\boldsymbol{\theta}^{(i)}) \tag{2.7}$$

In the Newton's method, the next iteration of the parameters is simply taken to be $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} + \boldsymbol{\delta}^{(i)}$, where $\boldsymbol{\delta}^{(i)}$ minimizes $q(\boldsymbol{\delta})$, in this case $\boldsymbol{\delta}$ is computed by setting $\partial q(\boldsymbol{\delta})/\partial \boldsymbol{\delta} = 0$. So Newton's method updates the parameter in each iterate as:

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - H^{-1}(\boldsymbol{\theta}^{(i)}) g(\boldsymbol{\theta}^{(i)}) \tag{2.8}$$

The following is a theorem to show the convergence of Newton's method.

Theorem 2.1.1. *If $\boldsymbol{\theta}^{(i)}$ is sufficiently close to the true minimizer $\hat{\boldsymbol{\theta}}$ at given iteration, and $H(\boldsymbol{\theta}^{(i)})$ is positive definite, then the Newton's method is well defined for the following iterations, and converges at a second order rate.*

Proof. Assume that the elements of the Hessian matrix $H(\boldsymbol{\theta})$ satisfy the Lipschitz condition $|H_{ij}(\boldsymbol{\theta}) - H_{ij}(\boldsymbol{\theta}')| \leq k||\boldsymbol{\theta} - \boldsymbol{\theta}'||$, for a constant k. Then by Taylor series expansion of the gradient function $g(\boldsymbol{\theta})$,

$$g(\hat{\boldsymbol{\theta}}) = g(\boldsymbol{\theta}^{(i)}) + H(\boldsymbol{\theta}^{(i)})(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{(i)}) + O(||\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{(i)}||^2) \tag{2.9}$$

Since $H(\boldsymbol{\theta}^{(i)})$ is positive definite, $d(\boldsymbol{\theta})$ is convex and $H^{-1}(\boldsymbol{\theta}^{(i)})$ exists. If we multiply $H^{-1}(\boldsymbol{\theta}^{(i)})$ on both sides of the equation

$$0 = -\boldsymbol{\delta}^{(i)} + \hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{(i)} + O(||\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{(i)}||^2 H^{-1}(\boldsymbol{\theta}^{(i)})) \tag{2.10}$$

Then we have

$$||\hat{\boldsymbol{\theta}} - (\boldsymbol{\theta}^{(i)} + \boldsymbol{\delta}^{(i)})|| \leq c||\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{(i)}||^2 \tag{2.11}$$

For some constant c. If $\boldsymbol{\theta}^{(i)}$ is in a close neighborhood of $\hat{\boldsymbol{\theta}}$ such as $||\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{(i)}|| < m/c$, where $0 < m < 1$, then we have

$$||\hat{\boldsymbol{\theta}} - (\boldsymbol{\theta}^{(i)} + \boldsymbol{\delta}^{(i)})|| \leq c||\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{(i)}|| * \frac{m}{c} \leq m||\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{(i)}|| < ||\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{(i)}|| \tag{2.12}$$

So the sequence would converge to $\hat{\boldsymbol{\theta}}$ and the rate is shown to be second order from equation 2.11. $\square$

Gauss-Newton algorithm is a common iterative method used for non-linear least square problems and is an approximation of Newton's method using only first order derivatives. The target function to optimize has the form

$$S(\theta) = \sum_{i=1}^{m} d_i(x, \theta)^2 \tag{2.13}$$

Starting with an initial guess $\theta^{(0)}$, the algorithm proceeds by the iteration

$$\theta^{(s+1)} = \theta^{(s)} - (J^T J)^{-1} J^T D(x, \theta^{(s)}) \tag{2.14}$$

Repeat until the predefined convergence criteria has reached. Here $D = (d_1, \ldots, d_m)$ and J is the Jacobian matrix.

$$(J)_{ij} = \frac{\partial d_i(\theta^{(s)})}{\partial \theta_j} \tag{2.15}$$

In a general non-linear regression model, the Jacobian matrix could be derived in exact form. Thus Gauss-Newton method could easily be implemented. However, when the model is in differential equation form, the Jacobian matrix is no longer available analytically, even though the value of first order derivatives is essentially necessary for Gauss-Newton algorithm. Under this specific circumstance, we would implement a framework to approximate Jacobian matrix using numerical discretization-based ordinary differential equation approximations by expanding the

ODE equations.

To apply Gauss-Newton algorithm to do least squares estimation, which is to minimize the objective function

$$S(\theta) = \sum_{i=1}^{m} (y_i - \eta(t_i, \theta))^2 \tag{2.16}$$

The first order derivative of $\boldsymbol{\eta}(\theta, \boldsymbol{t})$ with respect to θ is

$$\boldsymbol{U}(\theta, \boldsymbol{t}) = \begin{bmatrix} \frac{\partial}{\partial\theta_1}\eta(\theta, t_1) & \frac{\partial}{\partial\theta_2}\eta(\theta, t_1) & \dots & \frac{\partial}{\partial\theta_n}\eta(\theta, t_1) \\ \frac{\partial}{\partial\theta_1}\eta(\theta, t_2) & \frac{\partial}{\partial\theta_2}\eta(\theta, t_2) & \dots & \frac{\partial}{\partial\theta_n}\eta(\theta, t_2) \\ \dots & & & \\ \frac{\partial}{\partial\theta_1}\eta(\theta, t_m) & \frac{\partial}{\partial\theta_2}\eta(\theta, t_m) & \dots & \frac{\partial}{\partial\theta_n}\eta(\theta, t_m) \end{bmatrix} \tag{2.17}$$

Consider the basic form for compartmental models

$$\dot{\boldsymbol{X}}(t) = A\boldsymbol{X}(t) \tag{2.18}$$

where A is a matrix composed by parameters. $\frac{\mathrm{d}}{\mathrm{d}\theta}X(t)$ is needed to construct the Jacobian matrix. Since we have:

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\dot{\boldsymbol{X}}(t) = \frac{\mathrm{d}}{\mathrm{d}\theta}A \cdot \boldsymbol{X}(t) + A \cdot \frac{\mathrm{d}}{\mathrm{d}\theta}\boldsymbol{X}(t) \tag{2.19}$$

Let's assume $\boldsymbol{U}(t) = \frac{\mathrm{d}}{\mathrm{d}\theta}\boldsymbol{X}(t)$, then $\frac{\mathrm{d}}{\mathrm{d}\theta}\dot{\boldsymbol{X}}(t) = \dot{\boldsymbol{U}}(t)$. Then the model could be expanded to the following form:

$$\begin{bmatrix} \dot{\boldsymbol{X}}(t) \\ \dot{\boldsymbol{U}}(t) \end{bmatrix} = \begin{bmatrix} A & 0 \\ \frac{\mathrm{d}}{\mathrm{d}\theta}A & A \end{bmatrix} \begin{bmatrix} \boldsymbol{X}(t) \\ \boldsymbol{U}(t) \end{bmatrix} \tag{2.20}$$

Then numerical ODE approximation methods like Runge-Kutta, Adams-Bashforth-Moulton, etc could be used to get the approximation for both $\boldsymbol{X}(t)$ and $\boldsymbol{U}(t)$, for $t = t_0, \ldots, t_n$. In this way the Jacobian matrix for Gauss-Newton algorithm is also built. Then the least square estimator or weighted least squares estimator could be derived. Here θ could be a scalar or vector. It could be assumed that as the approximation from selected numerical ODE integration gets close enough to the true value, the properties of Gauss-Newton algorithm will hold. It is worth noticing that the convergence of Gauss-Newton algorithm is conditional on the fact that the starting value is in the neighborhood of the true parameter values.

Approximate Confidence Intervals

Besides point estimation for parameters, we would like to obtain the confidence intervals for the parameters and furthermore perform hypothesis tests like the Wald test. Based on the observation model

$$Y_i = X(t_i) + \epsilon_i \qquad i = 1, \ldots, n \tag{2.21}$$

By assumption the residual error $\epsilon_i, i = 1, \ldots, n$ are independent and identically distributed, the least square estimator $\hat{\theta}$ of θ minimizes

$$L(\theta) = \sum_{i=1}^{n} (y_i - X(t_i))^2 \tag{2.22}$$

So least square estimator $\hat{\theta}$ satisfies

$$\left. \frac{\partial L(\theta)}{\partial \theta} \right|_{\hat{\theta}} = 0 \tag{2.23}$$

Based on non-linear regression inference, the Jacobian matrix plays the same role as X in linear regression.

$$\theta - \hat{\theta} = (J^T J)^{-1} J' \epsilon \tag{2.24}$$

If we assume $\epsilon \sim N(0, \sigma^2 I)$, we have

$$\theta - \hat{\theta} \sim N(0, \sigma^2 (J^T J)^{-1}) \tag{2.25}$$

where σ^2 is estimated by $s^2 = \frac{L(\hat{\theta})}{n-m}$. Therefore, the marginal confidence intervals for elements of θ could be represented as:

$$\hat{\theta} \pm t_{1-\alpha/2, n-m} \cdot s \cdot (\mathrm{diag}(J^T J)^{-1})^{1/2} \tag{2.26}$$

The Jacobian matrix J is provided by the proposed Gauss-Newton based method, thus the confidence interval could be computed without any extra effort. Hypothesis tests on θ could be conducted similarly.

Simulation Study: a 2-Compartment Model

A compartment model can simulate the biologic process involved in the kinetic behavior of a drug after it has been introduced into the body, leading to a better understanding of its pharmacodynamics effects.

For a specific 2-compartment model illustrated in Figure 2.1, we have

$$A = \begin{bmatrix} -\theta_1 & \theta_2 \\ \theta_1 & -\theta_2 - \theta_3 \end{bmatrix} \tag{2.27}$$

Since $\theta = [\theta_1, \theta_2, \theta_3]$ is a 3 dimension vector. We define $U_j(t) = \frac{d}{d\theta_j} X(t)$ and $\boldsymbol{U}(t) = (U_1(t), U_2(t), U_3(t))'$, then $\frac{d}{d\theta_j} \dot{\boldsymbol{X}(t)} = \dot{U_j}(t)$, $j = 1, 2, 3$. Then the model becomes:

$$\begin{bmatrix} \dot{\boldsymbol{X}}(t) \\ \dot{U_j}(t) \end{bmatrix} = \begin{bmatrix} A & 0 \\ \frac{d}{d\theta_j} A & A \end{bmatrix} \begin{bmatrix} \boldsymbol{X}(t) \\ U_j(t) \end{bmatrix} \tag{2.28}$$

for θ_j equals θ_1, θ_2, θ_3 respectively. The initial value for $\boldsymbol{X}(t)$ is usually predefined. If not we could set them as parameters and get their least squares estimator. The initial value for $\boldsymbol{U}(t)$ should be set to zero.

For the observed data set: $\{t_i, y_{1i}, y_{2i}\}$; $i = 1, \ldots, n$. To get the least square estimation of θ is to minimize

$$\sum_{i=1}^{n}(X_1(t_i) - y_{1i})^2 + \sum_{i=1}^{n}(X_2(t_i) - y_{2i})^2 \tag{2.29}$$

with respect to $\theta_1, \theta_2, \theta_3$. In many situations it is appropriate to give different weights to those terms.

To make it easier to read, we concatenate $\mathbf{y}_1$ and $\mathbf{y}_2$ and name it $\mathbf{y}$. We did the same thing to $\mathbf{X}_1(t)$ and $\mathbf{X}_2(t)$ and get $\mathbf{X}(t)$. Let $f(x) = \sum_{i=1}^{n} r_{1i}^2 + \sum_{i=1}^{n} r_{2i}^2 = \|r\|^2$, where $r_{1i} = y_{1i} - X_1(t_i)$ and $r_{2i} = y_{2i} - X_2(t_i)$. Then

$$J_1 = \begin{bmatrix} \frac{\partial r_{11}}{\partial \theta_1} & \cdots & \frac{\partial r_{11}}{\partial \theta_m} \\ \vdots & & \vdots \\ \frac{\partial r_{1n}}{\partial \theta_1} & \cdots & \frac{\partial r_{1n}}{\partial \theta_m} \end{bmatrix} = - \begin{bmatrix} \frac{\partial X_1(t_1)}{\partial \theta_1} & \cdots & \frac{\partial X_1(t_1)}{\partial \theta_m} \\ \vdots & & \vdots \\ \frac{\partial X_1(t_n)}{\partial \theta_1} & \cdots & \frac{\partial X_1(t_n)}{\partial \theta_m} \end{bmatrix} \tag{2.30}$$

$$J_2 = \begin{bmatrix} \frac{\partial r_{21}}{\partial \theta_1} & \cdots & \frac{\partial r_{21}}{\partial \theta_m} \\ \vdots & & \vdots \\ \frac{\partial r_{2n}}{\partial \theta_1} & \cdots & \frac{\partial r_{2n}}{\partial \theta_m} \end{bmatrix} = - \begin{bmatrix} \frac{\partial X_2(t_1)}{\partial \theta_1} & \cdots & \frac{\partial X_2(t_1)}{\partial \theta_m} \\ \vdots & & \vdots \\ \frac{\partial X_2(t_n)}{\partial \theta_1} & \cdots & \frac{\partial X_2(t_n)}{\partial \theta_m} \end{bmatrix} \tag{2.31}$$

And

$$J = \begin{bmatrix} J_1 \\ J_2 \end{bmatrix} \tag{2.32}$$

is the Jacobian matrix.

The Algorithm is:

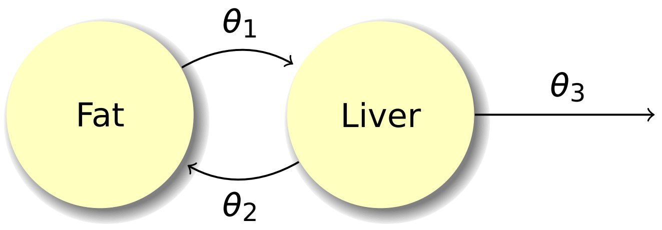


Figure 2.1: Example diagram of the 2-compartment model

1. Set $\theta^{(0)}$ and set $s = 0$.

2. Compute $X(t_i)$, $U_1(t_i)$, $U_2(t_i)$, $U_3(t_i)$, $i = 1, \ldots, n$ using initial value numerical ODE approximations based on $\theta^{(s)}$. Form the Jacobian matrix J.

3. Update $\theta^{(s+1)} = \theta^{(s)} - (J^T J)^{-1} J^T (\mathbf{y} - \mathbf{X}(t))$

4. Test convergence. Stop if convergence is reached, otherwise set $s = s + 1$ and go back to step 2.

Furthermore we would like to apply the method described above on a real dataset. We would like to build a pharmacokinetic two compartment model with two sets of observation data (Karline Soetaert, 2011): a substance accumulated in the fat and eliminated by the liver. We got 8 different time points: $0, 4, 8, 12, 16, 20, 24, 28$. The observed concentration on fat are X_1 : 0.0001, 0.041, 0.05, 0.039, 0.031, 0.025, 0.017, 0.012 at each time point. The concentration on liver are X_2 : 1.31, 0.61, 0.49, 0.41, 0.20, 0.12, 0.16, 0.21 respectively.

A dynamic 2-compartment model is built as section 2.1 illustrates. Using the method described in section 2.1, the least square estimator and the appropriate confidence interval for $\boldsymbol{\theta}$ is obtained. The result shows that our algorithm takes about 15 iterations to converge and the estimation of θ is $(0.673, 0.070, 0.085)$ with 95% confidence intervals $[-9.401, 10.747]$, $[-0.993, 1.133]$, $[0.058, 0.113]$ respectively. The plots of the model along with observations is on Figure 2.2.

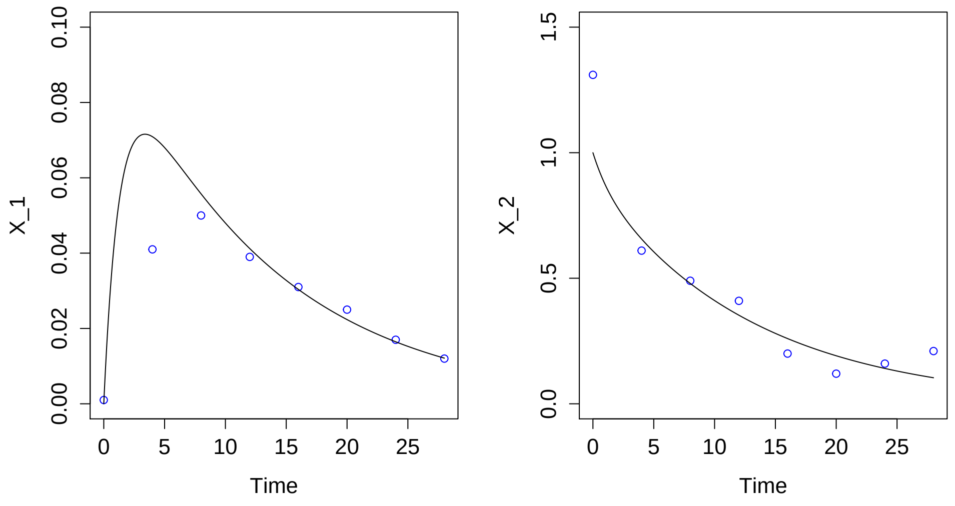


Figure 2.2: Plots of 2-compartment model with observations

2.2 Application: Acetaminophen in a Lactating Goat

Wilson et al. (1986) conducted a study to examine the relationship of milk and plasma levels of a drug. A solution of acetaminophen was infused into a lactating goat through an EJV catheter using an IVAC 630 pump. Levels of concentration in plasma and milk were measured at selected points in time but not the same time for each compartment. The three compartment system shown in Figure 2.3 is considered appropriate for modeling this biological system.

Time t is measured from zero at the time infusion starts. Infusion stops at $t = 63.3$,

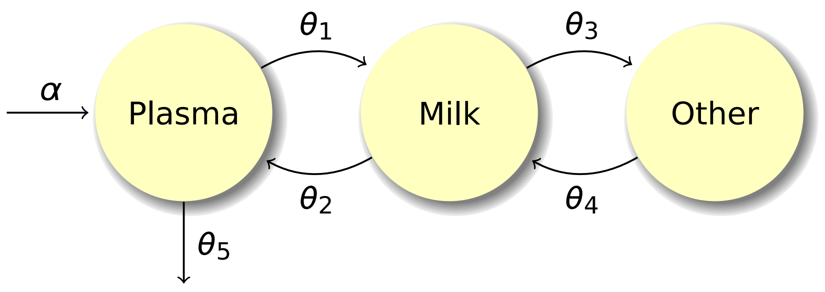


Figure 2.3: The compartmental diagram. The infusion input parameter, α is set to 0 after $t = 63.3$ min.

but observations continue through 300 minutes. Levels of drug concentration in both plasma and milk are sampled but not at exactly the same times.

The model structure is

$$\dot{\boldsymbol{X}}(t) = \boldsymbol{a}_i + A\boldsymbol{X}(t) \qquad \boldsymbol{X}(t_0) = \boldsymbol{X}_0, \quad i = 1, 2 \tag{2.33}$$

The system matrix is:

$$A = \begin{bmatrix} -(\theta_5 + \theta_1) & \theta_2 & 0 \\ \theta_1 & -(\theta_2 + \theta_3) & \theta_4 \\ 0 & \theta_3 & -\theta_4 \end{bmatrix} \tag{2.34}$$

This matrix applies through both phases. The infusion input vectors are

$$\boldsymbol{a}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \alpha \qquad \boldsymbol{a}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.35}$$

Plasma		**Milk**	
Time	Concentration	Time	Concentration
10.47	13.08	10.00	0.29
20.00	18.62	20.00	3.36
30.00	22.05	30.00	4.01
40.00	24.79	40.00	7.74
50.00	25.71	50.00	10.40
61.00	28.11	60.00	12.86
70.17	17.94	70.52	12.41
80.00	9.89	80.33	17.86
90.80	6.20	91.13	16.22
100.22	4.04	100.75	11.39
109.73	3.58	110.33	14.60
119.88	2.27	120.47	13.09
149.92	0.69	150.33	7.31
180.00	0.36	180.60	4.89
210.63	0.23	211.07	2.47
240.00	0.14	240.38	2.32
		280.33	1.99
		300.42	0.67

Table 2.1: Observations for concentration in plasma and milk level of a drug.

The experimental dataset is shown in Table 2.1. As we can see, the challenging part of model estimation is that: the ODE system changes in two consecutive stages divided by a given time point. So initial value of ODE system during the second stage (after $t = 63.3$) is unknown and needs to estimate using the information in the first stage. Moreover, the number of observations and observation time are different for milk and plasma level, which increases the complexity of our computation.

We use Gauss-Newton based algorithm for parameter estimation, and the approximate confidence interval for the 6 parameters is also been calculated. The validity of approximate confidence intervals produced by Gauss-Newton Algorithm is assessed by comparing with bootstrap confidence intervals. The final result is shown in Table 2.2. As we can see those two sets of confidence intervals are consistent in a certain

Parameter	Estimation	Approximate CLs	Bootstrap CLs
α	1.562	[1.34, 1.77]	[1.45, 1.68]
θ_1	0.014	[0.012, 0.017]	[0.012, 0.015]
θ_2	0.004	[-0.004, 0.012]	[-0.004, 0.006]
θ_3	0.008	[-0.001, 0.017]	[0.004, 0.016]
θ_4	0.000	[-0.003, 0.003]	[-0.002, 0.002]
θ_5	0.043	[0.030, 0.056]	[0.036, 0.049]

Table 2.2: Summary statistics for parameter estimations for Gauss-Newton based algorithm.

level, which proves the validity of the approximate confidence interval generated using information from Gauss-Newton algorithm. Figure 2.4 shows the fitted value versus actual observations. As we can see not only the fitted value for observed compartment (plasma and milk) can be obtained, the unobserved other compartment could also be predicted by the ODE model.

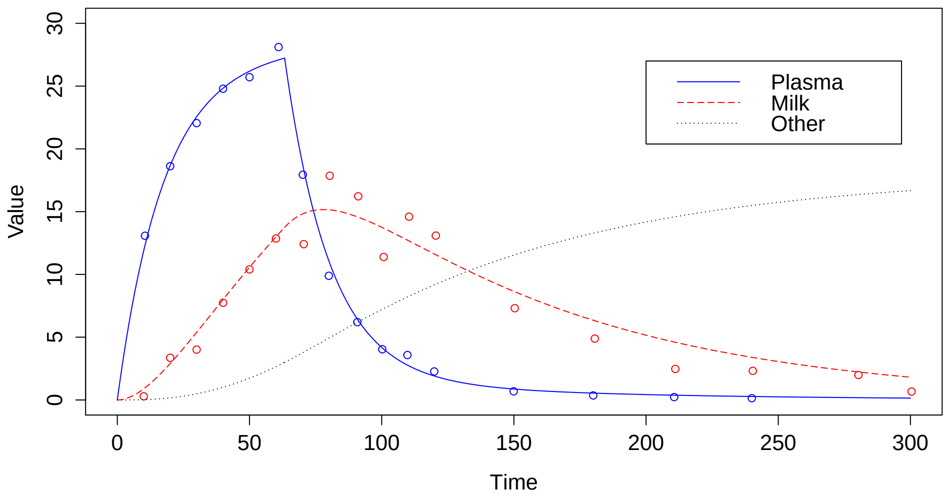


Figure 2.4: Model fit versus actual observations. Concentrations in plasma, milk and other compartment are in the plot.

2.3 Differential Evolution Based Method

One issue with the conventional gradient-based optimization methods such as the Gauss-Newton method is that it may fail to converge or may converge to a local minimum if the initial values of the unknown parameters and the state variables are not close enough to the true values, as shown in 2.1.1. Therefore, global optimization method is needed when there is no accurate enough prior information for the parameters or when the Gauss-newton algorithm fails to converge.

Differential Evolution is a very simple and powerful population based, stochastic function minimizer in the continuous search domain which belongs to the class of genetic algorithms. It is based on a particular way of constructing so-called mutant vectors by using differences between randomly selected elements from the current population. It is designed to be a stochastic direct search method thus it has the ability to handle non-differentiable, nonlinear and multi-modal cost functions. In what follows, a brief sketch of Differential Evolution algorithm is presented.

Differential Evolution

Let's assume N_p to be the number of parameter vectors $x \in IR^d$, where d denotes dimension. The initial N_p parameter vectors are generated either by random values between upper and lower bounds defined by user, or using values assigned by user. Each generation involves creation of a new population from the current population members $\{x_i, i = 1, \ldots, N_p\}$, which is accomplished by mutation of population members. Each member will go through mutation, recombination and selection step iteratively until some stopping criterion is reached. An initial mutant parameter vector v_i is built by randomly selecting three population members, x_{i_1}, x_{i_2} and x_{i_3}. Then

$$v_i = x_{i_1} + F(x_{i_2} - x_{i_3}) \tag{2.36}$$

where F is a positive scale factor. In some implementations, Equation 2.36 would include a term in direction of the best member as shown in (2.40). After the first mutation operation, mutation is continued until d mutations have been made, with an optional crossover probability $\mathrm{CR} \in [0, 1]$. The crossover probability CR controls the fraction of the parameter values that are copied from the mutant. Mutation is applied in this way to each member of the population. If an element of the trial parameter vector is found to violate the bounds after mutation and crossover, it is reset in such a way that the bounds are respected (with the specific protocol depending on the implementation). Then, the objective function values associated with the children are determined. If a trial vector has equal or lower objective function value than the previous vector, then it replaces the previous vector in the population; otherwise the previous vector retains.

It has been shown to be effective on a large range of classic optimization problems. Ali and Törn (2004) found that Differential Evolution was more accurate and efficient than controlled random search and another genetic algorithm. Lampinen and Storn (2004) showed that it was more accurate than some alternative optimization approaches, such as four genetic algorithms, etc.

The general structure of combining Differential Evolution with numerical ODE approximation goes below:

1. Set up the population size N_p, the weighting factor F and mutation factor M.

2. Create initial population. Set up the upper and lower bound for each parameter

$$x_j^L \leq x_{ij} \leq x_j^U \tag{2.37}$$

   Randomly select the initial parameter $\boldsymbol{x} = \{x_{ij}\}, i = 1, \ldots, N_p, j = 1, \ldots, d$ uniformly on $[x_j^L, x_j^U]$.

3. Use numerical ODE solver to compute and store the best parameter set that maximizes the log likelihood (or minimize the mean square error), call it $\boldsymbol{x}_{ib}$.

4. Create difference vectors

$$\boldsymbol{D}_i = \boldsymbol{x}_{i_1} - \boldsymbol{x}_{i_2}, \qquad i = 1, \ldots, N_p \tag{2.38}$$

   where i_1 and i_2 are randomly selected from the population index.

5. Apply mutation on difference vectors

$$D_{ik} = \begin{cases} -D_{ik} & \text{if runif} < M \\ D_{ik} & \text{otherwise} \end{cases} \tag{2.39}$$

   where $i = 1, \ldots, N_p$ and $k = 1, \ldots, d$.

6. Form and check offspring.

$$\boldsymbol{T}_i = \boldsymbol{x}_i + F \cdot \boldsymbol{D}_i + F \cdot (\boldsymbol{x}_{ib} - \boldsymbol{x}_i) \tag{2.40}$$

$$\boldsymbol{x}_i = \begin{cases} \boldsymbol{T}_i & \text{if } \ell(\boldsymbol{T}_i) > \ell(\boldsymbol{x}_i) \\ \\ \boldsymbol{x}_i & \text{otherwise} \end{cases} \qquad i = 1, \ldots, N_p \tag{2.41}$$

   where ℓ represents the log likelihood function that computed by selected numerical ODE solvers.

7. Repeat step 3 to 6 until the stopping criteria is met

Hybrid Method

Despite of all the excellent features about Differential Evolution, there is no proof of convergence for the algorithm. Moreover, incorporating numerical ODE integrations into Differential Evolution is quite computationally intensive for complicated ODE models with a relatively large set of parameters to estimate, since numerical ODE solvers need to be called multiple times for all members in the population in each iteration. On the other hand, the rate of convergence of Gauss-Newton algorithm can approach quadratic. So under certain circumstances the it be more efficient in searching for optimal values than Differential Evolution. However, Gauss-Newton algorithm may converge slowly or not at all if the starting value is far from optimal or the matrix $J^T J$ is ill-conditioned, even on the situation when the approximate Jacobian matrix using numerical ODE integration is accurate enough.

To take the advantage of both optimization algorithms, we propose and implement a hybrid method which uses Differential Evolution to do a pre-selection for starting values that would be used in Gauss-Newton algorithm. The input parameter is number of generations (iterations) for Differential Evolution, which can be a small integer like 5 or 10. The hybrid method is expected to be more stable and efficient than Gauss-Newton method under some circumstances.

In Section 2.5 a series of numerical study is conducted to compare the performance of the hybrid method with Gauss-Newton algorithm based method and Differential Evolution based method under the situation of a linear ODE system.

Application: FitzHugh-Nagumo model

The FitzHugh-Nagumo (FHN) model is a two-dimensional model that describes the voltage potential across the cell membrane of the axon of giant squid neurons. The motivation for the FHN model was to isolate conceptually the essentially mathematical properties of excitation and propagation from the electrochemical properties of sodium and potassium ion flow.

$$\frac{dV}{dt} = \gamma(V - \frac{V^3}{3} + R)$$

$$\frac{dR}{dt} = -\frac{1}{\gamma}(V - \alpha + \beta R)$$

Where V is the membrane potential that depends on a recovery variable R, and $\alpha, \beta, \gamma \in IR^+$ are model parameters.

This is a typical nonlinear differential equation model in which the parameters are hardly accessible for direct measurement. Differential Evolution is applied here along with numerical simulation to do parameter estimation and bootstrap method is used to generate confidence intervals.

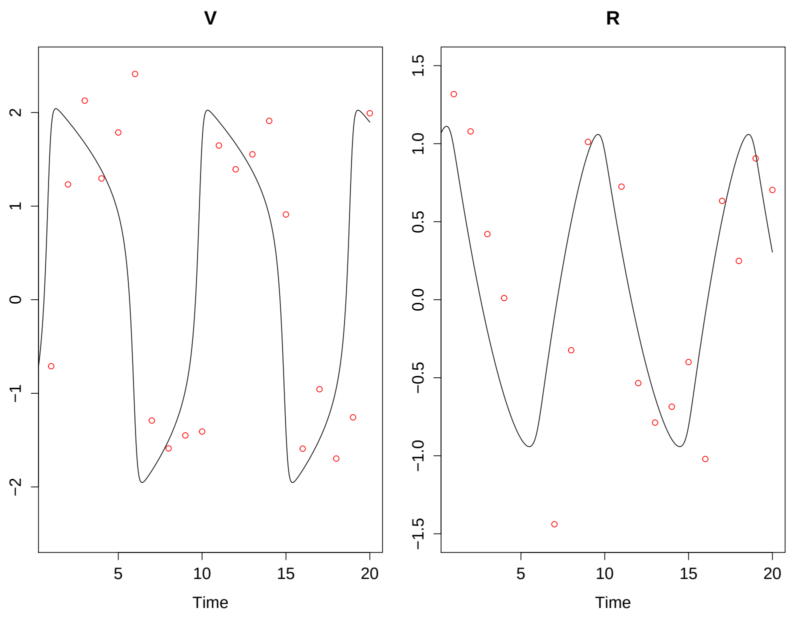


Figure 2.5: Plots of simulated data and solution to the true model with $\alpha = \beta = 0.2$, $\gamma = 3$ and initial conditions $V = -1, R = 1$.

Samples were generated at 20 equally spaced time points from 0 to 20 by numerically solving the differential equations with parameters $\alpha = 0.2$, $\beta = 0.2$, $\gamma = 3.0$ and initial values $V = -1$, $R = 1$. Normal random numbers with mean 0 and standard deviations 0.5 were added to the data as noise. The simulated dataset with the true FHN model are shown in Figure 2.5.

Bootstrap was performed to get confidence intervals for parameters. The residuals from original model were resampled with replacement 500 times and added to the data to get 500 new datasets. Models were refitted each time with a new dataset to get a set of estimations for all parameters. Those sample quantiles are derived as bootstrap

Parameter	**True Value**	**Estimation**	**Confidence Limits**
α	0.2	0.189	[0.13, 0.32]
β	0.2	0.442	[0.06, 0.63]
γ	3.0	2.831	[2.54, 3.05]

Table 2.3: Summary statistics for parameter estimates of the FHN model.

confidence limits.

The least square estimators for parameters α, β, γ and confidence intervals are shown in Table 2.3.

2.4 Monte Carlo Approach

One disadvantage about the Differential Evolution based method is that there is no closed form solution for the confidence intervals and bootstrap procedure is very time-consuming. Monte Carlo method could be utilized as an alternate approach. Since the observation model is

$$\boldsymbol{Y}_i = \boldsymbol{X}(t_i) + \boldsymbol{\epsilon}_i, \qquad i = 1, \dots, n \tag{2.42}$$

If we assume $\boldsymbol{\epsilon}_i$ are independent and normally distributed as $N(0, \sigma^2)$, then the model likelihood could be written as:

$$p(\boldsymbol{Y}|\boldsymbol{X}(\boldsymbol{\theta})) = \prod_{k \in K} N(\boldsymbol{X_k}(\boldsymbol{\theta}), \sigma_k^2 I_{d_k}) \tag{2.43}$$

where k is the index for a total of K state variables. d_k stands for the number of observations for the state variable x_k. $\boldsymbol{\sigma}^2 = (\sigma_1^2, \dots, \sigma_K^2)$ are the observational

Parameter	**True Value**	**Posterior Mean**	**Credible Interval**
α	0.2	0.196	[0.09, 0.29]
β	0.2	0.351	[0.03, 0.66]
γ	3.0	2.822	[2.50, 3.04]

Table 2.4: Summary statistics for parameter estimates of the FHN model by MCMC method.

variances. We could assign prior distribution to $\boldsymbol{\theta} = (\alpha, \beta, \gamma)$. Then the posterior distribution for $\boldsymbol{\theta}$ could be written as:

$$p(\boldsymbol{\theta}|\boldsymbol{Y}) = \pi(\boldsymbol{\theta})p(\boldsymbol{Y}|\boldsymbol{X}(\boldsymbol{\theta})) \tag{2.44}$$

A wide prior $\Gamma(1, 3)$ is assigned to each of the parameters α, β and γ. Since $\boldsymbol{X}(\boldsymbol{\theta})$ has no closed form solution, the posterior distribution could not be computed directly. Thus Markov Chain Monte Carlo (MCMC) method are required to draw samples from the posterior distribution.

The MCMC algorithm used here is the robust adaptive Metropolis sampler. 5000 posterior samples were generated from 51000 iterations of the MCMC algorithm, where the first 1000 values were used as burn-in and the remaining samples were thinned by a factor of 10. Initial values are randomly drawn from the prior distribution. The trace plots for the MCMC algorithms are shown in Figure 2.6, which indicates that the prior distribution is well calibrated since the parameters is shown to have sufficient state changes as the algorithm runs.

Figure 2.7 shows the two fitted model, compared with the true model. We can see that the two fitted model are almost identical and they are both quite close to the true model. This is also demonstrated from Table 2.4. The posterior mean and credible

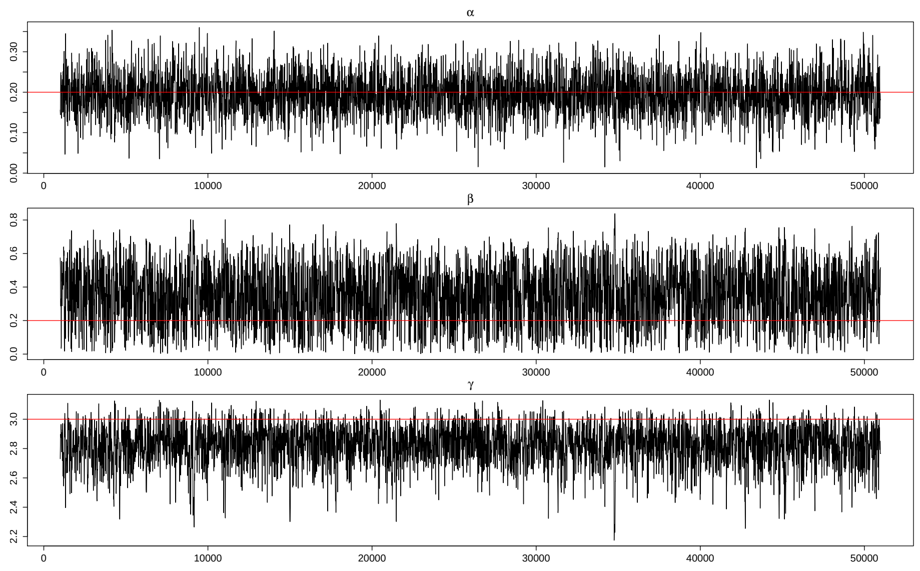


Figure 2.6: Trace plots. The true parameters values are depicted by the red line.

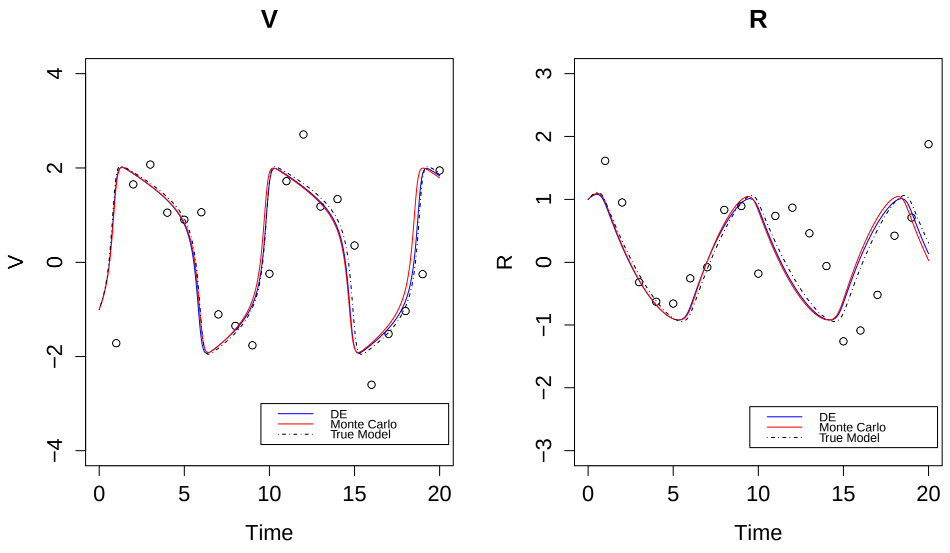


Figure 2.7: Model fit plots for Differential Evolution based method and Monte Carlo method.

intervals are close to the estimation by Differential evolution and bootstrap confidence intervals. A major concern for the Monte Carlo method is the time-consuming issue, since generating Markov chains by calling numerical ODE solvers in each iteration could be quite computationally intensive.

2.5 Numerical Study and Method Comparison

To compare the accuracy, convergence rate and other performances of the numerical ODE integration based methods under different scenarios, we conducted a series of simulation study. The study is based on a two-compartment model. The experiment data is simulated with a given set of true parameters , and random errors with mean zero and known standard deviance is added to simulate the real data that are often contaminated with measurement error. The variance of random errors is increasing in each sequence of simulation study.

The methods that been compared in this section are: the state-of-the-art Gauss-Newton algorithm based method which is recommended by Englezos and Kalogerakis (2000), the proposed Differential Evolution based method and the hybrid method that combines Differential Evolution and Gauss-Newton Algorithm.

Even though Gauss-Newton Algorithm based method is a relatively efficient method since it exhibits quadratic convergence to the optimum, numerical stability and parameter initial value issue is the drawback. It is possible to converge to a local minimum or even fail to converge when starting with a poor initial point. Thus it is crucial to be prudential when it comes to selecting the initial values. It is always helpful if there is an expertise of where the parameter value should be around that we can directly use as initial point. However when there is no such expertise, we propose the idea of searching for appropriate initial value in a given range using Differential Evolution first and then inputting it into the Gauss-Newton algorithm.

We compared the performance of these methods by average relative error (ARE) and mean square error (MSE) in the numerical study as

$$\text{ARE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\hat{\theta}_i - \theta}{\theta}\right| \tag{2.45}$$

where $\hat{\theta}_i$ is the estimator of θ in the ith simulation run and $i = 1, \ldots, n$.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{\theta}_i - \theta)^2 \tag{2.46}$$

The convergence rate is the proportion of times when the algorithm converges, and is also considered as a criteria in the numerical study.

Table 2.5 and 2.6 summarizes the average relative error, mean square error and convergence rate of the three tested estimators based on 1000 simulation runs under different standard deviance scenarios. The true parameter value is: $\theta_1 = 0.670$, $\theta_2 = 0.069$ and $\theta_3 = 0.085$. We assume there's no prior information about the parameters, and the initial points were randomly selected from a uniform distribution from 0 to 1 for the Gauss-Newton algorithm. The initial population is also uniformly generated from 0 to 1 for Differential Evolution algorithm. Same initial value generation strategy is used in the hybrid method, in which we use Differential Evolution to run 10 steps to generate the input for Gauss-Newton algorithm. As we can see from the table, the estimation accuracy tends to get lower as the noise of data getting larger for both Differential Evolution method and proposed hybrid method. The damped Gauss-Newton method is unstable and generates large estimation errors compared to the other two. Moreover, as we mentioned before the original Gauss-Newton algorithm

(σ_1, σ_2)	parameter		$\hat{\theta}_{GN}$	$\hat{\theta}_{DE}$	$\hat{\theta}_{Hybrid}$
(0.001, 0.001)	θ_1	ARE	211.64	5.50	5.46
	θ_2		845.95	5.63	5.59
	θ_3		78.29	0.13	0.13
	Converge		0.70	1.00	1.00
(0.001, 0.002)	θ_1	ARE	217.24	78.87	29.17
	θ_2		141.65	85.29	31.05
	θ_3		152.39	0.29	0.29
	Converge		0.84	1.00	1.00
(0.001, 0.005)	θ_1	ARE	308.38	78.01	57.51
	θ_2		182.38	83.29	57.93
	θ_3		63.47	0.68	0.69
	Converge		0.82	1.00	1.00
(0.002, 0.001)	θ_1	ARE	269.81	16.45	18.14
	θ_2		213.65	17.45	18.12
	θ_3		108.72	0.17	0.20
	Converge		0.78	1.00	1.00
(0.005, 0.001)	θ_1	ARE	212.37	46.31	62.71
	θ_2		166.22	49.13	65.07
	θ_3		97.79	0.34	0.41
	Converge		0.90	1.00	1.00

Table 2.5: Average relative error (ARE) of three methods. $\hat{\theta}_{GN}$ represents estimation using Gauss-Newton algorithm based method, $\hat{\theta}_{DE}$ represents estimation using the proposed Differential Evolution based method and $\hat{\theta}_{Hybrid}$ represent estimation using the proposed hybrid method.

is not guaranteed to converge depends on the initial points and the proposed Hybrid method could overcome this issue. As we can see the average convergence rate for Gauss-Newton Algorithm in this numerical study is around 80 percent and the Hybrid algorithm has almost 100 percent convergence rate.

(σ_1, σ_2)	parameter		$\hat{\theta}_{GN}$	$\hat{\theta}_{DE}$	$\hat{\theta}_{Hybrid}$
(0.001, 0.001)	θ_1	MSE	8.70	0.23e-2	0.22e-2
	θ_2		9.58	0.26e-6	0.25e-6
	θ_3		0.18e-01	0.18e-7	0.19e-7
	Converge		0.70	1.00	1.00
(0.001, 0.002)	θ_1	MSE	4.39	4.33	0.24
	θ_2		0.03	0.05	0.003
	θ_3		0.61e-01	0.15e-6	0.12e-6
	Converge		0.84	1.00	1.00
(0.001, 0.005)	θ_1	MSE	9.73	2.27	0.54
	θ_2		0.04	0.03	0.01
	θ_3		0.10e-01	0.51e-6	0.50e-6
	Converge		0.84	1.00	1.00
(0.002, 0.001)	θ_1	MSE	13.03	0.05	0.07
	θ_2		0.15	0.07e-2	0.07e-2
	θ_3		0.31e-01	0.36e-7	0.10e-6
	Converge		0.78	1.00	1.00
(0.005, 0.001)	θ_1	MSE	6.85	0.24	0.43
	θ_2		0.74e-1	0.29e-2	0.52e-2
	θ_3		0.27e-01	0.13e-6	0.22e-6
	Converge		0.90	1.00	1.00

Table 2.6: Mean square error (MSE) of three methods. $\hat{\theta}_{GN}$ represents estimation using Gauss-Newton algorithm based method, $\hat{\theta}_{DE}$ represents estimation using the proposed Differential Evolution based method and $\hat{\theta}_{Hybrid}$ represent estimation using the proposed hybrid method.

Chapter 3 Estimation Method Based on Fourier Basis Smoothing

3.1 Introduction

Since the estimation methods based on numerical integration of differential equations require solving ODE initial value problems numerically in a repeated manner, usually it is computationally intensive and time consuming on large datasets. Some alternative methods have been proposed. Varah (1982) proposed a two stage estimation method for differential equations using cubic spline. Ramsay et al. (2007) proposed a parameter estimation method based on a penalized data smoothing methods along with a generalization of profiled estimation. Liang and Wu (2012) proposed a two-step method and estimate the derivative using local polynomial regression.

One issue with the cubic spline or penalized spline based method is that when the sample size is small, it is difficult to pick the knots and do smoothing. Due to the nature of pharmacokinetic study, usually the number of observations is limited. While on the other hand, using Fourier basis would avoid the procedure of choosing knots, thus more flexible to small sample dataset.

Motivated by Varah (1982) and Ramsay et al. (2007), we propose an estimation method using Fourier basis smoothing and pseudo least square estimation. It has the advantage of being less time consuming, and it does not have the initial value problem that is presented when using ODE numerical solvers. Comparing to spline based method, it works better on periodic or near periodic ODE model functions.

Moreover, the derivative with respect to time is simple and fast to get using Fourier transformation, and it plays an important role in the estimation procedure.

3.2 Model and Estimation Method

Fourier basis is formed by a sequence of sine and cosine with increasing frequency and equation is:

$$x(t) = \sum_{k=1}^{K} c_k \Phi_k(t) \tag{3.1}$$

Where $\Phi_1(t) = 1$, $\Phi_2(t) = \sin(\omega t)$, $\Phi_3(t) = \cos(\omega t)$, $\Phi_4(t) = \sin(2\omega t)$, $\Phi_5(t) = \cos(2\omega t)$, $\ldots$, $\Phi_{K-1}(t) = \sin(\frac{(K-1)\omega t}{2})$, $\Phi_K(t) = \cos(\frac{(K-1)\omega t}{2})$, where constant $\omega = 2\pi/P$ defines the period P of oscillation of the first sine/cosine pair. The dimension of basis K is always odd.

To fully declare a Fourier basis system we need to define the dimension of basis K, and the period width P. K needs to be properly chosen so that the basis can capture enough information while not over fitting the data, and P can be decided by a preliminary study. Often the default is the range of t values spanned by the data to indicate a non-periodic function.

The Fourier basis system is straightforward and fast to implement. It is very appropriate for describing periodic or near periodic data by natural, and it has decent computational properties. Moreover, it is not necessary to choose the cutting point therefore it works more naturally on small sample dataset. So the Fourier basis is selected as the data smoothing technique in the parameter estimation procedure.

To further describe the problem, we refer the definition of compartment models in Section 1.2 and define the model equation to be:

$$Y_{ij} = X_i(t_j) + \epsilon_{ij}, \qquad i = 1, \ldots, N, \quad j = 1, \ldots, n_i \tag{3.2}$$

Where $Y_{i1}, \ldots, Y_{in_i}$ are a sequence of observed data on successive time points $t_1, \ldots, t_{n_i}$ for compartment i, and $\epsilon_{i1}, \ldots, \epsilon_{in_i}$ are by assumption uncorrelated errors with zero mean and variance-covariance matrix $\boldsymbol{\Sigma}_i$. There are two ways to define the residuals in the observation model: one is additive error, assuming $\boldsymbol{\Sigma}_i = \boldsymbol{I}\sigma^2$; the other is proportional error, which assumes the noise is proportional to the expected value, then $\boldsymbol{\Sigma}_i = \boldsymbol{X_i}\sigma^2$, where $\boldsymbol{X_i} = \text{diag}(X_i(t_1), \ldots, X_i(t_{n_i}))$. It may be appropriate to apply a variance stabilization transformation, such as log or square root, to both side of the model equation. Carroll and Ruppert (1988) gives a detailed illustration on how appropriate transformations could remove heteroscedasticity when the variance is a function of the mean and how the convexity, or concavity, of a transformation determines its effect upon skewness, therefore the intelligent use of transformation requires understanding of their effects upon non-normality and heterogeneity of variance components.

$X_i(t_j)$ is defined by solution of the following differential equation on the corresponding compartment i evaluated at given time point t_j

$$\frac{d\boldsymbol{X}(t)}{dt} = f(\boldsymbol{X}, t, \boldsymbol{\beta}), \qquad \boldsymbol{X}(t_0) = \boldsymbol{X}_0 \tag{3.3}$$

Sometimes we refer (3.2) as observation model and (3.3) as differential equation model. Here $\boldsymbol{X}(t)$ and $\boldsymbol{Y}(t)$ could be scaler or vector, depending on the number of compartments or number of observed compartments in the system. The initial condition $(t_0, \boldsymbol{X}_0)$ could be known or unknown. When it's unknown we usually treat it as an additional parameter. When multiple compartments are observed, $\boldsymbol{X}(t) = (X_1(t), \ldots, X_N(t))$.

Assuming the additive error structure and defining n_{tot} be the total number of observations, the model likelihood can be derived as:

$$L(\boldsymbol{\beta}, \sigma^2) = \prod_{i=1}^{n} P(\boldsymbol{Y}_i | \boldsymbol{\beta}, \sigma^2) \tag{3.4}$$

$$= (2\pi\sigma^2)^{-n_{tot}/2} \exp(-\frac{1}{2\sigma^2} \sum_{i=1}^{N} |\boldsymbol{Y}_i - \boldsymbol{X}_i(\boldsymbol{t}, \boldsymbol{\beta})|^2) \tag{3.5}$$

For parameter estimation, we use a computation approach to first smooth the data then minimize the derivative error. The purpose of the Fourier basis system is to smooth the data, namely, to remove the measurement error from the data, as well as to evaluate rates of change (derivatives) which play an important role in next stage.

- Step 1: Build a Fourier basis for data smoothing. For each t_i, obtain the Fourier estimate $\hat{\mathcal{F}}(t_i)$ and its derivate $\hat{\mathcal{F}}'(t_i)$.
- Step 2: Minimize the so-called pseudo least square objective function

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} [\hat{\mathcal{F}}'(t_i) - f(\hat{\mathcal{F}}(t_i), t_i, \boldsymbol{\beta})]^2 \tag{3.6}$$

Liang and Wu (2012) proves the pseudo least square estimator has good asymptotic properties such as consistency and asymptotically normal. The estimator has closed form if f is a linear function and if it's nonlinear, Gauss-Newton, Nelder-Mead or another optimization method could be used.

The purpose of Fourier basis smoothing in step 1 is to remove only the measurement noises from the observational data, and the dimension of Fourier basis should be the lowest possible that fits the measurements satisfactorily. If the ODE model is indeed the true model then computed output vector from the ODE model should correspond to the error-free measurements. So a convenient way to validate if the Fourier smoothing is satisfactory is to plot the raw data, the smoothed data and the output from the final ODE model in one graph for visualization. If the initial fit from Fourier basis smoothing is reasonably close to the final ODE model fit, and the residuals appear to be normal, then the Fourier basis smoothing is done in a correct way. If not, it is always a valid option to go back and redo the Fourier smoothing.

3.3 Simulation Study

To test the performance of the proposed method on complicated ODE model structures, especially on those differential equations that do not have analytical solutions, a simulation study is conducted. The FitzHugh-Nagumo (FHN) model is a two-dimensional model that describes the voltage potential across the cell membrane of the axon of giant squid neurons. The motivation for the FHN model was to isolate conceptually the essentially mathematical properties of excitation and propagation from the electrochemical properties of sodium and potassium ion flow. The ODE

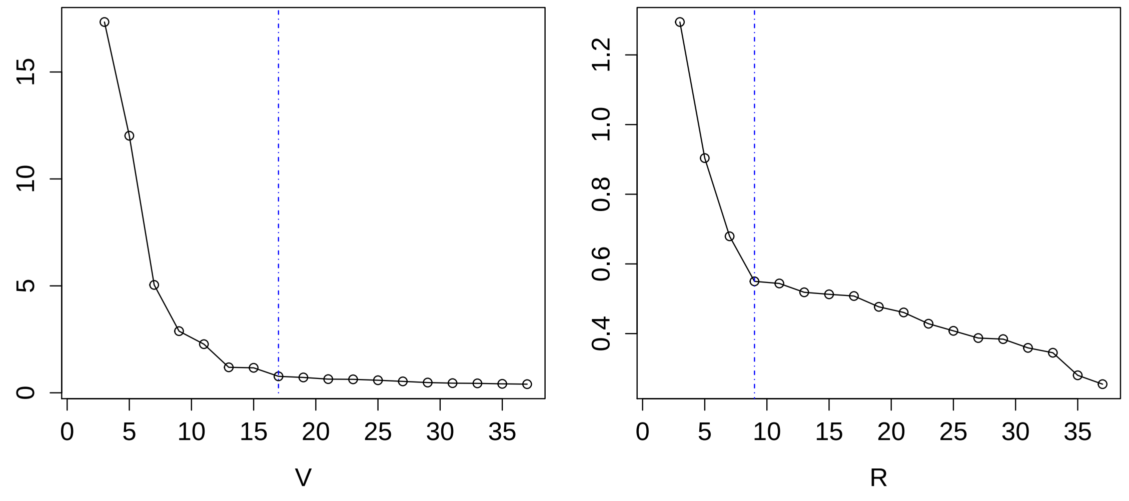


Figure 3.1: Number of parameters versus residual sum of squares (RSS) for V and R.

model structure of FHN model is:

$$\frac{dV}{dt} = \gamma(V - \frac{V^3}{3} + R) \tag{3.7}$$

$$\frac{dR}{dt} = -\frac{1}{\gamma}(V - \alpha + \beta R) \tag{3.8}$$

where V is the membrane potential that depends on a recovery variable R, and $\alpha, \beta, \gamma \in IR^+$ are model parameters.

As an example, 80 observations are simulated from time = 0 to time = 20 by increment of 0.25. Using the proposed two step estimation method with Fourier transformation. After a preliminary study we set period value $P = 9$. Then based on Figure 3.1 we checked the residual sum of squares (RSS) versus dimension of Fourier basis. We also checked the plots of the actual fit to make sure there is no over-fitting. If the Fourier curve overfits the data, not only the initial fit would be affected by random observation errors, but also the estimation of time derivatives would potentially be seriously biased, which would cause the pseudo least square function in step 2 to be biased and unreliable. As we know that as the dimension of basis gets larger, the RSS is always getting smaller, as shown in Figure 3.1. It is appropriate to stop at the number of dimension where the RSS is not dramatically decreasing, and no signs of over fitting is detected. Meanwhile, the plot of derivatives is another tool in determining the validity of the initial fit. In the example we select the dimension of basis for V and R to be 17 and 9 respectively. The initial Fourier basis smoothing is shown on Figure 3.2.

Once we obtain the Fourier estimate $\hat{\mathcal{V}}(t_i)$, $\hat{\mathcal{R}}(t_i)$ and its derivate $\hat{\mathcal{V}}'(t_i)$, $\hat{\mathcal{R}}'(t_i)$. We go to step 2 and obtain the pseudo least square estimator for α, β and γ.

$$
\begin{aligned}
\{\hat{\alpha}, \hat{\beta}, \hat{\gamma}\} = \underset{\alpha,\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} \Bigg\{ \left[\hat{\mathcal{V}}'(t_i) - \gamma \left(\hat{\mathcal{V}}(t_i) - \frac{\hat{\mathcal{V}}^3(t_i)}{3} + \hat{\mathcal{R}}(t_i) \right) \right]^2 \\
+ \left[\hat{\mathcal{R}}'(t_i) + \frac{\hat{\mathcal{V}}(t_i) - \alpha + \beta \hat{\mathcal{R}}(t_i)}{\gamma} \right]^2 \Bigg\}
\end{aligned}
\tag{3.9}
$$

The final fit is shown on Figure 3.3. Compared with the initial fit, the final fit is more smooth and it better represents the dynamics of the ODE model, and is less influenced by the observation noise. These final plots show relatively high accuracy and since the method bypasses calling a numerical ODE solver, the estimation speed is much improved.

As we can imagine, the accuracy of this method could not surpass that of the numerical ODE integration based method, especially for limited sample size situa-

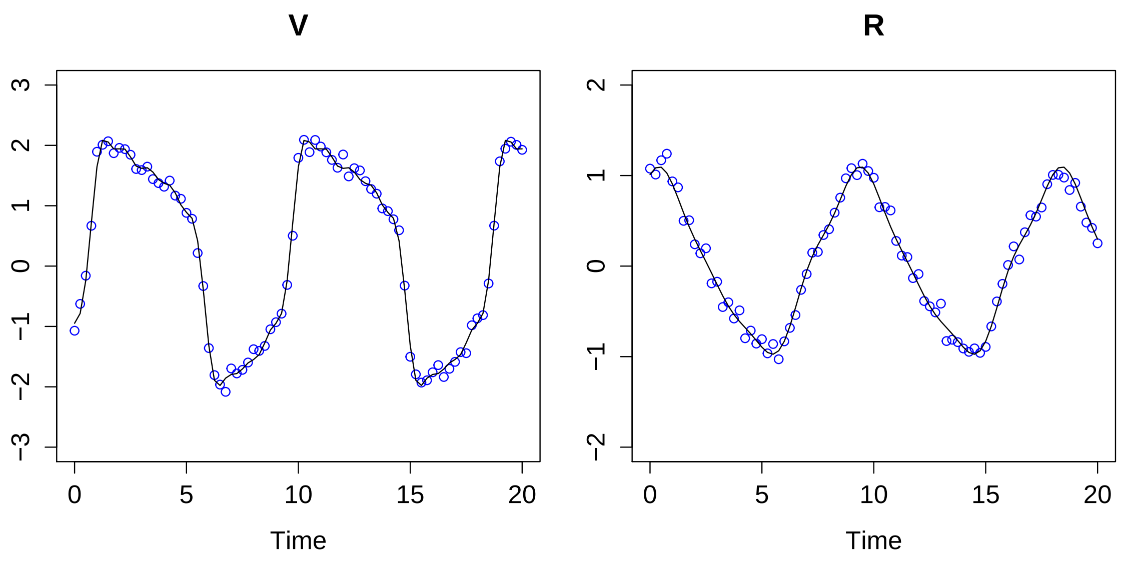


Figure 3.2: Initial fit using Fourier basis smoothing.

tion, or heavily noised data. However, in ideal cases the results could be as good, with much less computation time to gather the results, and it could also provide useful preliminary information for numerical ODE integration based method. In the following simulation studies, a hybrid method is proposed, using a two stage estimation schema, whereas in stage one the estimation based on Fourier basis smoothing is performed, and in stage two the Gauss-Newton based method is conducted, using the output of stage one as initial values for the parameters.

A series of simulation studies has been conducted to assess the accuracy and validity of the proposed estimation algorithm based on the FitzHugh-Nagumo (FHN) model. Different simulation scenarios is set up, with number of observations varies from 20 to 80 and standard deviation of observation noises changes from 0.05 to 0.1. For each scenario, 1000 datasets with observation noises are simulated, and the mean, standard deviation, average relative error (ARE), mean square error (MSE) and mean

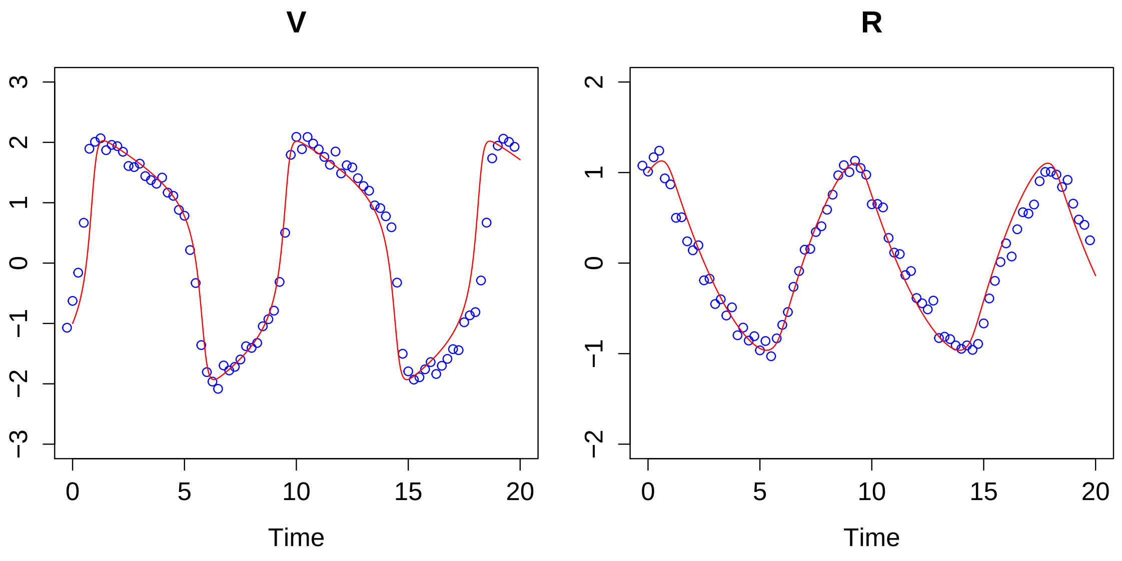


Figure 3.3: Final fit using numerical ODE solver with estimated parameters.

computation time is recorded in Table 3.1.

First of all, as we can see from the results, the running time for the estimation algorithm is quite low since we bypassed the numerical ODE solver; secondly, it is noticed that as the number of observations going up, the estimation is getting more accurate and stable with less ARE and MSE value. This is because when more data points are observed, the Fourier basis would better smooth and represent the data curvature, which leads to a better approximate of the true model. In the end, we can see that as the random observation noise gets larger, the results get less accurate and the estimation variance gets larger. Moreover, it can be assumed that for small sample size situations, this method would not work as well as the numerical ODE integration based method, as the true longitudinal curve would be difficult to capture accurately. But even in that situation, this method is still valuable because the estimation is fast to get and it could serve as the initial value for the numerical ODE integration based

Num Obs	(σ_1, σ_2)	Parameter	Mean	Std	ARE	MSE	Time
		α	0.34	0.011	0.170	1.3e-3	
20	$(0.05, 0.05)$	β	0.021	0.039	0.894	3.3e-2	0.043
		γ	2.28	0.081	0.238	5.1e-1	
		α	0.243	0.025	0.221	2.4e-3	
20	$(0.1, 0.1)$	β	0.0093	0.077	0.915	4.3e-2	0.043
		γ	2.22	0.156	0.259	6.2e-1	
		α	0.209	0.011	0.058	2.1e-4	
40	$(0.05, 0.05)$	β	0.177	0.030	0.154	1.4e-3	0.051
		γ	3.01	0.101	0.0267	1.0e-2	
		α	0.212	0.020	0.097	5.7e-4	
40	$(0.1, 0.1)$	β	0.172	0.058	0.255	4.2e-3	0.050
		γ	2.900	0.183	0.0566	4.3e-2	
		α	0.211	0.007	0.057	1.8e-4	
80	$(0.05, 0.05)$	β	0.183	0.022	0.113	7.7e-4	0.060
		γ	2.88	0.054	0.0408	1.8e-2	
		α	0.214	0.015	0.083	4.1e-4	
80	$(0.1, 0.1)$	β	0.179	0.043	0.188	2.3e-3	0.063
		γ	2.78	0.099	0.0727	5.6e-2	

Table 3.1: Simulation results for the method based on Fourier basis smoothing. The estimation mean, standard deviation, average relative error (ARE), mean square error (MSE) and mean computation time (in second) is presented for each parameter under different scenarios.

method which could lead to a more accurate and stable estimation.

To compare the performance of the Fourier basis smoothing based method with the numerical ODE integration based method, a series of simulation study is conducted. The FHN model is used as the settings of the simulation, two different scenarios for the random measurement errors are implemented. The three methods being tested are: Fourier basis smoothing based method (FS), Gauss-Newton based method (GN) and a hybrid method, in which the Fourier basis smoothing based method is conducted first and then its output is used as the initial value for the Gauss Newton based method. For the Gauss-Newton method, random numbers are generated as the starting parameter values for the algorithm. Based on the validity of the model, we generated α and β from unif$(0, 1)$ distribution, and γ from unif$(1, 10)$.

(σ_1, σ_2)	Parameter	Method	Mean	Std	ARE	MSE	Time
$(0.05, 0.05)$	α	FS	0.209	0.011	0.058	2.1e-4	0.051
		GN	0.238	0.132	0.296	1.8e-2	1.97
		Hybrid	0.201	0.006	0.019	3.3e-5	1.56
	β	FS	0.177	0.030	0.154	1.4e-3	0.051
		GN	0.307	0.213	0.661	5.7e-2	1.97
		Hybrid	0.199	0.028	0.098	7.6e-4	1.56
	γ	FS	3.01	0.101	0.0267	1.0e-2	0.051
		GN	3.38	1.20	0.171	1.6e+1	1.97
		Hybrid	3.00	0.023	0.004	5.3e-4	1.56
$(0.1, 0.1)$	α	FS	0.212	0.020	0.097	5.7e-4	0.050
		GN	0.251	0.139	0.360	2.2e-2	1.94
		Hybrid	0.202	0.015	0.046	2.3e-4	1.61
	β	FS	0.172	0.058	0.255	4.2e-3	0.050
		GN	0.332	0.230	0.793	7.0e-2	1.94
		Hybrid	0.200	0.054	0.198	2.9e-3	1.61
	γ	FS	2.900	0.183	0.0566	4.3e-2	0.050
		GN	3.50	1.369	0.223	2.1e+1	1.94
		Hybrid	2.986	0.074	0.0104	5.7e-3	1.61

Table 3.2: Simulation results for the Fourier basis smoothing based method (FS), Gauss-Newton based method (GN) and a hybrid method (Hybrid). The estimation mean, standard deviation, average relative error (ARE), mean square error (MSE) and mean computation time is presented for each parameter under different scenarios.

It is shown in the table that the Gauss-Newton based method is quite unstable and inaccurate, the convergence rate is only 0.773 in the 1000 times of simulation for the $(\sigma_1, \sigma_2) = (0.05, 0.05)$ setting, and 0.769 for $(\sigma_1, \sigma_2) = (0.1, 0.1)$, and the ARE and MSE values are both the highest among these three methods. For the Fourier basis smoothing based method, the estimation is decent with the highest computation speed, and from the table its mean computation time is significantly smaller than the others. Moreover, there is still room for improvement in terms of estimation accuracy for FS method. The hybrid method could almost completely resolve the convergence issue compared with using random starting points for the Gauss-Newton based method since the convergence rate increases to 1 in the simulation study. Meanwhile, it could improve the estimation accuracy for the Fourier transformation based method since the ARE and MSE are smaller than FS, and both way smaller than the random initial value cases. In the end, it is interesting to see that the computation time for the hybrid method is actually smaller than that of the Gauss-Newton based method with random initial value, since a better starting point could make the algorithm converge to the minimum in fewer iterative steps, and it could make up for the time used to run the Fourier basis smoothing method.

3.4 Real Data Application

Wu et al. (2011) and Ding and Wu (2014) present a mechanistic differential equation model that describes the expansion, trafficking and disappearance of activated virus-specific CD8$^+$ T cells in lymph nodes (T_E^m), spleens (T_E^s) and lungs (T_E^l) of mice during primary influenza A Virus (IAV) infection with an intensive sampling procedure. The dataset could be downloaded from the url in (Ding and Wu). The ODE model could be written as

$$\frac{\mathrm{d}}{\mathrm{d}t}T_E^m = [\rho_m D^m(t-\tau) - \delta_m]T_E^m - (\gamma_{ms} + \gamma_{ml})T_E^m \tag{3.10}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}T_E^s = [\rho_s D^s(t-\tau) - \delta_s]T_E^s - \gamma_{sl}T_E^s + \gamma_{ms}T_E^m \tag{3.11}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}T_E^l = \gamma_{ml}T_E^m + \gamma_{sl}T_E^s - \delta_l T_E^l \tag{3.12}$$

Based on Wu et al. (2011), D^m represents the number of mature Ag-bearing dendritic cells (DC) in mediastinal lymph node (MLN); D^s is the number of nature DCs in spleen, t is the time delay of the effects in DCs on the CD8$^+$ T cell activation; ρ_m and ρ_s are the proliferation rates of CD8$^+$ T cell simulated by DCs in MLN and spleen, respectively; δ_m, δ_w and δ_l are the loss rate in MLN, spleen and lung respectively; γ_{ms}, γ_{ml} and $\gamma_s l$ denote the migration rate from MLN to spleen, from MLN to lung and from spleen to lung, respectively.

A total of $n = 77$ data points at 9 distinct time points for T_E^m, T_E^s, T_E^l are available, as well as data for D^m. Data for D^s is not available, and it is approximated using D^m based on Wu et al. (2011). The smoothed estimates of D^m are used in the analysis. The time delay is set to $\tau = 3.08$, and parameters δ_m, δ_s and γ_{ml} are set to 0 based on Wu et al. (2011). A log transformation to the observations are implemented to stabilize the measurement error variance. Let $(X_1, X_2, X3) = (\log(T_m^E), \log(T_E^s), \log(T_E^l),$

Parameter	**Initial**	**Estimation**	**Std. Error**
ρ_m	1.29e-5	1.66e-5	4.86e-6
ρ_s	1.58e-5	4.49e-5	3.61e-6
δ_l	4.65e-2	3.97	7.01e-1
γ_{ms}	1.40e-1	1.57e-1	6.36e-2
γ_{sl}	2.42e-9	4.95e-1	6.06e-2

Table 3.3: Parameter estimation for the CD8$^+$ T cell data model.

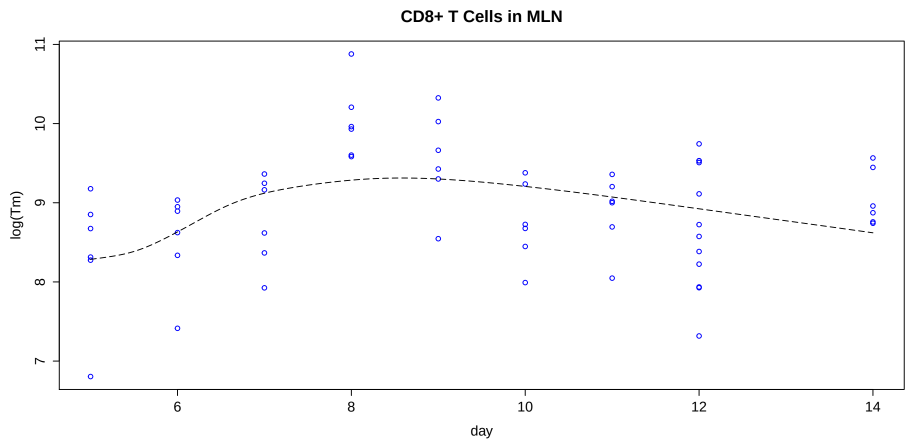
then the ODE model could be written as

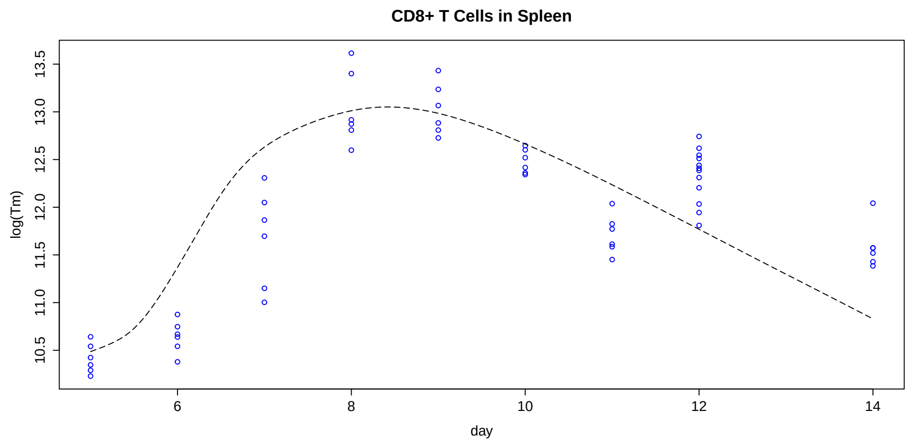
$$\frac{\mathrm{d}}{\mathrm{d}t}X_1 = \rho_m D^m(t-\tau) - \delta_m] - \gamma_{ms} - \gamma_{ml} \tag{3.13}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}X_2 = \rho_s D^s(t-\tau) - \delta_s - \gamma_{sl} + \gamma_{ms}\exp(X_1 - X_2) \tag{3.14}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}X_3 = \gamma_{ml}\exp(X_1 - X_3) + \gamma_{sl}\exp(X_2 - X_3) - \delta_l \tag{3.15}$$

The Fourier Basis Smoothing method is utilized to conduct a pre-analysis, and generates estimations for the parameters. Then using the output, the Gauss-Newton algorithm based method is implemented and final results are obtained. The final model fit is shown in Figure 3.4. The estimation results shown in Table 3.3 is consistent with Wu et al. (2011), which indicates that the proposed Hybrid method is valid and efficient in estimating the ODE model in this real data analysis.





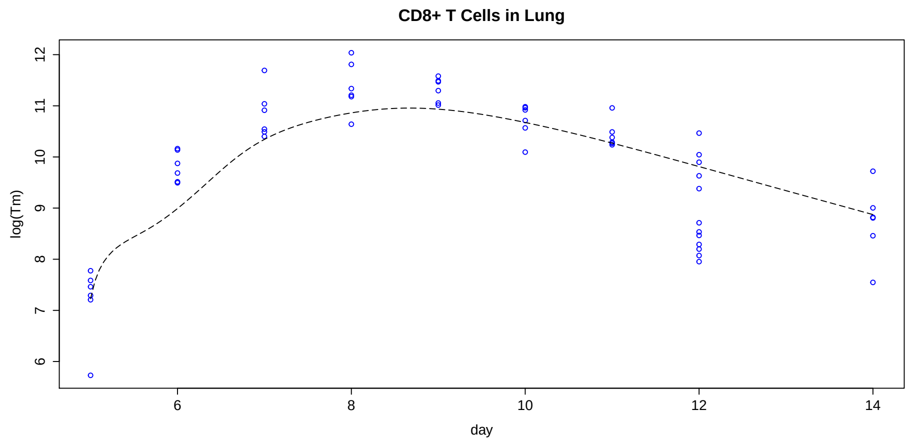


Figure 3.4: Data of CD8$^+$ T cell in MLN, spleen and lung with fitted curve respectively

Chapter 4 Mixed Model for Population Dynamical Systems

Previously all the models are based on longitudinal data for a single observation unit. In this section we would like to expand the model structure to include multiple experiment units (individuals) into a population-based full model. In this way we can characterize the typical parameter values in the population and the extent of their variation. The idea of mixed effect modeling is implemented in this chapter.

4.1 Introduction

The mixed effect model is a popular tool in longitudinal data analysis of biometric studies, and during the development process a lot of different methodologies have been proposed and implemented in the linear mixed model framework. Furthermore, nonlinear mixed effect models has been developed that focuses on features or mechanisms that underlie individual profiles of repeated measurements of the response and how these vary in the population. Population dynamical system model could be seen as an extension of nonlinear mixed effect models. A lot of methodologies and schema in nonlinear mixed models could be adapted and applied in population dynamical system models.

Traditional estimation method in nonlinear mixed effect model, especially population Pharmacokinetic and Pharmacodynamic models are based on the linearization of the nonlinear log likelihood function. The most representative method include first-order method (FO), first-order conditional estimation method (FOCE) and Laplacian approximation. They are still quite popular and useful for many nonlinear mixed model analysis problems. The common software packages are ‘nlme’ in R (Pinheiro et al., 2006) and proc nlmixed in SAS. A potential issue with these methods is that they are based on likelihood approximation, and therefore some properties for MLE, such as the standard deviation derived from the Fisher information matrix, or the likelihood ratio test for nested models do not hold for these methods in some situations (Meza et al., 2007).

More recently developed methods are the ’exact likelihood’ methods, where we maximize the likelihood directly, using deterministic or stochastic approximation to the integrals. Deterministic approximation methods include Gaussian quadrature and adaptive Gaussian quadrature. Stochastic approximation methods include Monte Carlo Expectation Maximization (MCEM) algorithm (Wei and Tanner, 1990) and Stochastic Approximation EM algorithm (Delyon et al., 1999) based method. Since no linear approximation is involved in these methods, the parameters obtained are true maximum likelihood estimates that all statistical properties of MLE could be applied.

The SAEM algorithm is a stochastic iterative algorithm for calculating the maximum likelihood estimator (MLE) in the general setting of incomplete data models. Suppose for individual i there is a sequence of observations y_i. Assuming the number of individuals in the dataset to be N and the set of parameters to be $\mathbf{\Phi}$, let b_i be the random effect vector for individual i, and the function $Q(\mathbf{\Phi})$ be the expectation of full log likelihood, then at each iteration m, the following steps are performed:

1. Simulation step: for $i = 1, 2, \ldots, N$, draw $b_i^{(m)}$ from the conditional distribution $P(\cdot|y_i; \mathbf{\Phi}_{m-1})$

2. Stochastic approximation: update $Q_m(\mathbf{\Phi})$ as

$$Q_m(\mathbf{\Phi}) = Q_{m-1}(\mathbf{\Phi}) + \gamma_m(\log p(\boldsymbol{y}, \boldsymbol{b}^{(m)}; \mathbf{\Phi}) - Q_{m-1}(\mathbf{\Phi})) \tag{4.1}$$

   where (γ_m) is a decreasing sequence of positive numbers such that $\gamma_1 = 1$, $\sum_{m=1}^{\infty} \gamma_m = \infty$ and $\sum_{k=1}^{\infty} \gamma_m^2 < \infty$.

3. Maximization step: update $\mathbf{\Phi}_m$ according to

$$\mathbf{\Phi}_m = \arg\max Q_m(\mathbf{\Phi}) \tag{4.2}$$

The stochastic approximation step seems quite complex, but the implementation will be much simplified when the complete model belongs to a regular exponential family, whereas we can just update the sufficient statistic of the complete model instead of updating the Q function, The sufficient statistic contains all needed information to compute any estimation. SAEM uses a recycling of simulated variates from one iteration to another, thus it is more computationally efficient, and has better convergence rate than the MCEM methods.

Most of the studies for mixed models are based on maximum likelihood estimation. However, it is well known that the MLE for variance component of the random effect parameters can be biased downwards since it does not adjust for the degree of freedom lost by estimating the fixed effect. Restricted maximum likelihood (REML) estimation could correct this problem by maximizing the likelihood of a set of residual contrasts. The original REML formation is only applied to linear mixed modeling, as the zero-mean residual contrasts are uncommon in nonlinear models.

There are two ways to solve this problem. One is to correct the bias in the profile score function of the variance components. The main step includes: integrate out the random effects, use simulation to estimate the bias and then adjust for the bias. But this method could be extremely time-consuming. Another more common method is to integrate out the fixed effects, which could be done using Gaussian quadrature or via stochastic methods (Meza et al., 2007). In the following section we presents a method of REML implementation for population dynamical system models within an exact likelihood estimation scheme, using SAEM, MCMC and numerical ODE solvers.

4.2 Model Structure

We consider the following model structure:

$$y_{ij} = \eta_i(t_{ij}) + \epsilon_{ij}, \qquad i = 1, \dots, N, \qquad j = 1, \dots, n_i \tag{4.3}$$

where the within-group errors are i.i.d Gaussian random variables $\epsilon_{ij} \sim N(0, \sigma^2)$. t_{ij} is the jth time point for observation unit i. η_i is the solution of a set of ODE equations:

$$\frac{\mathrm{d}\eta}{\mathrm{d}t} = f(\eta, t, \theta_i), \qquad \eta(t_0) = \eta_{i0}, \qquad i = 1, \dots, N \tag{4.4}$$

Parameter θ_i is modeled by:

$$\theta_i = \boldsymbol{X}\beta + b_i, \qquad b_i \sim N(0, \boldsymbol{\Gamma}) \tag{4.5}$$

Where β represents fixed effect coefficient and b_i is the individual Gaussian random effect. $\boldsymbol{X}$ is the known design matrix. Using this hierarchical structure the dynamical system model could combine with mixed effect model to explore the correlation between individual features, such as age, sex, etc, and parameter values such as transmission rate, etc.

4.3 Estimation Method

The first estimation method we propose is to incorporating numerical ODE solvers to Stochastic Approximation EM algorithm, considering the random effect as missing data and building an iterative algorithm. We consider the complete data set to be $\boldsymbol{W} = (\boldsymbol{y}, \boldsymbol{b})$ and the parameter set we would like to estimate is $\Phi = (\beta, \sigma, \boldsymbol{\Gamma})$.

The complete data likelihood can be written as:

$$L_{\boldsymbol{W}}(\boldsymbol{\Phi}) = \prod_{i=1}^{N} p(\boldsymbol{y}_i | \beta, b_i, \sigma^2) \cdot p(b_i | \boldsymbol{\Gamma}) \tag{4.6}$$

The observed data likelihood can be written as:

$$L_{\boldsymbol{y}}(\boldsymbol{\Phi}) = \int \prod_{i=1}^{N} p(\boldsymbol{y}_i | \beta, b_i, \sigma^2) \cdot p(b_i | \boldsymbol{\Gamma}) d\boldsymbol{b} \tag{4.7}$$

For maximum likelihood estimation, the goal is to estimate $\Phi = (\beta, \sigma, \boldsymbol{\Gamma})$ by maximizing the observed data likelihood $L_{\boldsymbol{y}}(\boldsymbol{\Phi})$. Expectation Maximization (EM) algorithm, first proposed by Dempster et al. (1977), is one representative way of modeling incomplete data. To use EM algorithm, we need to set up the complete log-likelihood:

$$\log L_{\boldsymbol{W}} = \sum_{i=1}^{N} \{\log p(\boldsymbol{y}_i|\beta, b_i, \sigma^2) + \log p(b_i|\boldsymbol{\Gamma})\} \tag{4.8}$$

As we notice that β and σ are only involved in the first term and $\boldsymbol{\Gamma}$ is only in the second. Moreover the $\boldsymbol{y}_i$s are independent given b_i is known. So the basic schema of the EM algorithm can be written as:

1. Select starting value for $\beta^{(0)}$, $\sigma^{(0)}$ and $\boldsymbol{\Gamma}^{(0)}$. Set $m = 0$.
2. E step: compute the expected value $\mathrm{E}[\log L_{\boldsymbol{W}}|\boldsymbol{y}]$ under $\beta^{(m)}$, $\sigma^{(m)}$, $\boldsymbol{\Gamma}^{(m)}$.
3. M step: find $\beta^{(m+1)}$, $\sigma^{(m+1)}$, $\boldsymbol{\Gamma}^{(m+1)}$ that maximize $\mathrm{E}[\log L_{\boldsymbol{W}}|\boldsymbol{y}]$. .
4. Return $\beta^{(m+1)}$, $\sigma^{(m+1)}$, $\boldsymbol{\Gamma}^{(m+1)}$ as MLE if convergence is reached, otherwise set $m = m + 1$ and go to step 2.

In fact, in step 3 the maximization for $\beta^{(m+1)}$, $\sigma^{(m+1)}$ and that for $\boldsymbol{\Gamma}^{(m+1)}$ can be separated because the log-likelihood is separable. So step 3 can be divided into 2 parts: first find $\beta^{(m+1)}$, $\sigma^{(m+1)}$ that maximize $\mathrm{E}[\log p(\boldsymbol{y}|\boldsymbol{b}, \beta, \sigma)|\boldsymbol{y}]$, then find $\boldsymbol{\Gamma}^{(m+1)}$ that maximizes $\mathrm{E}[\log p(\boldsymbol{b}|\boldsymbol{\Gamma})|\boldsymbol{y}]$.

EM algorithm alternates between performing the E step and M step until convergence is reached. In the E step, the expected value of the full log likelihood function is also called the Q function. It has been proved that the observed likelihood increases in each iteration, and it can converge to the maximum likelihood estimation under mild regularity conditions.

In general the expectation in step 2 is difficult to compute in closed form, so approximation method is necessary. It is possible to apply Metropolis-Hasting (MH) algorithm to obtain an approximation of the full condition probability $p(\boldsymbol{b}|\boldsymbol{y})$. To implement the MH algorithm, a candidate distribution for $\boldsymbol{b}$ is needed, from which we can draw candidate samples. Here, $p(\boldsymbol{b}|\boldsymbol{\Gamma})$ is selected as candidate distribution. Let $\boldsymbol{b}$ as the previous draw in MH and using candidate distribution we generate a new value $\boldsymbol{b}^\star$. The acceptance probability $p(\boldsymbol{b}, \boldsymbol{b}^\star) = \min\{1, A(\boldsymbol{b}, \boldsymbol{b}^\star)\}$, where $A(\boldsymbol{b}, \boldsymbol{b}^\star)$ can be derived as

$$A(\boldsymbol{b}, \boldsymbol{b}^\star) = \frac{p(\boldsymbol{b}^\star|\boldsymbol{y}, \beta, \sigma, \boldsymbol{\Gamma})p(\boldsymbol{b}|\boldsymbol{\Gamma})}{p(\boldsymbol{b}|\boldsymbol{y}, \beta, \sigma, \boldsymbol{\Gamma})p(\boldsymbol{b}^\star|\boldsymbol{\Gamma})} \tag{4.9}$$

$$= \frac{\prod_{i=1}^N p(\boldsymbol{y}_i|\beta, b_i^\star, \sigma^2) \cdot p(b_i^\star|\boldsymbol{\Gamma}) \prod_{i=1}^N p(b_i|\boldsymbol{\Gamma})}{\prod_{i=1}^N p(\boldsymbol{y}_i|\beta, b_i, \sigma^2) \cdot p(b_i|\boldsymbol{\Gamma}) \prod_{i=1}^N p(b_i^\star|\boldsymbol{\Gamma})} \tag{4.10}$$

$$= \frac{\prod_{i=1}^N p(\boldsymbol{y}_i|\beta, b_i^\star, \sigma^2)}{\prod_{i=1}^N p(\boldsymbol{y}_i|\beta, b_i, \sigma^2)} \tag{4.11}$$

Using this specific candidate distribution we can derive a neat format for acceptance rate that only depends on the conditional probability of $\boldsymbol{y}$ given $\boldsymbol{b}$.

Now we go back to look at the complete log-likelihood $\log L_{\boldsymbol{W}}$:

$$\log L_{\boldsymbol{W}} = -\frac{N_{tot}}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i,j}(y_{ij} - \eta_i(t_{ij}))^2 - \frac{N}{2}\log(|\boldsymbol{\Gamma}|) - \frac{1}{2}\sum_{i=1}^{N} b_i \boldsymbol{\Gamma}^{-1} b_i \tag{4.12}$$

$$= -\frac{N_{tot}}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i,j}(y_{ij} - \hat{\eta}_i(t_{ij}))^2 - \frac{N}{2}\log(|\boldsymbol{\Gamma}|) - \frac{1}{2}\sum_{i=1}^{N} b_i \boldsymbol{\Gamma}^{-1} b_i \tag{4.13}$$

where $N_{tot} = \sum_{i=1}^{N} n_i$ represents the total number of observations. Given β and $\boldsymbol{b}$, $\eta_i(t_{ij})$ in (4.18) can be approximated as $\hat{\eta}_i(t_{ij})$ using numerical ODE solvers introduced in previous sections. Moreover, given certain $\boldsymbol{b}$, we can update of β, $\boldsymbol{\Gamma}$ and σ^2 using the sufficient statistics $\sum_i b_i$, $\sum_i b_i b_i'$ and $\sum_{i,j}\epsilon_{ij}^2$ respectively.

By combining the Metropolis-Hasting sampling, numerical ODE solver and EM algorithm, an SAEM algorithm is developed as follows:

1. Select starting value for $\beta^{(0)}$, $\sigma^{(0)}$ and $\boldsymbol{\Gamma}^{(0)}$. Set $m = 0$.

2. Generate L Markov Chains to draw L values, $\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \ldots, \boldsymbol{b}^{(L)}$ from its full conditional density function $p(\boldsymbol{b}^{\star}|\boldsymbol{y}, \beta^{(m)}, \sigma^{(m)}, \boldsymbol{\Gamma}^{(m)})$ using Metropolis-Hasting algorithm and numerical ODE solver mentioned previously.

3. Find $\beta^{\star}$ and $\sigma^{\star}$ that maximize the Monte Carlo estimate of $\mathrm{E}[\log p(\boldsymbol{y}|\boldsymbol{b}, \beta, \sigma)|\boldsymbol{y}]$, which is $\frac{1}{L}\sum_{l=1}^{L}[-\frac{N_{tot}}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i,j}(y_{ij} - \hat{\eta}_i^{(l)}(t_{ij}))^2]$, where $\hat{\eta}^{(l)}$ is the nu-

merical ODE approximation of η using $\boldsymbol{b}^{(l)}$. Set

$$\beta^{(m+1)} = \beta^{(m)} + \gamma_m(\beta^\star - \beta^{(m)}) \tag{4.14}$$

$$\sigma^{(m+1)} = \sigma^{(m)} + \gamma_m(\sigma^\star - \sigma^{(m)}) \tag{4.15}$$

Where (γ_m) is a smoothing parameter, i.e. a decreasing sequence of positive numbers which helps accelerating convergence.

4. Find $\mathbf{\Gamma}^\star$ that maximize $\frac{1}{L}\sum_{l=1}^{L}[-\frac{1}{2}\sum_{i=1}^{N} b_i^{(l)'}\mathbf{\Gamma}^{-1}b_i^{(l)}]$. Set

$$\mathbf{\Gamma}^{(m+1)} = \mathbf{\Gamma}^{(m)} + \gamma_m(\mathbf{\Gamma}^\star - \mathbf{\Gamma}^{(m)}) \tag{4.16}$$

   Set $m = m+1$.

5. Return $\beta^{(m+1)}$, $\sigma^{(m+1)}$, $\mathbf{\Gamma}^{(m+1)}$ as MLE if convergence is reached, otherwise set $m = m+1$ and go to step 2.

The sequence γ_m plays a role of including previous information into the new step, as well as help the algorithm reach convergence. A common choice for γ_m sequence is $\gamma_m = \frac{1}{m}$, or a piecewise function

$$\gamma_m = \begin{cases} 1 & m \le K \\ \frac{1}{m-K} & m > K \end{cases} \tag{4.17}$$

for a positive number K. In this way the algorithm would not utilize previous information until it is getting closer to MLE. The validity of the SAEM-MLE algorithm is discussed in the simulation study section.

4.4 REML Estimation

The maximum likelihood estimation for variance parameters is known to be biased downwards. REML accounts for the degree of freedom lost by estimating the fixed effects and makes a less biased estimation for the random effect variance. Therefore it is a preferable estimation method, especially for small sample size situation. The challenging part of implementing REML algorithm on the mixed model of dynamical systems is due to the complex and non-analytical form of the likelihood function. Following the idea of REML estimation for nonlinear mixed effect model by Meza et al. (2007), a REML estimation for population dynamical system model is developed. The combination of EM, Monte Caro simulation and numerical approximation for differential equation model is incorporated in the REML estimation algorithm.

We treat fixed effect parameter β as random, assuming it to be non-informative $\beta \sim N(0, \boldsymbol{V})$, where $|\boldsymbol{V}| = \infty$. Then the parameter to estimate is $\Phi^\star = (\boldsymbol{\Gamma}, \sigma)$ for REML and the complete log-likelihood can be written as:

$$\log L_{\boldsymbol{W}} = -\frac{N_{tot}}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i,j}(y_{ij} - \eta_i(t_{ij}))^2 - \frac{N}{2}\log(|\boldsymbol{\Gamma}|) - \frac{1}{2}\sum_{i=1}^{N} b_i\boldsymbol{\Gamma}^{-1}b_i + \text{const} \tag{4.18}$$

The SAEM algorithm for REML estimation can be put as:

1. Select starting value for $\sigma^{(0)}$ and $\boldsymbol{\Gamma}^{(0)}$. Set $m = 0$.

2. Generate L Markov Chains to draw L values, $\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \ldots, \boldsymbol{b}^{(L)}$ and $\beta^{(1)}, \beta^{(2)}, \ldots, \beta^{(L)}$

from $p(\boldsymbol{b}^\star, \beta^\star | \boldsymbol{y}, \sigma^{(m)}, \boldsymbol{\Gamma}^{(m)})$ using MCMC algorithm and numerical ODE solver mentioned previously.

3. Find $\sigma^\star$ that maximize the Monte Carlo estimate of $\mathrm{E}[\log p(\boldsymbol{y}|\boldsymbol{b}, \beta, \sigma)|\boldsymbol{y}]$, which can be presented as $\frac{1}{L}\sum_{l=1}^{L}[-\frac{N_{tot}}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i,j}(y_{ij} - \hat{\eta}_i^{(l)}(t_{ij}))^2]$, where $\hat{\eta}^{(l)}$ is the numerical ODE approximation of η using $\boldsymbol{b}^{(l)}$ and $\beta^{(l)}$. Set

$$\sigma^{(m+1)} = \sigma^{(m)} + \gamma_m(\sigma^\star - \sigma^{(m)}) \tag{4.19}$$

   Where (γ_m) is a smoothing parameter, i.e. a decreasing sequence of positive numbers which helps accelerating convergence.

4. Find $\boldsymbol{\Gamma}^\star$ that maximize $\frac{1}{L}\sum_{l=1}^{L}[-\frac{1}{2}\sum_{i=1}^{N} b_i^{(l)\prime}\boldsymbol{\Gamma}^{-1}b_i^{(l)}]$. Set

$$\boldsymbol{\Gamma}^{(m+1)} = \boldsymbol{\Gamma}^{(m)} + \gamma_m(\boldsymbol{\Gamma}^\star - \boldsymbol{\Gamma}^{(m)}) \tag{4.20}$$

5. Return $\sigma^{(m+1)}$ and $\boldsymbol{\Gamma}^{(m+1)}$ as REML estimation if convergence is reached, otherwise set $m = m + 1$ and go to step 2.

For step 2, to sample from the condition distribution of $p(\boldsymbol{b}^\star, \beta^\star | \boldsymbol{y}, \sigma^{(m)}, \boldsymbol{\Gamma}^{(m)})$ is difficult since we can not directly calculate the probability density function. Therefore MCMC scheme is utilized here. Compared to sampling from their joint conditional density, it is more convenient to use the Gibbs sampling scheme. For example, for the ith Markov chain, we can draw $\boldsymbol{b}^{(i)}$ from the conditional distribution $p(\boldsymbol{b}^\star | \boldsymbol{y}, \sigma^{(m)}, \boldsymbol{\Gamma}^{(m)}, \beta^{(i)\prime})$, where $\beta^{(i)\prime}$ is the ith sampled β from previous iteration. This can be done using MH algorithm described in previous subsection. Then $\beta^{(i)}$ can be drawn from $p(\beta^\star|\boldsymbol{y}, \sigma^{(m)}, \boldsymbol{\Gamma}^{(m)}, \boldsymbol{b}^{(i)})$ using MH algorithm with a Gaussian proposal distribution centered at previous point. Let β be the previous draw in MH and using candidate distribution we generate a new value $\beta^\star$. Since we have a non-informative prior for β, the acceptance probability $p(\beta, \beta^\star) = \min\{1, A(\beta, \beta^\star)\}$, where $A(\beta, \beta^\star)$ can be derived as

$$A(\beta, \beta^\star) = \frac{p(\beta^\star|\boldsymbol{y}, \boldsymbol{b}, \sigma, \boldsymbol{\Gamma})p(\beta|\beta^\star)}{p(\beta|\boldsymbol{y}, \boldsymbol{b}, \sigma, \boldsymbol{\Gamma})p(\beta^\star|\beta)} \tag{4.21}$$

$$= \frac{\prod_{i=1}^{N} p(\boldsymbol{y}_i|\beta^\star, b_i, \sigma^2) \cdot p(\beta^\star|\boldsymbol{V}) \cdot p(\beta|\beta^\star)}{\prod_{i=1}^{N} p(\boldsymbol{y}_i|\beta, b_i, \sigma^2) \cdot p(\beta|\boldsymbol{V}) \cdot p(\beta^\star|\beta)} \tag{4.22}$$

$$= \frac{\prod_{i=1}^{N} p(\boldsymbol{y}_i|\beta^\star, b_i, \sigma^2) \cdot p(\beta|\beta^\star)}{\prod_{i=1}^{N} p(\boldsymbol{y}_i|\beta, b_i, \sigma^2) \cdot p(\beta^\star|\beta)} \tag{4.23}$$

The computation of $p(\boldsymbol{y}_i|\beta, b_i, \sigma^2)$ is evaluated by the numerical ODE approximation algorithm selected.

Step 3 and 4 can be reduced to updating $\boldsymbol{\Gamma}$ and σ^2 without using optimization algorithms. In the algorithm, they can be updated as:

$$\sigma^{2^\star} = \frac{1}{L}\sum_{l=1}^{L}\left[\frac{1}{N_{tot}}\sum_{i,j}(y_{ij} - \hat{\eta}_i^{(l)}(t_{ij}))^2\right] \tag{4.24}$$

$$\boldsymbol{\Gamma}^\star = \frac{1}{L}\sum_{l=1}^{L}\left[\frac{1}{N}\sum_{i=1}^{N}(b_i^{(l)}b_i^{(l)'})\right] \tag{4.25}$$

In the following section, several simulation studies are conducted to validate and compare the SAEM-MLE and SAEM-REML algorithm for population dynamical system models without analytic solutions. Moreover, a simulation study is done with respect to parallelization of the SAEM algorithm, which significantly improves its computation efficiency.

4.5 Simulation Study

Simulation Study 1: Modeling Mercury Pollution in Fish

Simulation study is conducted in order to test the properties of those two algorithms. To model mercury pollution in fish, a compartment model can be used. Let $x(t)$ represent the concentration of mercury in the tissue of the fish at time t. The differential equation is

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \theta_0 - \theta_1 x(t) \tag{4.26}$$

The initial condition is assumed to be $x(0) = 0$. The parameters θ_0 and θ_1 are rate constants associated with uptake and discharge for each object. The observation model can be written as follows (for object i, at time t_{ij}):

$$y_{ij} = x_{ij} + \epsilon_{ij} \tag{4.27}$$

Where ϵ_{ij} is assumed to be i.i.d Gaussian distributed $\epsilon_{ij} \sim N(0, \sigma^2)$. x_{ij} is the solution of the corresponding differential equation:

$$\frac{\mathrm{d}x_i(t)}{\mathrm{d}t} = \theta_{0i} - \theta_{1i} x(t), \quad i = 1, \ldots, N \tag{4.28}$$

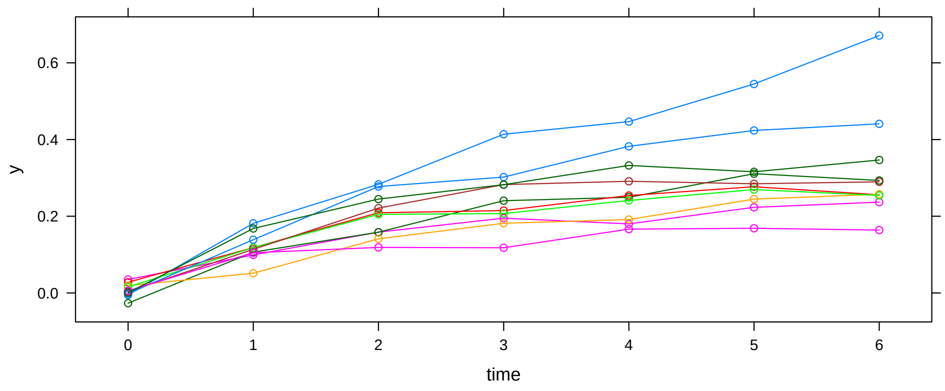


Figure 4.1: Example Simulated Data: 10 different observation units, each having 7 records.

at time t_{ij}, where $i = 1, \ldots, N$ and $j = 1, \ldots, n_i$. The two parameters of the model θ_{0i} and θ_{1i} are assumed to be normally distributed.

$$\theta_{0i} = \alpha + b_{0i}, \quad \text{where } b_{0i} \sim N(0, \sigma_0^2) \tag{4.29}$$

$$\theta_{1i} = \beta + b_{1i}, \quad \text{where } b_{1i} \sim N(0, \sigma_1^2) \tag{4.30}$$

where α and β are considered as fixed effects, i.e., with the same value for all fish. Meanwhile b_{0i} and b_{1i} are Gaussian random effects for each object. The set of parameters to estimate is $\{\alpha, \beta, \sigma, \sigma_0, \sigma_1\}$.

Data were simulated using the following parameter values: $N = 10$, $n_i = 7$, $\alpha = 0.15$, $\beta = 0.50$, $\sigma = 0.05$, $\sigma_0 = 0.05$, $\sigma_1 = 0.15$. Using the estimation method described previously, we simulated the dataset 1000 times. One of the example simulated sample is shown in Figure 4.1. The summary statistics for estimates are obtained and displayed in Table 4.1 and 4.2. The estimation accuracy is evaluated

Parameter	**True Value**	**Estimation**	**Std Dev**	**ARE**
α	0.15	0.144	0.0028	4.29
β	0.50	0.495	0.0081	1.60
σ	0.02	0.019	0.0016	7.32
σ_0	0.05	0.048	0.0101	16.55
σ_1	0.15	0.143	0.0290	16.84

Table 4.1: Summary statistics for parameter estimates using SAEM with Numerical ODE solvers.

	Method	σ	σ_0	σ_1
True Value		0.02	0.05	0.15
Mean	ML	0.019	0.048	0.143
	REML	0.019	0.049	0.142
SD	ML	0.0016	0.0101	0.0290
	REML	0.0018	0.0093	0.0265
ARE	ML	7.32	16.55	16.84
	REML	8.36	15.39	14.88

Table 4.2: Summary statistics for the variance parameter using SAEM-MLE and SAEM-REML estimation

by average relative error (ARE), which is defined as

$$\text{ARE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\hat{\theta}_i - \theta}{\theta}\right| \tag{4.31}$$

where $\hat{\theta}_i$ is the estimator of θ in the ith simulation run and $i = 1, \ldots, n$.

From the table, it can be seen that the SAEM-MLE could generate relative accurate estimations for $\boldsymbol{\beta}$ and σ, however the estimation for the random effects variance is not satisfying, with relatively large values of ARE. The SAEM-REML generates slightly better estimation, even though it is not significant enough. It can be assumed that when more random effect coefficients are included in the model, the difference could be larger.

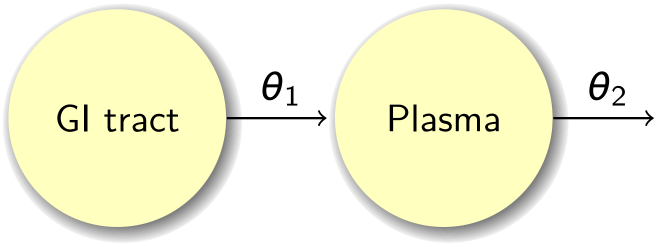


Figure 4.2: Compartmental diagram for a 2-compartment model in pharmacokinetics.

Simulation Study 2: Population PK Modeling Analysis

In this study a partially observed 2-compartment model in pharmacokinetics is presented. Based on the compartmental diagram in Figure 4.2, the ODE model has the following form:

$$\frac{\mathrm{d}x_1(t)}{\mathrm{d}t} = -\theta_1 x_1(t) \tag{4.32}$$

$$\frac{\mathrm{d}x_2(t)}{\mathrm{d}t} = \theta_1 x_1(t) - \theta_2 x_2(t) \tag{4.33}$$

By assumption we have the initial condition $(x_1(0), x_2(0)) = (1, 0)$. Moreover, usually only observations from the second compartment is available. The observation model is:

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{\epsilon} \tag{4.34}$$

where $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N)$ and $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N)$. A proportional error term $\boldsymbol{\epsilon} \sim N(0, \boldsymbol{\Sigma})$ is assumed, where $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{x}$. Suppose we have N subjects in the experiments, each having observations on time point $t_{i1}, t_{i2}, \ldots, t_{in_i}$, then $\boldsymbol{y}_i = (y_{i1}, y_{i2}, \ldots, y_{in_i})$ and $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in_i})$. More specifically, the observation model can be written as:

$$y_{ij} = x_{ij} + \epsilon_{ij} \tag{4.35}$$

where ϵ_{ij} is assumed to be i.i.d Gaussian distributed as $\epsilon_{ij} \sim N(0, \sigma^2 x_{ij}^2)$. x_{ij} represents the solution of $x_{i2}(t)$ of the corresponding differential equation:

$$\frac{\mathrm{d}x_{i1}(t)}{\mathrm{d}t} = -\theta_{1i} x_{i1}(t) \tag{4.36}$$

$$\frac{\mathrm{d}x_{i2}(t)}{\mathrm{d}t} = \theta_{1i} x_{i1}(t) - \theta_{2i} x_{i2}(t) \tag{4.37}$$

at time t_{ij}, where $i = 1, \ldots, N$ and $j = 1, \ldots, n_i$. The two individual parameters θ_{1i} and θ_{2i} are composed of two sub parameters.

$$\boldsymbol{\theta} = \boldsymbol{\beta} + \boldsymbol{b}, \quad \text{where } \boldsymbol{b} \sim N(0, \boldsymbol{\Gamma}) \tag{4.38}$$

where $\boldsymbol{\theta} = (\theta_{1i}, \theta_{2i})$ is the individual coefficient of population pharmacokinetics, $\boldsymbol{\beta} = (\alpha, \beta)$ are fixed effects, i.e. with the same value for all subjects, $\boldsymbol{b} = (b_{1i}, b_{2i})$ are individual Gaussian random effects, and $\boldsymbol{\Gamma}$ is the variance-covariance matrix. We can specify different covariance structures for $\boldsymbol{\Gamma}$. The set of parameters is $\{\boldsymbol{\beta}, \boldsymbol{\Gamma}, \sigma\}$.

We set $N = 30$, $t = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 24$, and we assume $(x_1(0), x_2(0)) = (1000, 0)$. The true parameter is $(0.50, 0.60, 0.10, 0.10, 0.05)$. The initial value for estimation is $(0.80, 0.80, 0.20, 0.20, 0.05)$. The summary statistics for both MLE and REML estimates are obtained. The simulation study shows that the algorithm is relatively robust to starting values of the parameters. One example of simulated dataset is plotted in Figure 4.3. Using the parameter estimation method described

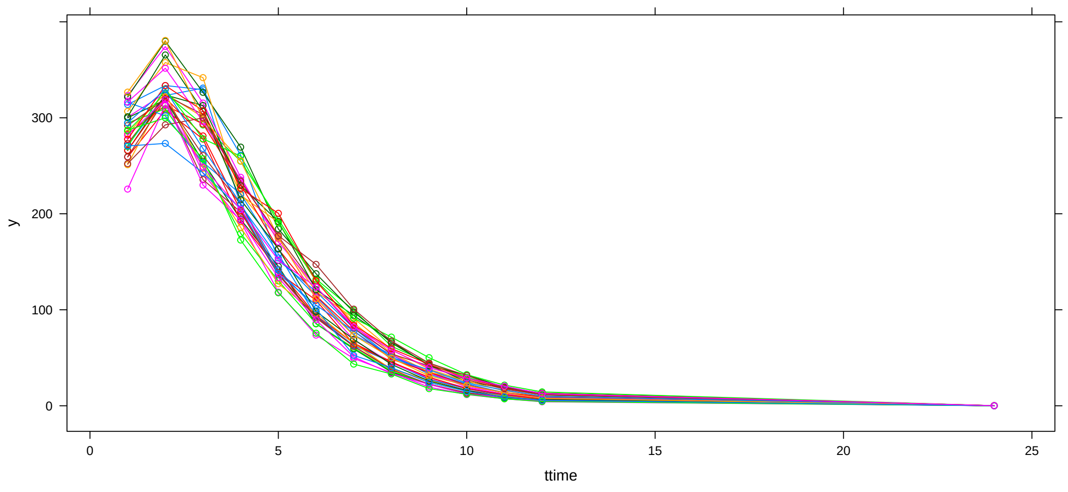


Figure 4.3: Example of simulated data: 30 different subjects, each having 13 observation records.

Parameter	True Value	Estimation	Std Dev	ARE
α	0.50	0.498	0.018	2.90
β	0.60	0.601	0.019	2.47
σ_1	0.10	0.087	0.012	15.18
σ_2	0.10	0.090	0.013	13.80
σ	0.05	0.0504	0.006	4.43

Table 4.3: Summary statistics for parameter estimations using SAEM with Numerical ODE solvers.

previously, we got the results shown in Table 4.3 and 4.4. From the two tables, it can be seen that the REML could correct the bias for the estimation of random effect variance in a certain level. For model diagnostic purpose, a model fit plot for each of the 30 individuals from one example run are generated in Figure 4.4. From the plot, it can be seen that the final estimated model could precisely capture the actual observations, and the difference between individuals could be discovered fairly well.

	Method	σ	σ_1	σ_2
True Value		0.05	0.10	0.10
Mean	ML	0.0504	0.087	0.090
	REML	0.0512	0.098	0.099
SD	ML	0.006	0.012	0.013
	REML	0.019	0.016	0.012
ARE	ML	4.43	15.18	13.80
	REML	7.36	13.11	9.96

Table 4.4: Summary statistics for the variance parameter using MLE and REML estimation

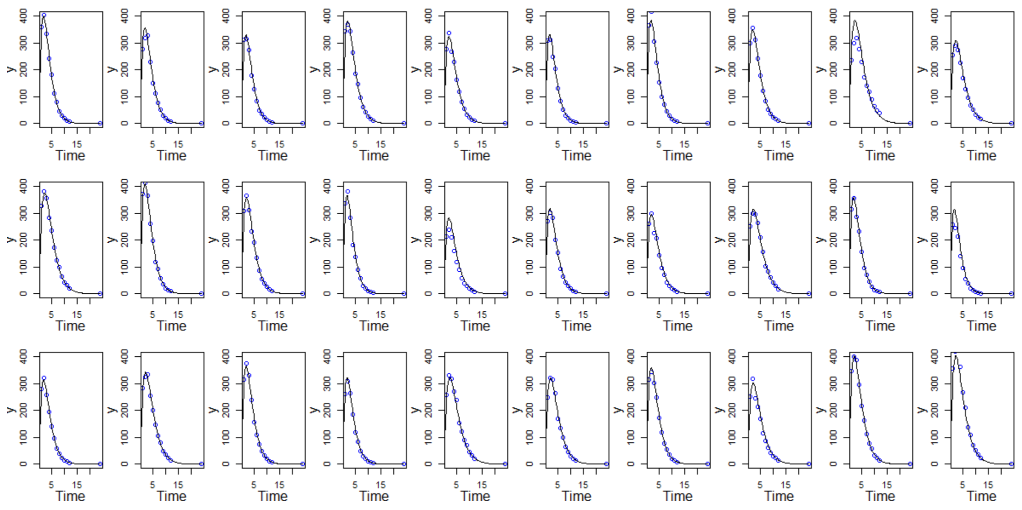


Figure 4.4: Observations versus predictions: estimated dynamical system for different individuals.

Simulation Study 3: Parallelized SAEM Algorithm

Since multiple Markov chains are generated each iteration of the SAEM algorithm for each objects in the experiment, it is quite computationally intensive. Parallel computing is one available option that could dramatically increase the efficiency of the algorithm, therefore it is studied and implemented.

The simulation step in the SAEM algorithm is usually very time consuming since likelihood has to be approximated many times for each subject on each Markov chain that being generated. At the same time, it is noticed that those computations can

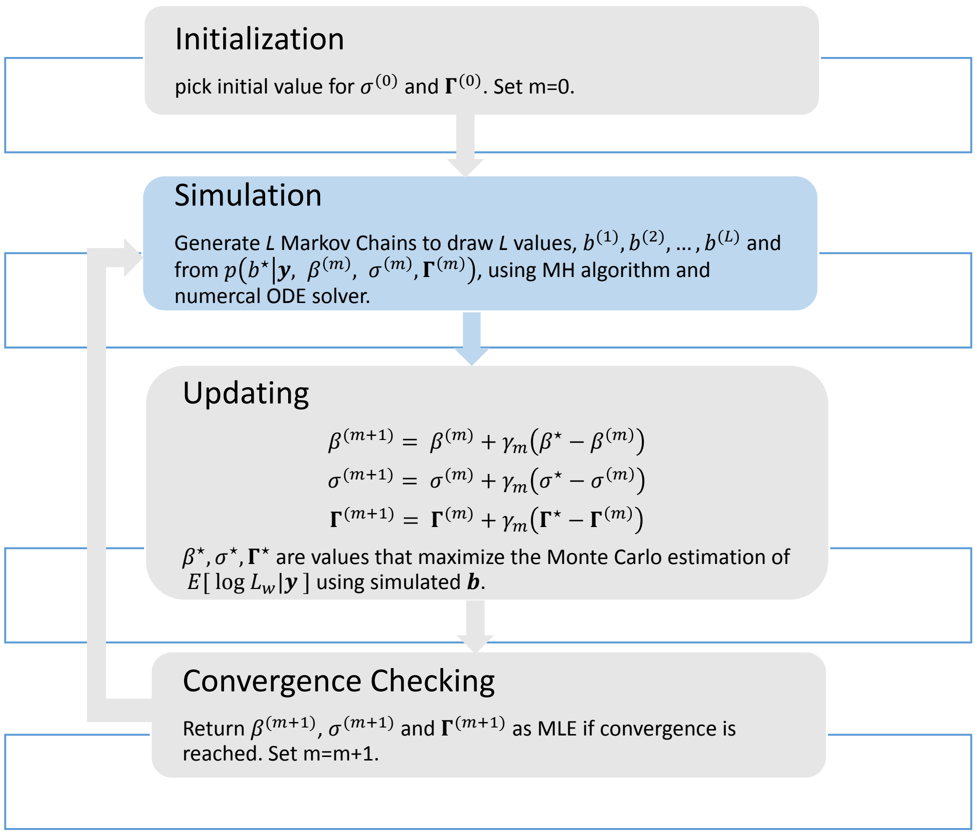

Initialization
pick initial value for $\sigma^{(0)}$ and $\mathbf{\Gamma}^{(0)}$. Set m=0.
Simulation
Generate L Markov Chains to draw L values, $b^{(1)}, b^{(2)}, \ldots, b^{(L)}$ and from $p(b^{\star}|\boldsymbol{y},\ \beta^{(m)},\ \sigma^{(m)}, \mathbf{\Gamma}^{(m)})$, using MH algorithm and numercal ODE solver.
Updating
$\beta^{(m+1)} = \beta^{(m)} + \gamma_m(\beta^{\star} - \beta^{(m)})$
$\sigma^{(m+1)} = \sigma^{(m)} + \gamma_m(\sigma^{\star} - \sigma^{(m)})$
$\mathbf{\Gamma}^{(m+1)} = \mathbf{\Gamma}^{(m)} + \gamma_m(\mathbf{\Gamma}^{\star} - \mathbf{\Gamma}^{(m)})$
$\beta^{\star}, \sigma^{\star}, \mathbf{\Gamma}^{\star}$ are values that maximize the Monte Carlo estimation of $E[\log L_w|\boldsymbol{y}]$ using simulated $\boldsymbol{b}$.
Convergence Checking
Return $\beta^{(m+1)}$, $\sigma^{(m+1)}$ and $\mathbf{\Gamma}^{(m+1)}$ as MLE if convergence is reached. Set m=m+1.

Figure 4.5: Basic structure of SAEM-MLE algorithm with numerical ODE solvers. The simulation step is the part that parallel computing is implemented

be done separately and independently. So this makes a great example to do parallel computing to improve the efficiency of the algorithm. Figure 4.5 and 4.6 shows the part of SAEM algorithm where parallel computation could be implemented.

Basically there are two ways of doing parallel computation in the algorithm. One is to parallelize on the Markov chains, the other is to parallelize on individual subjects. Our simulation study shows that both parallel method could decrease the running time for the algorithm while reaching the same level of accuracy on the parameter estimation.

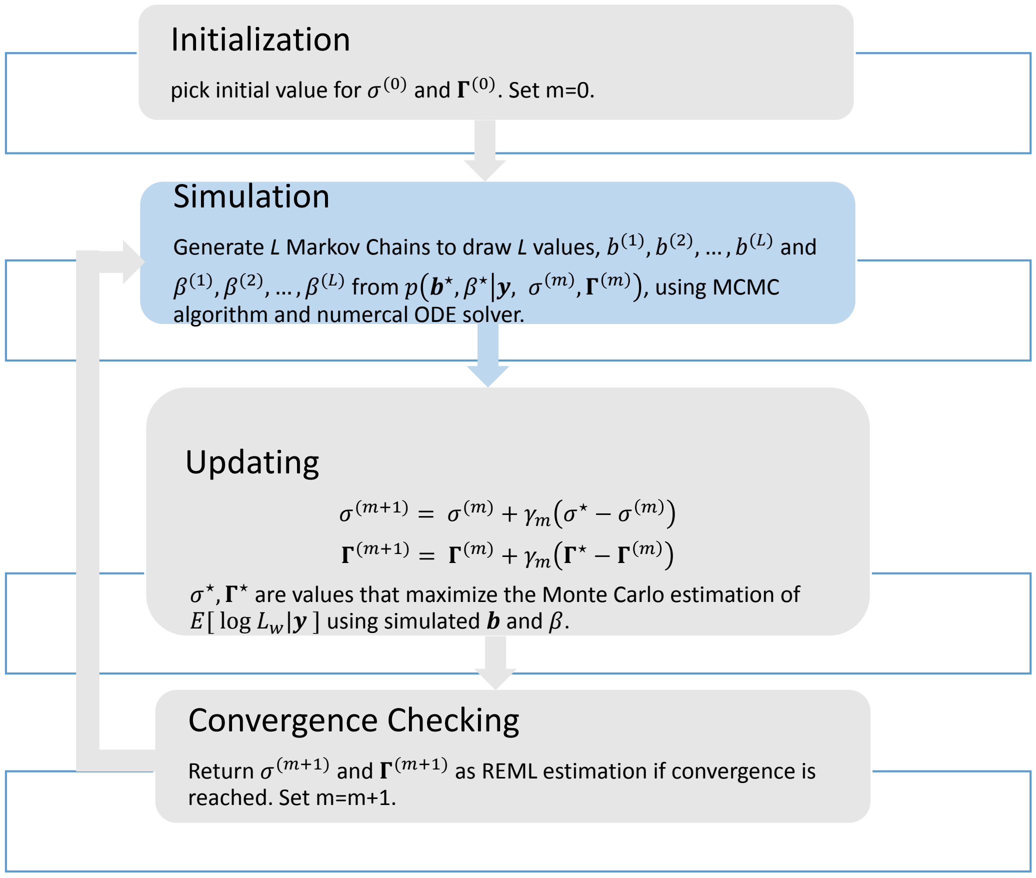

Initialization
pick initial value for $\sigma^{(0)}$ and $\mathbf{\Gamma}^{(0)}$. Set m=0.
Simulation
Generate L Markov Chains to draw L values, $b^{(1)}, b^{(2)}, \ldots, b^{(L)}$ and $\beta^{(1)}, \beta^{(2)}, \ldots, \beta^{(L)}$ from $p(\boldsymbol{b}^\star, \beta^\star | \boldsymbol{y}, \ \sigma^{(m)}, \mathbf{\Gamma}^{(m)})$, using MCMC algorithm and numercal ODE solver.
Updating
$\sigma^{(m+1)} = \sigma^{(m)} + \gamma_m(\sigma^\star - \sigma^{(m)})$
$\mathbf{\Gamma}^{(m+1)} = \mathbf{\Gamma}^{(m)} + \gamma_m(\mathbf{\Gamma}^\star - \mathbf{\Gamma}^{(m)})$
$\sigma^\star, \mathbf{\Gamma}^\star$ are values that maximize the Monte Carlo estimation of $E[\log L_w | \boldsymbol{y}]$ using simulated $\boldsymbol{b}$ and β.
Convergence Checking
Return $\sigma^{(m+1)}$ and $\mathbf{\Gamma}^{(m+1)}$ as REML estimation if convergence is reached. Set m=m+1.

Figure 4.6: Basic structure of SAEM-REML algorithm with numerical ODE solvers. The simulation step is the part that parallel computing is implemented

There are several packages that endow R with multi-threading capabilities (Eubank and Kupresanin, 2011). The CRAN website has a task view for high-performance computing that introduces the current state as well as future development of parallel computing in R. In the simulation study two R packages 'doParallel' (Analytics and Weston, 2014) and 'foreach' (Analytics and Weston, 2013) is used. The doParallel package is a parallel back-end for the foreach package. It provides a mechanism needed to execute foreach loops in parallel. All together they could provide a nice, efficient parallel programming platform for multiprocessor/multi-core computers run-

N_{MC}	1000	2000	3000	5000	10000
Regular	10.52	20.88	31.07	51.57	103.38
Parallel	2.99	6.02	8.77	14.57	28.92
SF	3.51	3.47	3.54	3.53	3.59

Table 4.5: Mean computation time and speedup factor for regular run and parallel run for SAEM algorithm using ODE solvers based on different number of MCMC samples. $N_{sub} = 20.L = 5$.

ning operating systems such as Linux and Mac OS.

To parallel on the Markov chain Monte Carlo samples, we set the number of simulation trials to be 100, and we compute the mean computation time for both paralleled and unparalleled version of code. In addition, we defined a speedup factor as follows to compare their performance.

$$\text{SF} = \frac{\text{Mean Computation Time for Single CPU Run}}{\text{Mean Computation Time for Parallel Run}} \tag{4.39}$$

The number of experiment subjects we use is 30. Based on this we increase the number of MCMC samples from 1000 to 10000, and check the mean computation time and speedup factor. The simulation study shows that parallel computation could significantly improve the computation speed for the algorithm. Moreover, a similar simulation study is conducted to parallelize on individual subjects, which also shows that the parallel computation is much more efficient than regular run, and it can be assumed that the performance of parallel computation will get better when more powerful machine is used.

N_{sub}	10	20	30	50	100
Regular	5.25	10.39	17.25	27.47	56.26
Parallel	1.68	3.05	4.39	6.97	13.83
SF	3.13	3.40	3.92	3.94	4.06

Table 4.6: Mean computation time and speedup factor for regular run and parallel run for SAEM algorithm using ODE solvers based on different number of subjects.$N_{MC} = 1000$, $L = 5$. The simulation is done on a Intel Core(TM) processor with 4 core CPU and 8 logical processors.

4.6 Approximate Confidence Intervals

After we computed the point estimator for the parameters $\mathbf{\Phi} = (\boldsymbol{\beta}, \sigma, \mathbf{\Gamma})$, the follow-up question is how confident are we on the accuracy of our estimation. The Fisher information matrix is a way to assess the variance of parameter estimations. Bauer and Guzy (2004) introduces a method to estimation Fisher information matrix using Monte Carlo simulations from the final iteration of EM algorithms.

The Fisher information matrix of log likelihood is defined as:

$$I = E_{\boldsymbol{y}}\left[-\frac{\partial^2 \log(p(\boldsymbol{y}|\mathbf{\Phi}))}{\partial \Phi^2}\right] \tag{4.40}$$

where we define

$$p(y_i|\mathbf{\Phi}) = \int_{-\infty}^{\infty} p_i(y_i, b_i|\mathbf{\Phi}) d\mathbf{\Phi} \tag{4.41}$$

and

$$p_i(y_i, b_i|\mathbf{\Phi}) = p_i(y_i|b_i, \beta, \sigma) p(b_i|\Gamma) \tag{4.42}$$

Then we have

$$
\begin{aligned}
&E_{\boldsymbol{y}}\left[-\frac{\partial^2 \log(p(\boldsymbol{y}|\boldsymbol{\Phi}))}{\partial \Phi^2}\right] && (4.43)\\
&= \int_y \left[-\frac{\partial^2 \log(p(\boldsymbol{y}|\boldsymbol{\Phi}))}{\partial \Phi^2}\right] p(\boldsymbol{y}|\boldsymbol{\Phi})d\boldsymbol{y} && (4.44)\\
&= \int_y \left[-\frac{1}{p(\boldsymbol{y}|\boldsymbol{\Phi})}\frac{\partial^2 p(\boldsymbol{y}|\boldsymbol{\Phi})}{\partial \Phi_i \partial \Phi_j} + \frac{\partial \log(p(\boldsymbol{y}|\boldsymbol{\Phi}))}{\partial \Phi_i}\frac{\partial \log(p(\boldsymbol{y}|\boldsymbol{\Phi}))}{\partial \Phi_j}\right] p(\boldsymbol{y}|\boldsymbol{\Phi})d\boldsymbol{y} && (4.45)\\
&= \int_y \left[\frac{\partial \log(p(\boldsymbol{y}|\boldsymbol{\Phi}))}{\partial \Phi_i}\frac{\partial \log(p(\boldsymbol{y}|\boldsymbol{\Phi}))}{\partial \Phi_j}\right] p(\boldsymbol{y}|\boldsymbol{\Phi})d\boldsymbol{y} && (4.46)\\
&= E_y\left[\frac{\partial \log(p(\boldsymbol{y}|\boldsymbol{\Phi}))}{\partial \Phi_i}\frac{\partial \log(p(\boldsymbol{y}|\boldsymbol{\Phi}))}{\partial \Phi_j}\right] && (4.47)
\end{aligned}
$$

Since

$$
\begin{aligned}
\int_y -\frac{1}{p(\boldsymbol{y}|\boldsymbol{\Phi})}\frac{\partial^2 p(\boldsymbol{y}|\boldsymbol{\Phi})}{\partial \Phi_i \partial \Phi_j} p(\boldsymbol{y}|\boldsymbol{\Phi})d\boldsymbol{y} &= \int_y -\frac{\partial^2 p(\boldsymbol{y}|\boldsymbol{\Phi})}{\partial \Phi_i \partial \Phi_j} d\boldsymbol{y} && (4.48)\\
&= -\frac{\partial^2 \int_y p(\boldsymbol{y}|\boldsymbol{\Phi})d\boldsymbol{y}}{\partial \Phi_i \partial \Phi_j} && (4.49)\\
&= -\frac{\partial^2 1}{\partial \Phi_i \partial \Phi_j} && (4.50)\\
&= 0 && (4.51)
\end{aligned}
$$

This is a general equation for approximating the Fisher information matrix. Suppose we have N independent subjects: $y_1, \ldots, y_N$. It is proved in Bauer and Guzy (2004) that under mild regularity conditions we have

$$
\begin{aligned}
&E_y\left[\frac{\partial \log(p(\boldsymbol{y}|\boldsymbol{\Phi}))}{\partial \Phi_i}\frac{\partial \log(p(\boldsymbol{y}|\boldsymbol{\Phi}))}{\partial \Phi_j}\right] && (4.52)\\
&= \sum_{i=1}^{N} E\left[\frac{-\partial \log(p(y_i, b_i|\Phi))}{\partial \Phi_i}|y_i, \Phi\right] E\left[\frac{-\partial \log(p(y_i, b_i|\Phi))}{\partial \Phi_j}|y_i, \Phi\right] && (4.53)
\end{aligned}
$$

The problem is becoming that how to evaluate $E_{b_k}\left[\frac{-\partial\log(p(y_k,b_k|\mathbf{\Phi}))}{\partial\mathbf{\Phi}_i}|y_k,\mathbf{\Phi}\right]$ for each $\mathbf{\Phi}_i$. we know $\mathbf{\Phi}=(\beta,\sigma,\Gamma)$. By methods of differentiation, we have

$$E_{b_k}\left[\frac{-\partial\log(p(y_k,b_k|\mathbf{\Phi}))}{\partial\beta}|y_k,\mathbf{\Phi}\right] = -\Gamma^{-1}\bar{b}_k \tag{4.54}$$

$$E_{b_k}\left[\frac{-\partial\log(p(y_k,b_k|\mathbf{\Phi}))}{\partial\Gamma}|y_k,\mathbf{\Phi}\right] = \Gamma^{-1}(\Gamma-\bar{\Gamma}_k)\Gamma^{-1}-\frac{1}{2}\text{diag}[\Gamma^{-1}(\Gamma-\bar{\Gamma}_k)\Gamma^{-1}] \tag{4.55}$$

$$E_{b_k}\left[\frac{-\partial\log(p(y_k,b_k|\mathbf{\Phi}))}{\partial\sigma^2}|y_k,\mathbf{\Phi}\right] = -\frac{1}{2}(\frac{1}{\hat{\sigma}^2}-\frac{(y_k-\hat{y}_k)^2}{\hat{\sigma}^4}) \tag{4.56}$$

where $\bar{b}_k,\bar{\Gamma}_k$ are the posterior mean for each subject and $\hat{y}_k$ are the predicted value using maximum likelihood estimator and selected numerical ODE solver. To save time sometimes we can compute the derivative with respect to σ^2 numerically using finite difference method. The format of above equation may change based on the assumption we made about the variance parameters. After the approximate Fisher information matrix is computed, the inverse is an estimate for the parameter's variance-covariance matrix.

4.7 Summary

In this chapter, both SAEM-MLE and SAEM-REML with numerical ODE solver is implemented in R. The simulation study shows that REML estimation procedure can correct the bias on the variance components of random effects in a certain level. The REML estimation is preferred especially in the situation when the sample size is relative small, or there is a large number of random effect coefficients in the model.

Since the population Pharmacokinetic models can be under very complicated structures with lots of parameters involved, together with the time-consuming process of running numerical ODE solvers, it is very important to maintain high computation efficiency for the estimation algorithms. Therefore, parallel computing is essential in the implementation of SAEM algorithms with MCMC and numerical ODE solver.

Different variance structure, like AR(1) or compound symmetry, could be implemented and validated under the basic algorithm scheme. This could be a topic for the future work.

Appendix

R code

This is the R code for the simulation study of numerical ODE integration based method.

```r
###simluation study and method comparison for chapter 2

library(deSolve)
library(MASS)

###step 1: set parameter of interest

theta_1=0.67
theta_2=0.069
theta_3=0.085
Time=seq(0,28,4)

###step 2: actual data simulation

dataSim=function(Theta1,Theta2,Theta3){
parameters=c(theta_1=Theta1,theta_2=Theta2,theta_3=Theta3)
state=c(X_1=0,X_2=1)
Time=seq(0,28,4)
compart=function(t,state,parameters){
with(as.list(c(state, parameters)),{
#rate of change
dX_1=-theta_1*X_1+theta_2*X_2
dX_2=theta_1*X_1+(-theta_2-theta_3)*X_2
#return the rate of change
list(c(dX_1,dX_2))
})
}
out=ode(y=state,times=Time,func=compart,parms=parameters,method="
    impAdams")
return(out[,c("X_1","X_2")])
}
error_1=rnorm(8,0,0.001)
error_2=rnorm(8,0,0.001)
DataStore=dataSim(theta_1,theta_2,theta_3)+cbind(error_1,error_2)

### step 3 parameter estimation

###method 1: differential evolution
###model(theta) estimation
NumApp=function(Theta, data){
parameters=c(theta_1=Theta[1],
theta_2=Theta[2],
theta_3=Theta[3])
state=c(X_1=0,
X_2=1)
compart=function(t,state,parameters){
with(as.list(c(state, parameters)),{
#rate of change
dX_1=-theta_1*X_1+theta_2*X_2
dX_2=theta_1*X_1+(-theta_2-theta_3)*X_2
#return the rate of change
list(c(dX_1,dX_2))
})
}
out=ode(y=state,times=Time,func=compart,parms=parameters,method="
    impAdams")
return(sum((out[,c("X_1","X_2")]-data)^2))
}

DiffEvolution=function(data, ngen){
X_1=data[,1]
X_2=data[,2]
n=3     # dimension of parameter space
npop=20   # number in population
F=0.65 # the weighting factor [0.5, 1.0]
M=0.2  # mutation factor [0.0,0.3]
Theta=matrix(nrow=n,ncol=npop)  # hold values for current generation
D=matrix(nrow=n,ncol=npop)  # hold part of adjustment
cost=vector("numeric",npop)
# create initial population
ib=1       # index of best value so far
best=1e100  # best value so far
for(j in 1:npop)
{
Theta[,j]=runif(n,0,1)
cost[j]=NumApp(Theta[,j],data)
if(cost[j]<best)
{
ib=j
best=cost[j]
}
}

# begin the evolution
for(i in 1:ngen)
{
# create difference vectors
for(j in 1:npop)
{
l=sample(1:npop,2)
D[,j]=Theta[,l[1]]-Theta[,l[2]]
}
# apply mutations
for(j in 1:npop)
for(k in 1:n)
if(runif(1)<M)
```

```
D[k,j]=-D[k,j]
# form and check offspring
for(j in 1:npop)
{
trial=Theta[,j]+F*D[,j]+F*(Theta[,ib]-Theta[,j])
if(NumApp(trial,data)<cost[j])
{
Theta[,j]=trial
cost[j]=NumApp(trial,data)
}
}
# check for best individual
for(j in 1:npop)
{
if(cost[j]<best)
{
best=cost[j]
ib=j
}
}
#  print(best)
}
return(Theta[,ib])
}
DiffEvolution(DataStore,100)

###Gauss-Newton Method

GaussNewton=function(data, Theta_Start){
Theta=Theta_Start
alpha=0.5
delta_max=1
iter=1
iter_max=10000
Time=seq(0, 28, 4)
X_1=data[,1]
X_2=data[,2]
while (delta_max>10^(-5)&&iter<iter_max){
parameters=c(theta_1=Theta[1],
theta_2=Theta[2],
theta_3=Theta[3])
state=c(X_1=0,
X_2=1,
U_1=0,
U_2=0,
U_3=0,
U_4=0,
U_5=0,
U_6=0)
compart=function(t,state,parameters){
with(as.list(c(state, parameters)),{
#rate of change
dX_1=-theta_1*X_1+theta_2*X_2
dX_2=theta_1*X_1+(-theta_2-theta_3)*X_2
dU_1=-X_1-theta_1*U_1+theta_2*U_2
dU_2=X_1+theta_1*U_1+(-theta_2-theta_3)*U_2
dU_3=X_2-theta_1*U_3+theta_2*U_4
dU_4=-X_2+theta_1*U_3+(-theta_2-theta_3)*U_4
dU_5=-theta_1*U_5+theta_2*U_6
dU_6=-X_2+theta_1*U_5+(-theta_2-theta_3)*U_6
#return the rate of change
list(c(dX_1,dX_2,dU_1,dU_2,dU_3,dU_4,dU_5,dU_6))
})
}
out=ode(y=state,times=Time,func=compart,parms=parameters)
J=-matrix(c(out[,"U_1"],out[,"U_2"],out[,"U_3"],out[,"U_4"],out[,"U_5"],
    out[,"U_6"]),16,3)
delta_X=c(X_1-out[,"X_1"],X_2-out[,"X_2"])
Theta=Theta-alpha*ginv(t(J)%*%J)%*%t(J)%*%delta_X
delta_max=max(abs(ginv(t(J)%*%J)%*%t(J)%*%delta_X))
iter=iter+1
}
print(iter)
return(Theta)
}

Theta_init=runif(3,0,1)
GaussNewton(DataStore,Theta_init)

### Method 3: Hybrid Method
DiffEvolution=function(data,ngen){
X_1=data[,1]
X_2=data[,2]
n=3     # dimension of parameter space
npop=20   # number in population
F=0.65 # the weighting factor [0.5, 1.0]
M=0.2  # mutation factor [0.0,0.3]
Theta=matrix(nrow=n,ncol=npop)  # hold values for current generation
D=matrix(nrow=n,ncol=npop)  # hold part of adjustment
cost=vector("numeric",npop)
# create initial population
ib=1       # index of best value so far
best=1e100  # best value so far
for(j in 1:npop)
{
Theta[,j]=runif(n,0, 1)
cost[j]=NumApp(Theta[,j],data)
if(cost[j]<best)
{
ib=j
best=cost[j]
}
}
# begin the evolution
for(i in 1:ngen)
{
# create difference vectors
for(j in 1:npop)
```

```
{
l=sample(1:npop,2)
D[,j]=Theta[,l[1]]-Theta[,l[2]]
}
# apply mutations
for(j in 1:npop)
for(k in 1:n)
if(runif(1)<M)
D[k,j]=-D[k,j]
# form and check offspring
for(j in 1:npop)
{
trial=Theta[,j]+F*D[,j]+F*(Theta[,ib]-Theta[,j])
if(NumApp(trial,data)<cost[j])
{
Theta[,j]=trial
cost[j]=NumApp(trial,data)
}
}
# check for best individual
for(j in 1:npop)
{
if(cost[j]<best)
{
best=cost[j]
ib=j
}
}
#  print(best)
}
return(Theta[,ib])
}

theta_start=DiffEvolution(DataStore,5)
GaussNewton(DataStore,theta_start)
```

Simluation study of numerical ODE integration based method

The following is the R code for the simulation Study on Chapter 3.

```

###Method based on Fourier Basis Smoothing, on FHN model

library(deSolve)
library(MASS)

nsample=1000
parfinal=matrix(NA,nsample,3)
for(i_sample in 1:nsample){
Time=seq(1,20,1)
Theta_real=c(0.2,0.2,3)
parameters=c(alpha=Theta_real[1],beta=Theta_real[2],gamma=Theta_real[3])
state=c(X_1=-1,X_2=1)
compart=function(t,state,parameters){
with(as.list(c(state,parameters)),{
#rate of change
dX_1=gamma*(X_1-X_1^3/3+X_2)
dX_2=(-1/gamma)*(X_1-alpha+beta*X_2)
#return the rate of change
list(c(dX_1,dX_2))
})
}

Time.1=seq(0,20,0.5)
out=ode(y=state,times=Time.1,func=compart,parms=parameters,method="
    impAdams")
n=length(Time.1)
V=out[,"X_1"]+rnorm(n,0,0.1)
R=out[,"X_2"]+rnorm(n,0,0.1)
suppressPackageStartupMessages(library(fda))
suppressPackageStartupMessages(library(fda.usc))
suppressPackageStartupMessages(library(reshape2))
### create fourier basis
basis9=create.fourier.basis(rangeval=range(Time.1),period=9,nbasis=9)
fourier9.fd=smooth.basis(argvals=Time.1,y=R,fdParobj=basis9)$fd
fou_x2_0=eval.fd(Time.1,fourier9.fd)
fou_x2_1=eval.fd(Time.1,fourier9.fd,Lfdobj=1)
basis17=create.fourier.basis(rangeval=range(Time.1),period=9,nbasis=17)
fourier17.fd=smooth.basis(argvals=Time.1,y=V,fdParobj=basis17)$fd
fou_x1_0=eval.fd(Time.1,fourier17.fd)
fou_x1_1=eval.fd(Time.1,fourier17.fd,Lfdobj=1)
x1hat=fou_x1_0
x1de=fou_x1_1
x2hat=fou_x2_0
x2de=fou_x2_1
error=function(vec,err='l2'){
if (err=='l2'){
return( sum((x1de-vec[3]*(x1hat-x1hat^3/3+x2hat))^2)+sum((x2de+(x1hat-
    vec[1]+vec[2]*x2hat)/vec[3])^2));
}else{
return( sum(abs(x1de-vec[3]*(x1hat-x1hat^3/3+x2hat)))+sum(abs(x2de+(
    x1hat-vec[1]+vec[2]*x2hat)/vec[3])));
}
}
re=list()
for(i in 1:10){
yyy=runif(3)
optim(yyy,error,method="Nelder-Mead",control=list(trace=F,maxit=2000,
    abstol=1e-10,reltol=1e-10))->re[[i]]
}
final_par=re[[which.min(unlist(lapply(re,function(x) x$value)))]]$par
parfinal[i_sample,]=final_par
}
colMeans(parfinal)
```

Simulation study on methods based on Fourier basis smoothing

This is the R code for the simulation study on population dynamical system models.

```

require(stats)
require(graphics)
library(MASS)
library(deSolve)
#### data simulation
time=c(0:12,24)
nobs=length(time)-1
alpha_sim=0.5
beta_sim=0.6
sigma1_sim=0.1
sigma2_sim=0.1
sigma_sim=0.05
####  simluate data x.
x_sim=function(alpha,beta,time){
parameters=c(theta_1=alpha,theta_2=beta)
state=c(X1=1000,X2=0)
Time=time
compart=function(t,state,parameters){
with(as.list(c(state,parameters)),{
#rate of change
dX1=-theta_1*X1
dX2=theta_1*X1-theta_2*X2
list(c(dX1,dX2))
})
}
out=ode(y=state,times=as.vector(unlist(Time)),func=compart,parms=
    parameters,method="impAdams")
sim_x=out[-1,c("X2")]
return(sim_x)
}
theta1=alpha_sim+rnorm(1,0,sigma1_sim)
theta2=beta_sim+rnorm(1,0,sigma2_sim)
x=x_sim(theta1,theta2,time)
y=x*(1+rnorm(nobs,0,sigma_sim))
obj=1
ttime=time[-1]
patient=data.frame(obj,ttime,y)
for(i in 2:30){
theta1=alpha_sim+rnorm(1,0,sigma1_sim)
theta2=beta_sim+rnorm(1,0,sigma2_sim)
x=x_sim(theta1,theta2,time)
y=x*(1+rnorm(nobs,0,sigma_sim))
obj=i
ttime=time[-1]
temp=data.frame(obj,ttime,y)
patient=rbind(patient,temp)
}
library(lattice)
xyplot(y~ttime,type=c('l','p'),groups=obj,data=patient,auto.key=F)

ll_sim=function(y_pred,y,sigma){
```

```r
varc=y_pred*y_pred*sigma*sigma;
d_pred=(y_pred-y)^2
ll= -0.5*sum(d_pred/varc+log(varc)+log(2*pi))
}

####      Specify initial parameter value

n_iter=5
n_sub=30
alpha=0.6
beta=0.8
sigma1=0.20
sigma2=0.20
sigma=0.05
sig_updt=1
sigma_h=0.00000001
n_sim=1000
dpre=rep(0,n_sim)
dlds=rep(0, n_sim)
h_ds=rep(0, n_sim)
ll_i_sim=rep(0, n_sim)
par_par=matrix(NA,n_sim,2)
ll_par=rep(NA,n_sim)
theta_i=matrix(0,n_sub,2)
b_i_sim=matrix(0,n_sub,4)
sigma_updt_i_sim=rep(0,n_sub)
sigma_h_updt_i_sim=rep(0,n_sub)
sigma_val_i_sim=rep(0,n_sub)
gam=rep(1,100)
for(i in 1:100){
gam[i]=1/i
}

library(doParallel)
library(foreach)
####  EM Algorithm
for(i_iter in 1:n_iter){
cl=makeCluster(10)
registerDoParallel(cl)
result=foreach(i_sub=1:n_sub,.combine=rbind,.packages=c('MASS','deSolve'
    ))%dopar%
{
patientsub=subset(patient,obj==i_sub)
t=patientsub[,2]
y=patientsub[,3]
## sampling random effects
bsim=mvrnorm(1,c(0,0), diag(c(sigma1^2, sigma2^2)))
val=c(alpha,beta)+bsim
## MCMC step
for (i in 1: n_sim){
par_old=val
y_old=y_pred=x_sim(par_old[1], par_old[2], c(0,t))
ll_val=ll_old=ll_sim(y_old, y, sigma)
varc=y_pred*y_pred*sigma*sigma
dpred_val=d_pred=(y_pred-y)^2
sig_val=dpred_val/(y_old^2)
varc_g1=y_old*y_old*(sigma+0.5*sigma_h)*(sigma+0.5*sigma_h)
varc_g2=y_old*y_old*(sigma-0.5*sigma_h)*(sigma-0.5*sigma_h)
varc_g=(varc_g1-varc_g2)/sigma_h
dlds_t_1=((varc-d_pred)/(2*(varc^2)))*varc_g
dlds_val=dlds_t_1
b_new=mvrnorm(1,c(0,0), diag(c(sigma1^2, sigma2^2)))
par_new=c(alpha,beta)+b_new
y_new=y_pred=x_sim(par_new[1],par_new[2],c(0,t))
ll_new=ll_sim(y_new,y,sigma)
varc=y_pred*y_pred*sigma*sigma
d_pred=(y_pred-y)^2
varc_g1=y_old*y_old*(sigma+0.5*sigma_h)*(sigma+0.5*sigma_h)
varc_g2=y_old*y_old*(sigma-0.5*sigma_h)*(sigma-0.5*sigma_h)
varc_g=(varc_g1-varc_g2)/sigma_h
dlds_t_2=((varc-d_pred)/(2*(varc^2)))*varc_g
ruse=exp(ll_new-ll_old)
u = runif(1)
if(ruse>log(u)){
val=par_new
dpred_val=d_pred
sig_val=dpred_val/(y_new^2)
ll_val=ll_new
dlds_val=dlds_t_2
}

par_par[i,]=val
ll_i_sim[i]=ll_val
dpre[i]=sig_val
dlds[i]=sum(dlds_val)
h_ds[i]=sum(dlds_val*dlds_val)
}
par=par_par
ll_i_sim_f=exp(ll_i_sim-max(ll_i_sim))
ll_i_sim_rat=ll_i_sim_f/sum(ll_i_sim_f)
theta_i_t=t(par)%*%matrix(ll_i_sim_rat,n_sim,1)
theta_i_avg=matrix(0,n_sim,2)
theta_i_avg[,1]=theta_i_t[1]
theta_i_avg[,2]=theta_i_t[2]
b_i_sim_t=t(par*ll_i_sim_rat-theta_i_avg*ll_i_sim_rat)%*% (par-theta_i_
    avg)
theta_i[i_sub,]=theta_i_t
b_i_sim[i_sub,]=matrix(b_i_sim_t,4,1)
sigma_val_i_sim[i_sub]=t(dpre)%*%matrix(ll_i_sim_rat,n_sim,1)
###items used to update sigma
sigma_updt_i_sim[i_sub]=sum(ll_i_sim_rat*dlds)
sigma_h_updt_i_sim[i_sub]=sum(ll_i_sim_rat*h_ds)
c(theta_i[i_sub,],b_i_sim[i_sub,],sigma_updt_i_sim[i_sub],sigma_h_updt_i
    _sim[i_sub], sigma_val_i_sim[i_sub])
}
stopCluster(cl)
theta_i=result[,1:2]
b_i_sim=result[,3:6]
```

```
sigma_updt_i_sim=result[,7]
sigma_h_updt_i_sim=result[,8]
##updated population mean and covariance
omega_tmp=matrix(0,n_sub,4)
MU_new=apply(theta_i,2,mean)
sigma_new=sqrt(mean(result[,9]))
for(j in 1:n_sub){
theta_tmp=theta_i[j,]-MU_new
theta_tmp2=theta_tmp%*%t(theta_tmp)
omega_tmp[j,]=matrix(theta_tmp2,4,1)
}
omega_new=colMeans(omega_tmp,na.rm=T)+colMeans(b_i_sim,na.rm=T)
sigma1_new=sqrt(omega_new[1])
sigma2_new=sqrt(omega_new[4])
sigma_update=(1/sum(sigma_h_updt_i_sim))*sum(sigma_updt_i_sim)
alpha=alpha+gam[i_iter]*(MU_new[1]-alpha)
beta=beta+gam[i_iter]*(MU_new[2]-beta)
sigma1=sigma1+gam[i_iter]*(sigma1_new-sigma1)
sigma2=sigma2+gam[i_iter]*(sigma2_new-sigma2)
if (sig_updt==1){
sigma=sigma+gam[i_iter]*(sigma_new-sigma)}
else{
sigma=sigma-sigma_update
}
}
parfinal=c(alpha,beta,sigma1,sigma2,sigma)
### Example code for SAEM-REML algorithm for 2-compartmental model
### data simulation
time=c(0:12,24)
nobs=length(time)-1
alpha_sim=0.5
beta_sim=0.6
sigma1_sim=0.1
sigma2_sim=0.1
sigma_sim=0.05
#### simluate data x.
x_sim=function(alpha,beta,time){
parameters=c(theta_1=alpha,theta_2=beta)
state=c(X1=1000,X2=0)
Time=time
compart=function(t,state,parameters){
with(as.list(c(state,parameters)),{
#rate of change
dX1=-theta_1*X1
dX2=theta_1*X1-theta_2*X2
list(c(dX1, dX2))
})
}
out=ode(y=state,times=as.vector(unlist(Time)),func=compart,parms=
    parameters,method="impAdams")
sim_x=out[-1,c("X2")]
return(sim_x)
}
theta1=alpha_sim+rnorm(1,0,sigma1_sim)
```

```
theta2=beta_sim+rnorm(1,0,sigma2_sim)
x=x_sim(theta1,theta2,time)
y=x*(1+rnorm(nobs,0,sigma_sim))
obj=1
ttime=time[-1]
patient=data.frame(obj,ttime,y)
for(i in 2:30){
theta1=alpha_sim+rnorm(1,0,sigma1_sim)
theta2=beta_sim+rnorm(1,0,sigma2_sim)
x=x_sim(theta1,theta2,time)
y=x*(1+rnorm(nobs,0,sigma_sim))
obj=i
ttime=time[-1]
temp=data.frame(obj,ttime,y)
patient=rbind(patient,temp)
}
library(lattice)
xyplot(y~ttime,type=c('l','p'),groups=obj,data=patient,auto.key=F)
ll_sim=function(y_pred,y,sigma){
varc=y_pred*y_pred*sigma*sigma;
d_pred=(y_pred-y)^2
ll=-0.5*sum(d_pred/varc+log(varc)+log(2*pi))
}
### Specify initial parameter value
n_iter=5
n_sub=30
alpha=0.8
beta=0.8
sigma1=0.20
sigma2=0.20
sigma=0.05
sigma_h=0.00000001
sig_updt=1
n_bi=100
n1_sim=100
n2_sim=100
n_sim=n1_sim*n2_sim
#n_sim=1000
dlds=rep(0,n_sim)
h_ds=rep(0,n_sim)
ll_i_sim=rep(0,n_sim)
dpred_sim=rep(0,n_sim)
par_par=matrix(NA,n_sim,2)
ll_par=rep(NA,n_sim)
theta_i=matrix(0,n_sub,2)
b_i_sim=matrix(0,n_sub,4)
sigma_updt_i_sim=rep(0,n_sub)
sigma_h_updt_i_sim=rep(0,n_sub)
sigma_val_i_sim=rep(0,n_sub)
gam=rep(1,100)
for(i in 1:100){
gam[i]=1/i
}
library(doParallel)
library(foreach)
###   EM Algorithm
for(i_iter in 1:n_iter){
cl=makeCluster(10)
registerDoParallel(cl)
result=foreach(i_sub=1:n_sub,.combine=rbind,.packages=c('MASS','deSolve'
    ))%dopar%
{
patientsub=subset(patient,obj==i_sub)
t=patientsub[,2]
y=patientsub[,3]
bi=mvrnorm(1,c(0,0),diag(c(sigma1^2, sigma2^2)))
beta_i=mvrnorm(1,c(alpha, beta),diag(c(sigma1^2, sigma2^2)))
val=bi+beta_i
### MCMC step
for(i in 1: n1_sim){
par_old=val
y_old=y_pred=x_sim(par_old[1], par_old[2], c(0,t))
ll_val=ll_old=ll_sim(y_old, y, sigma)
varc=y_pred*y_pred*sigma*sigma
dpred_val=d_pred=(y_pred-y)^2
sig_val=dpred_val/(y_old^2)
varc_g1=y_old*y_old*(sigma+0.5*sigma_h)*(sigma+0.5*sigma_h)
varc_g2=y_old*y_old*(sigma-0.5*sigma_h)*(sigma-0.5*sigma_h)
varc_g=(varc_g1-varc_g2)/sigma_h
dlds_t_1=((varc-d_pred)/(2*(varc^2)))*varc_g
dlds_val=dlds_t_1
for (k in 1:n_bi){
bi_new=mvrnorm(1,c(0,0),diag(c(sigma1^2, sigma2^2)))
par_new=bi_new+beta_i
y_new=y_pred=x_sim(par_new[1], par_new[2], c(0,t))
ll_new=ll_sim(y_new, y, sigma)
ruse=exp(ll_new-ll_old)
u=runif(1)
if(ruse>log(u)){
bi=bi_new
ll_val=ll_new
}
}
for (j in 1:n2_sim){
beta_j=mvrnorm(1,c(alpha,beta),diag(c(sigma1^2, sigma2^2)))
par_new=bi+beta_j
y_new=y_pred=x_sim(par_new[1], par_new[2], c(0,t))
ll_new=ll_sim(y_new, y, sigma)
varc=y_pred*y_pred*sigma*sigma
d_pred_new=(y_pred-y)^2
varc_g1=y_old*y_old*(sigma+0.5*sigma_h)*(sigma+0.5*sigma_h);
varc_g2=y_old*y_old*(sigma-0.5*sigma_h)*(sigma-0.5*sigma_h);
varc_g=(varc_g1-varc_g2)/(1*sigma_h);
dlds_t_2=((varc-d_pred)/(2*(varc^2)))*varc_g;
ruse=exp(ll_new-ll_old)
u=runif(1)
if(ruse>log(u)){
val=par_new
ll_val=ll_new
dpred_val=d_pred_new
sig_val=dpred_val/(y_new^2)
}
par_par[(i-1)*n1_sim+j,]=val
ll_i_sim[(i-1)*n1_sim+j]=ll_val
dpred_sim[(i-1)*n1_sim+j]=sig_val
dlds[(i-1)*n1_sim+j]=sum(dlds_val)
h_ds[(i-1)*n1_sim+j]=sum(dlds_val*dlds_val)
}
}
par=par_par
ll_i_sim_f=exp(ll_i_sim-max(ll_i_sim))
ll_i_sim_rat=ll_i_sim_f/sum(ll_i_sim_f)
theta_i_t=t(par)%*%matrix(ll_i_sim_rat,n_sim,1)
theta_i_avg=matrix(0,n_sim,2)
theta_i_avg[,1]=theta_i_t[1]
theta_i_avg[,2]=theta_i_t[2]
b_i_sim_t=t(par*ll_i_sim_rat-theta_i_avg*ll_i_sim_rat)%*% (par-theta_i_
    avg)
theta_i[i_sub,]=theta_i_t
b_i_sim[i_sub,]=matrix(b_i_sim_t,4,1)
###update sigma
sigma_val_i_sim[i_sub]=t(dpred_sim)%*%matrix(ll_i_sim_rat,n_sim,1)
sigma_updt_i_sim[i_sub]=sum(ll_i_sim_rat*dlds)
sigma_h_updt_i_sim[i_sub]=sum(ll_i_sim_rat*h_ds)
c(theta_i[i_sub,], b_i_sim[i_sub,], sigma_updt_i_sim[i_sub], sigma_h_
    updt_i_sim[i_sub],sigma_val_i_sim[i_sub])
}
stopCluster(cl)
theta_i=result[,1:2]
b_i_sim=result[,3:6]
sigma_updt_i_sim=result[,7]
sigma_h_updt_i_sim=result[,8]
##updated population mean and covariance
MU_new=apply(theta_i,2,mean)
sigma_new=sqrt(mean(result[,9]))
sigma1_new=sqrt(var(theta_i[,1]-MU_new[1]))
sigma2_new=sqrt(var(theta_i[,2]-MU_new[2]))
alpha=alpha+gam[i_iter]*(MU_new[1]-alpha)
beta=beta+gam[i_iter]*(MU_new[2]-beta)
sigma1=sigma1+gam[i_iter]*(sigma1_new-sigma1)
sigma2=sigma2+gam[i_iter]*(sigma2_new-sigma2)
sigma_update=(1/sum(sigma_h_updt_i_sim))*sum(sigma_updt_i_sim)
if (sig_updt==1){
sigma=sigma+gam[i_iter]*(sigma_new-sigma)}
else{
sigma=sigma-sigma_update
}
}
parfinal=c(sigma1,sigma2,sigma)
```

This is the code for the real data application in Chapter 3.

```

library(CollocInfer)
library(deSolve)
library(lokern)
library(MASS)
set.seed(123)

###Set kernel K(u)=(1-u^2)_{+}
kernel.e=function(x){
3/4*(1-x^2)*(1>x^2)+1e-10
}
### Input data
### Here we have the data is in the two csv files
workdata=read.csv("./data_cd8.csv",
header=TRUE,sep=",",quote="\"",dec=".",
fill=TRUE,comment.char="")
workdataD=read.csv("./data_dc.csv",
header=TRUE,sep=",",quote="\"",dec=".",
fill=TRUE,comment.char="")

### Clean data: drop the "NA" case and fit ODE from day 5 to day 14
workdata=data.frame(workdata)
workdata=workdata[!is.na(workdata$data_lun_2008),]
fitteddata=workdata[workdata$time>=5&workdata$time<=14,]

### Initial fit using Fourier basis smoothing
suppressPackageStartupMessages(library(fda))
suppressPackageStartupMessages(library(fda.usc))
suppressPackageStartupMessages(library(reshape2))
### Create and estimate the Fourier basis system
basis3=create.fourier.basis(rangeval = range(fitteddata$time),period=9,
    nbasis=3)
fourier3.fd=smooth.basis(argvals=fitteddata$time,y=log(fitteddata$data_
    mln_2008),fdParobj = basis3)$fd
fou_x1_0=eval.fd(fitteddata$time,fourier3.fd)
fou_x1_1=eval.fd(fitteddata$time,fourier3.fd, Lfdobj=1)
basis3=create.fourier.basis(rangeval=range(fitteddata$time),period=12,
    nbasis=3)
fourier3.fd=smooth.basis(argvals=fitteddata$time,y=log(fitteddata$data_
    spl_2008),fdParobj=basis3)$fd
fou_x2_0=eval.fd(fitteddata$time,fourier3.fd)
fou_x2_1=eval.fd(fitteddata$time,fourier3.fd,Lfdobj=1)
basis3=create.fourier.basis(rangeval=range(fitteddata$time),period=10,
    nbasis=3)
fourier3.fd=smooth.basis(argvals=fitteddata$time,y=log(fitteddata$data_
    lun_2008),fdParobj=basis3)$fd
fou_x3_0=eval.fd(fitteddata$time,fourier3.fd)
fou_x3_1=eval.fd(fitteddata$time,fourier3.fd,Lfdobj=1)
x1hat=fou_x1_0
x1de=fou_x1_1
x2hat=fou_x2_0
x2de=fou_x2_1
```

```
x3hat=fou_x3_0
x3de=fou_x3_1
### Substitude fourier esimation into least square objective function
error=function(vec){
rm=vec[1]
rs=vec[2]
dl=vec[3]
gms=vec[4]
gsl=vec[5]
Dt=10^glkerns(workdataD$TIME,workdataD$LOGDC,x.out=fitteddata$time
    -5+5-3.08,deriv=0)$est
return(sum((x1de-(exp(rm)*Dt-exp(gms)))^2/(log(10)*log(10)))+sum((x2de
    -((exp(rs)*Dt-exp(gsl)) + exp(gms)*exp(x1hat-x2hat)))^2/(log(10)*log
    (10))) + sum((x3de-(0*exp(x1hat-x3hat)+exp(gsl)*exp(x2hat-x3hat) -
    exp(dl)))^2/(log(10)*log(10))));
}
re=list()
for(i in 1:10){
yyy=c(runif(2,-20,-5),runif(1,0.5,2),runif(2,-20,0))
optim(yyy,error,method="Nelder-Mead",control=list(trace=F,maxit=2000,
    abstol=1e-10,reltol=1e-10))->re[[i]]
}
init_par=re[[which.min(unlist(lapply(re,function(x) x$value)))]]$par
init_par
exp(init_par)
#### Use spline to smooth covariate Dt for use in ODE
dt.spline=smooth.spline(workdataD$TIME,workdataD$LOGDC,all.knots=T)

### Use Gauss-Newton method for final fit
yfit=log(c(fitteddata$data_mln_2008,fitteddata$data_spl_2008,fitteddata$
    data_lun_2008))
ODEmodel=function(Time, State, Pars){
with(as.list(c(State, Pars)), {
Dt=10^predict(dt.spline,Time+5-3.08)$y
dX1=(rm*Dt-gms)   #log(Tm)
dX2=(rs*Dt-gsl)+gms*exp(X1-X2)    #log(Ts)
dX3=gsl*exp(X2-X3)-dl #log(Tl)
return(list(c(dX1,dX2,dX3)))
})
}
time.out=fitteddata$time-5
time.0=unique(time.out)
time.freq=time.0-time.0
for(i in seq(length(time.freq))){
time.freq[i]=sum(time.out==time.0[i])}
Themodel=function(X1,X2,X3,theparms){
names(X1)="X1"
names(X2)="X2"
names(X3)="X3"
names(theparms)=c("rm","rs","dl",
"gms","gsl")
tmp=ode(func=ODEmodel,y=c(X1,X2,X3),parms=theparms,times=time.0)
tmp=apply(tmp,2,function(x) rep(x,time.freq))
return(c(tmp[,"X1"],tmp[,"X2"],tmp[,"X3"]))
}
xini=c(X1=log(3.96e+3),X2=log(3.64e+4),X3=log(1.31e+3))
dt.spline=smooth.spline(workdataD$TIME, workdataD$LOGDC, all.knots=T)
themodel=nls(yfit~Themodel(x1,x2,x3,thepar),
control=nls.control(warnOnly=T, tol=1e-4),
start=list(x1=xini[1],x2=xini[2],x3=xini[3],
thepar=exp(init_par)))
summary(themodel)
```

Real data application on CD8$^+$ T cell data

Bibliography

Ali, M. M. and Törn, A. (2004). Population set-based global optimization algorithms: some modifications and numerical studies. *Computers & Operations Research*, 31(10):1703–1725.

Analytics, R. and Weston, S. (2013). foreach: Foreach looping construct for R. *R package version*, 1(1):2013.

Analytics, R. and Weston, S. (2014). doparallel: Foreach parallel adaptor for the parallel package. *R package version*, 1(8).

Ascher, U. M. and Petzold, L. R. (1998). *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam.

Bauer, R. J. and Guzy, S. (2004). Monte carlo parametric expectation maximization (mc-pem) method for analyzing population pharmacokinetic/pharmacodynamic data. In *Advanced Methods of Pharmacokinetic and Pharmacodynamic Systems Analysis Volume 3*, pages 135–163. Springer.

Brown, R. R., Riley, J. D., and Bennett, M. M. (1965). Stability properties of adams-moulton type methods. *Mathematics of Computation*, 19(89):90–96.

Bulirsch, R. and Stoer, J. (2002). *Introduction to numerical analysis*. Springer Heidelberg.

Butcher, J. C. (1963). Coefficients for the study of runge-kutta integration processes. *Journal of the Australian Mathematical Society*, 3(02):185–201.

Carroll, R. J. and Ruppert, D. (1988). *Transformation and weighting in regression*, volume 30. CRC Press.

Chen, T., He, H. L., Church, G. M., et al. (1999). Modeling gene expression with differential equations. In *Pacific symposium on biocomputing*, volume 4, page 4.

Cnaan, A., Laird, N., and Slasor, P. (1997). Tutorial in biostatistics: Using the general linear mixed model to analyse unbalanced repeated measures and longitudinal data. *Stat Med*, 16:2349–2380.

Coddington, E. A. and Levinson, N. (1955). *Theory of ordinary differential equations*. Tata McGraw-Hill Education.

Delyon, B., Lavielle, M., and Moulines, E. (1999). Convergence of a stochastic approximation version of the em algorithm. *Annals of statistics*, pages 94–128.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

Ding, A. A. and Wu, H. Estimation of ODE Parameters Using Constrained Local Polynomial Regression, year = 2014, url = https://www.urmc.rochester.edu/biostat/people/faculty/wusite/publications/peer-reviewed.cfm, urldate = 2016-11-01.

Ding, A. A. and Wu, H. (2014). Estimation of ordinary differential equation parameters using constrained local polynomial regression. *Statistica Sinica*, 24(4):1613.

Englezos, P. and Kalogerakis, N. (2000). *Applied parameter estimation for chemical engineers*. CRC Press.

Eubank, R. L. and Kupresanin, A. (2011). *Statistical Computing in C++ and R*. CRC Press.

Haario, H., Kalachev, L., and Laine, M. (2013). Reduction and identification of dynamic models. simple example: generic receptor model. *Discrete and Continuous Dtnamical Systems - Series B*, 18(2):417–435.

Iba, H. (2008). Inference of differential equation models by genetic programming. *Information Sciences*, 178(23):4453–4468.

Karline Soetaert, T. P. (2011). Differential equations in R. *Tutorial useR conference 2011*.

Lampinen, J. and Storn, R. (2004). *Differential evolution*. Springer.

Liang, H. and Wu, H. (2012). Parameter estimation for differential equation models using a framework of measurement error in regression models. *Journal of the American Statistical Association*.

Liao, J. and Lipsitz, S. R. (2002). A type of restricted maximum likelihood estimator of variance components in generalised linear mixed models. *Biometrika*, 89(2):401–409.

Liu, L., Felgner, F., and Frey, G. (2010). Comparison of 4 numerical solvers for stiff and hybrid systems simulation. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–8. IEEE.

Matis, J., Wehrly, T., and Metzler, C. (1983). On some stochastic formulations and related statistical moments of pharmacokinetic models. *Journal of pharmacokinetics and biopharmaceutics*, 11(1):77–92.

McGoff, K., Mukherjee, S., Pillai, N., et al. (2015). Statistical inference for dynamical systems: A review. *Statistics Surveys*, 9:209–252.

Meza, C., Jaffrézic, F., and Foulley, J.-L. (2007). REML estimation of variance parameters in nonlinear mixed effects models using the SAEM algorithm. *Biometrical Journal*, 49(6):876–888.

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The computer journal*, 7(4):308–313.

Patterson, H. and Thompson, R. (1975). Maximum likelihood estimation of components of variance. In *Proceedings of the 8th international biometric conference*, pages 197–207.

Petzold, L. (1983). Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM journal on scientific and statistical computing*, 4(1):136–148.

Pinheiro, J., Bates, D., DebRoy, S., and Sarkar, D. (2006). nlme: an R package for fitting and comparing gaussian linear and nonlinear mixed-effects models. *See http://www. stats. bris. ac. uk/R.*

Pinheiro, J. C. and Bates, D. M. (1995). Approximations to the log-likelihood function in the nonlinear mixed-effects model. *Journal of computational and Graphical Statistics*, 4(1):12–35.

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

Rios, L. M. and Sahinidis, N. V. (2013). Derivative-free optimization: A review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293.

Soetaert, K., Petzoldt, T., and Setzer, R. W. (2010). Solving differential equations in r: package desolve. *Journal of Statistical Software*, 33.

Spendley, W., Hext, G. R., and Himsworth, F. R. (1962). Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 4(4):441–461.

Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.

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

Wei, G. C. and Tanner, M. A. (1990). A monte carlo implementation of the em algorithm and the poor man's data augmentation algorithms. *Journal of the American statistical Association*, 85(411):699–704.

Wilson, J., Hinson, J., Johnson, V., Woods, T. W., Smith, I., and Brown, R. (1986). Pharmacologic factors contributing to variance in the milk to plasma ratio for acetaminophen in the goat. *Developmental pharmacology and therapeutics*, 10(1):60–72.

Wolfinger, R. (1993). Laplace's approximation for nonlinear mixed models. *Biometrika*, 80(4):791–795.

Wu, H., Kumar, A., Miao, H., Holden-Wiltse, J., Mosmann, T. R., Livingstone, A. M., Belz, G. T., Perelson, A. S., Zand, M. S., and Topham, D. J. (2011). Modeling of influenza-specific cd8+ t cells during the primary response indicates that the spleen is a major source of effectors. *The Journal of Immunology*, 187(9):4474–4482.

Vita

- Education

  MS in Statistics: University of Kentucky, 2013

  BS in Statistics: Shandong University, 2011

- Working Experience

  Graduate Teaching and Research Assistant, University of Kentucky, 2011 - 2016

  Decision Science Professional Intern, Walt Disney World Company, 2015