Metrum Research Group LLC
Phone: 860.735.7043
billg@metrumrg.com

2 Tunxis Road, Suite 112
Tariffville, CT 06081
www.metrumrg.com

Torsten

Torsten: A Pharmacokinetic/Pharmacodynamic Model Library for Stan

User Manual
(Torsten Version 0.85, Stan version 2.18.0)

October 2018

Contents

Development team

Bill Gillespie. billg@metrumrg.com, Metrum Research Group, LLC

Yi Zhang. yiz@metrumrg.com, Metrum Research Group, LLC

Charles Margossian. charles.margossian@columbia.edu, Columbia University, Department of Statistics (formerly Metrum Research Group, LLC)

Acknowledgements

Institutions

We thank Metrum Research Group, Columbia University, and AstraZeneca.

Funding

This work was funded in part by the following organizations:

Office of Naval Research (ONR) contract N00014-16-P-2039. provided as part of the Small Business Technology Transfer (STTR) program. The content of the information presented in this document does not necessarily reflect the position or policy of the Government and no official endorsement should be inferred.

Bill & Melinda Gates Foundation.

Individuals

We thank the Stan Development Team for giving us guidance on how to create new Stan functions and adding features to Stan's core language that facilitate building ODE-based models.

We also thank Kyle Baron and Hunter Ford for helpful advice on coding in C++ and using GitHub, Curtis Johnston for reviewing the User Manual, and Yaming Su for using Torsten and giving us feedback.

Introduction

Stan is an open source probabilistic programing language designed primarily to do Bayesian data analysis [3]. Several of its features make it a powerful tool to specify and fit complex models. First, its language is very expressive and flexible. Secondly, it implements a variant of the No U-Turn Sampler (NUTS), an adaptive Hamiltonian Monte Carlo algorithm, which was proven more efficient than commonly used Markov chains Monte Carlo (MCMC) sampler for high dimensional problems [6, 2]. Our goal is to harness these innovative features and make Stan a better software for pharmacometrics modeling. Our efforts are twofold:

(1) We contribute to the development of features, such as functions that support differential equations based models, and implement them directly into Stan's core language.
(2) We develop Torsten, an extension with specialized pharmacometrics functions.

Throughout the process, we work very closely with the Stan Development Team. We have benefited immensely from their mentorship, advice, and feedback. Just like Stan, Torsten is an open source project that fosters collaborative work. Interested in contributing? Shoot us an e-mail and we will help you help us (billg@metrumrg.com)!

WARNING: The current version of Torsten is a *prototype*. It is being released for review and comment, and to support limited research applications. It has not been rigorously tested and should not be used for critical applications without further testing or cross-checking by comparison with other methods.

We encourage interested users to try Torsten out and are happy to assist. Please report issues, bugs, and feature requests on our GitHub page.

1. Overview

Torsten is a collection of Stan functions to facilitate analysis of pharmacometric data using Stan. The current version includes:

- Specific linear compartment models:
  - One compartment model with first order absorption.
  - Two compartment model with elimination from and first order absorption into central compartment
- General linear compartment model described by a system of first-order linear Ordinary Differential Equations (ODEs).
- General compartment model described by a system of first order ODEs.
- Mix compartment model with PK forcing function described by a linear one or two compartment model.

The models and data format are based on NONMEM®[1]/NMTRAN/PREDPP conventions including:

- Recursive calculation of model predictions

[1] NONMEM® is licensed and distributed by ICON Development Solutions.

  - This permits piecewise constant covariate values
- Bolus or constant rate inputs into any compartment
- Handles single dose and multiple dose histories
- Handles steady state dosing histories
  - Note: The infusion time must be shorter than the inter-dose interval.
- Implemented NMTRAN data items include: TIME, EVID, CMT, AMT, RATE, ADDL, II, SS

In general, all real variables may be passed as model parameters. A few exceptions apply to functions which use a numerical integrator (i.e. the general and the mix compartment models). The below listed cases present technical difficulties, which we expect to overcome in Torsten's next release:

- In the case of a multiple truncated infusion rate dosing regimen:
  - The bioavailability (F) and the amount (AMT) must be fixed.

This library provides Stan language functions that calculate amounts in each compartment, given an event schedule and an ODE system.

2. Implementation summary

- Current v0.85 Torsten is based on Stan v2.18.0.
- All functions are programmed in C++ and are compatible with the Stan math automatic differentiation library [4]
- One and two compartment models are based on analytical solutions of governing ODEs.
- General linear compartment models are based on semi-analytical solutions using the built-in matrix exponential function
- General compartment models are solved numerically using built-in ODE integrators in Stan. The tuning parameters of the solver are adjustable. The steady state solution is calculated using a numerical algebraic solver.
- A mix compartment model's PK forcing function is solved analytically, and its forced ODE system is solved numerically.

3. Development plans

Our current plans for future development of Torsten include the following:

- Build a system to easily share packages of Stan functions (written in C++ or in the Stan language)
- Allow numerical methods to handle bioavailability fraction (F) as parameters in all cases.
- Optimize Matrix exponential functions
  - Function for the action of Matrix Exponential on a vector
  - Hand-coded gradients
  - Special algorithm for matrices with special properties
- Fix issue that arises when computing the adjoint of the lag time parameter (in a dosing compartment) evaluated at $t_{\text{lag}} = 0$.
- Extend formal tests
  - We want more C++ Google unit tests to address cases users may encounter
  - Comparison with simulations from the R package `mrgsolve` and the software NONMEM®
  - Recruit non-developer users to conduct beta testing

4. Changelog

4.1. 0.85 *<2018-10-20 Sat>*.

- Added
  - Dosing rate as parameter
- Changed
  - Update with Stan version 2.18.0.

4.2. 0.84 <*2018-02-24 Sat*>.

- Added
  - Piecewise linear interpolation function.
  - Univariate integral functions.
- Changed
  - Update with Stan version 2.17.1.
  - Minor revisions to User Manual.
  - Bugfixes.

4.3. 0.83 <*2017-08-02 Wed*>.

- Added
  - Work with TorstenHeaders
  - Each chain has a different initial estimate
- Changed
  - User manual
  - Fix misspecification in ODE system for TwoCpt example.
  - Other bugfixes

4.4. 0.82 <*2017-01-29 Sun*>.

- Added
  - Allow parameter arguments to be passed as 1D or 2D arrays
  - More unit tests
  - Unit tests check automatic differentiation against finite differentiation.
- Changed
  - Split the parameter argument into three arguments: pMatrix (parameters for the ODEs – note: for linOdeModel, pMatrix is replaced by the constant rate matrix K), biovar (parameters for the biovariability), and tlag (parameters for the lag time).
  - bugfixes

4.5. 0.81 <*2016-09-27 Tue*>.

- Added linCptModel (linear compartmental model) function

4.6. 0.80a <*2016-09-21 Wed*>.

- Added check_finite statements in pred_1 and pred_2 to reject metropolis proposal if initial conditions are not finite

Installation

We are working with Stan development team to create a system to add and share Stan packages. In the mean time, the current repo contains forked version of Stan with Torsten. The latest version of Torsten (v0.85) is compatible with Stan v2.18.0. Torsten is agnostic to which Stan interface you use. Here we provide command line and R interfaces.

After downloading the project

- https://github.com/metrumresearchgroup/Torsten

to `torsten_path`, set the envionment variable `TORSTEN_PATH` as

```
# in bash
export TORSTEN_PATH=torsten_path
# in csh
setenv TORSTEN_PATH torsten_path
```

0.1. Command line interface. The command line interface `cmdstan` does not require installation. The following command builds a Torsten model `model_name` in `model_path`

```
cd $TORSTEN_PATH/cmdstan; make model_path/model_name
```

0.2. R interface. The R interface is based on rstan, the Stan's interface for R. To install R version of Torsten, at `$TORSTEN_PATH`, in R

```
source('install.R')
```

Please ensure the R toolchain includes a C++ compiler with C++14 support. In particular, R 3.4.0 and later is recommended as it contains toolchain based on gcc 4.9.3. On Windows platform, such a toolchain can be found in Rtools34 and later.

Please ensure `.R/Makevars` constains the following flags

```
CXX14 = g++ -fPIC               # or CXX14 = clang++ -fPIC

CXXFLAGS=-O3 -std=c++1y -mtune=native -march=native -Wno-unused-variable
↪  -Wno-unused-function
CXXFLAGS += -DBOOST_MPL_CFG_NO_PREPROCESSED_HEADERS -DBOOST_MPL_LIMIT_LIST_SIZE=30

CXX14FLAGS=-O3 -std=c++1y -mtune=native -march=native -Wno-unused-variable
↪  -Wno-unused-function
CXX14FLAGS += -DBOOST_MPL_CFG_NO_PREPROCESSED_HEADERS
↪  -DBOOST_MPL_LIMIT_LIST_SIZE=30
```

Fore more information of setting up `makevar` and its functionality, see

- http://dirk.eddelbuettel.com/code/rcpp/Rcpp-package.pdf

For more information of installation troubleshooting, please consult rstan wiki.

0.3. Testing. To test the installation, run

```
./test-torsten.sh --unit        # math unit test
./test-torsten.sh --signature   # stan function # signature test
./test-torsten.sh --model       # R model test, takes long time to finish
```

3

Using Torsten

The reader should have a basic understanding of how Stan works before reading this chapter. There are excellent resources online to get started with Stan

- http://mc-stan.org/documentation

In this section we go through the different functions Torsten adds to Stan. The code for the examples can be found in the `example-models` folder of your `TORSTEN_PATH`.

In general, we work with (i) a compartment model specified by an ODE system, and (ii) an event schedule which describes the treatment patients receive and the times at which measurements are taken. Torsten functions return the drug mass in each compartment at each event of the event schedule. By calling a Torsten function, the user specifies:

- the ODE system describing a compartment PK/PD model
- the method used to solve this ODE system
- the parameters of the model (i.e. coefficients in the ODE, lag times, and bioavailibility fraction)
- the clinical event schedule

1. One Compartment Model

```
matrix PKModelOneCpt(real[] time, real[] amt, real[] rate,
                     real[] ii, int[] evid, int[] cmt,
                     real[] addl, int[] ss, real[] theta,
                     real[] biovar, real[] tlag)
```

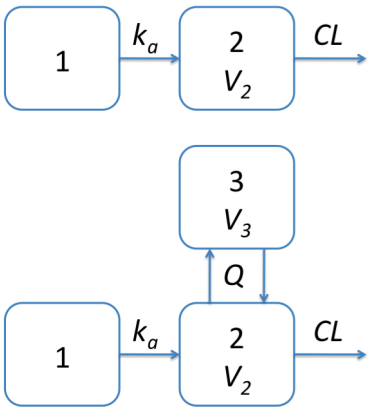

FIGURE 3.1. One and two compartment models with first order absorption implemented in Torsten.

The Torsten function `PKModelOneCpt` returns the drug mass in each compartment at each event of the event schedule. This result is stored inside a matrix. Denoting y the drug mass, the

ODE for the PK one compartment model, with a first-order absorption, is:

$$y_1' = -k_a y_1, \tag{3.1a}$$

$$y_2' = k_a y_1 - \left(\frac{CL}{V_2} + \frac{Q}{V_2}\right) y_2. \tag{3.1b}$$

This ODE gets solved analytically. The arguments are:

- the event arguments `time`, `amt`, `rate`, `ii`, `evid`, `cmt`, `addl`, and `ss`, which describe the event schedule of the clinical trial. The length of each array in the number of events.
- `theta`, the ODE coefficients CL, V_2, and k_a, in that order.
- `biovar`, the bioavailability fraction in each compartment
- `tlag`, the lag time in each compartment.

  Furthermore `theta`, `biovar`, and `tlag` may be either
  - one-dimensional arrays `real[]` if constant for all events, or
  - two-dimensional arrays `real[ , ]` if they vary between events. Specifically, the ith row of the array corresponds to the argument for the time interval (t_{i-1}, t_i). The number of rows equals to the number of events.
- Setting $k_a = 0$ eliminates the first-order absorption.
- The function returns a matrix with `nt` rows and `ncmt` columns, where `nt` is the number of time steps and `ncmt=2` is the number of compartments.

2. Two Compartment Model

```
matrix PKModelTwoCpt(real[] time, real[] amt, real[] rate,
                     real[] ii, int[] evid, int[] cmt,
                     real[] addl, int[] ss, real[] theta,
                     real[] biovar, real[] tlag)
```

The Torsten function `PKModelTwoCpt` (see also Figure 3.1) handles two-compartment PK models, described by the ODEs:

$$y_1' = -k_a y_1 \tag{3.2a}$$

$$y_2' = k_a y_1 - \left(\frac{CL}{V_2} + \frac{Q}{V_2}\right) y_2 + \frac{Q}{V_3} y_3 \tag{3.2b}$$

$$y_3' = \frac{Q}{V_2} y_2 - \frac{Q}{V_3} y_3 \tag{3.2c}$$

The arguments are:

- the event arguments `time`, `amt`, `rate`, `ii`, `evid`, `cmt`, `addl`, and `ss`, which describe the event schedule of the clinical trial. The length of each array in the number of events.
- `theta`, the ODE coefficients CL, Q, V_2, V_3, and k_a, in that order.
- See section 1 regarding model arguments `theta`, `biovar`, and `tlag`.
- Setting ka to 0 eliminates the first-order absorption.
- The function returns a matrix with `nt` rows and `ncmt` columns, where `nt` is the number of time steps and `ncmt=3` is the number of compartments.

2.1. Example. We model drug absorption in a single patient and simulate plasma drug concentrations:

- Multiple Doses: 1250 mg, every 12 hours, for a total of 15 doses

- The drug concentration in the central compartment, c, is measured 0.083, 0.167, 0.25, 0.5, 0.75, 1, 1.5, 2, 4, 6, 8, 10 and 12 hours after the 1st, 2nd, and 15th doses. In addition, measurements are made every 12 hours throughout the trial.

With the plasma concentration $\hat{c}$ solved from two-compartment ODEs in 2, we simulate c according to:

$$\log(c) \sim N\left(\log(\hat{c}), \sigma^2\right)$$
$$(CL, Q, V_2, V_3, ka) = \left(5 \text{ L/h}, 8 \text{ L/h}, 20 \text{ L}, 70 \text{ L}, 1.2 \text{ h}^{-1}\right)$$
$$\sigma^2 = 0.01$$

The data are generated using the R package `mrgsolve` [1].

Code below shows how Torsten function `PKModelTwoCpt` can be used to fit the above model.

```
data{
  int<lower = 1> nt;  // number of events
  int<lower = 1> nObs;  // number of observation
  int<lower = 1> iObs[nObs];  // index of observation

  // NONMEM data
  int<lower = 1> cmt[nt];
  int evid[nt];
  int addl[nt];
  int ss[nt];
  real amt[nt];
  real time[nt];
  real rate[nt];
  real ii[nt];

  vector<lower = 0>[nObs] cObs;  // observed concentration (Dependent Variable)
}

transformed data{
  vector[nObs] logCObs = log(cObs);
  int nTheta = 5;  // number of ODE parameters in Two Compartment Model
  int nCmt = 3;  // number of compartments in model

  // Since we're not trying to evaluate the bio-variability (F) and
  // the lag times, we declare them as data.
  real biovar[nCmt];
  real tlag[nCmt];

  biovar[1] = 1;
  biovar[2] = 1;
  biovar[3] = 1;

  tlag[1] = 0;
  tlag[2] = 0;
  tlag[3] = 0;

}

parameters{
  real<lower = 0> CL;
  real<lower = 0> Q;
  real<lower = 0> V1;
  real<lower = 0> V2;
```

```
  real<lower = 0> ka;
  real<lower = 0> sigma;

}

transformed parameters{
  real theta[nTheta];  // ODE parameters
  vector<lower = 0>[nt] cHat;
  vector<lower = 0>[nObs] cHatObs;
  matrix<lower = 0>[nt, nCmt] x;

  theta[1] = CL;
  theta[2] = Q;
  theta[3] = V1;
  theta[4] = V2;
  theta[5] = ka;

  // PKModelTwoCpt takes in the NONMEM data, followed by the parameter
  // arrays abd returns a matrix with the predicted amount in each
  // compartment at each event.
  x = PKModelTwoCpt(time, amt, rate, ii, evid, cmt, addl, ss,
                    theta, biovar, tlag);

  cHat = col(x, 2) ./ V1; // we're interested in the amount in the second
  ↪  compartment

  cHatObs = cHat[iObs]; // predictions for observed data recors
```

Three MCMC chains of 2000 iterations are simulated. The first 1000 iteration of each chain were discarded. Thus 1000 MCMC samples per chain were used for the subsequent analyses. The MCMC history plots (Figure 3.2) suggest that the 3 chains have converged to common distributions for all of the key model parameters. The fit to the plasma concentration data (Figure 3.5) are in close agreement with the data, which is not surprising since the fitted model is identical to the one used to simulate the data. Similarly the parameter estimates summarized in Table 3.1 and Figure 3.4 are consistent with the values used for simulation.

TABLE 3.1. Summary of the MCMC simulations of the marginal posterior distributions of the model parameters

	mean	se_{mean}	sd	2.5%	25%	50%	75%	97.5%	n_{eff}	Rhat
CL	4.823	0.002	0.092	4.647	4.762	4.823	4.883	5.012	2392.155	1.00
Q	7.596	0.013	0.586	6.479	7.201	7.594	7.977	8.785	1923.939	1.00
V1	21.073	0.069	2.573	16.017	19.352	21.046	22.817	26.097	1385.883	1.00
V2	76.365	0.105	5.611	65.805	72.623	76.172	79.916	87.971	2862.184	1.00
ka	1.231	0.004	0.177	0.907	1.107	1.221	1.344	1.599	1581.825	1.00
sigma	0.109	0.000	0.012	0.089	0.100	0.108	0.116	0.134	2560.112	1.00

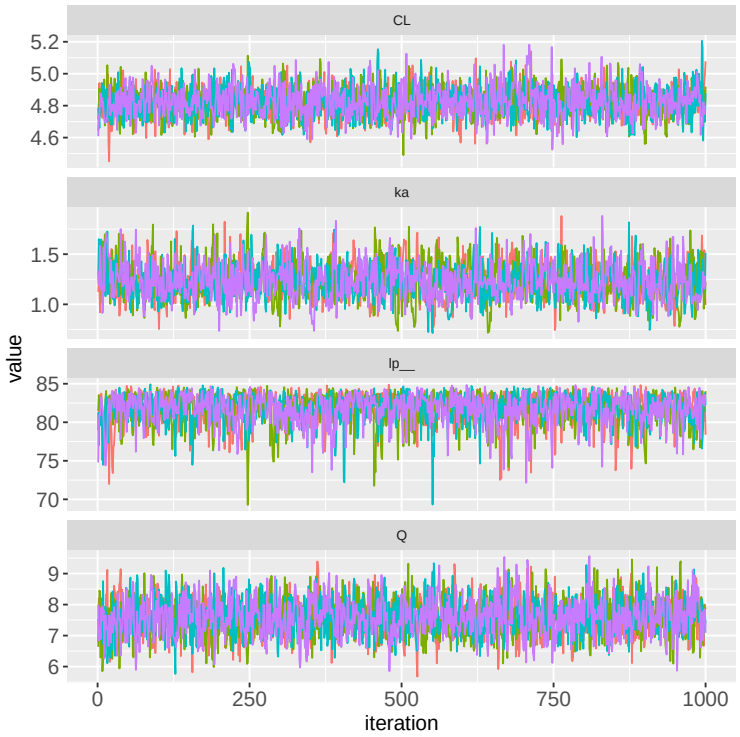

FIGURE 3.2. MCMC history plots for the parameters of a two compartment model with first order absorption (each color corresponds to a different chain)

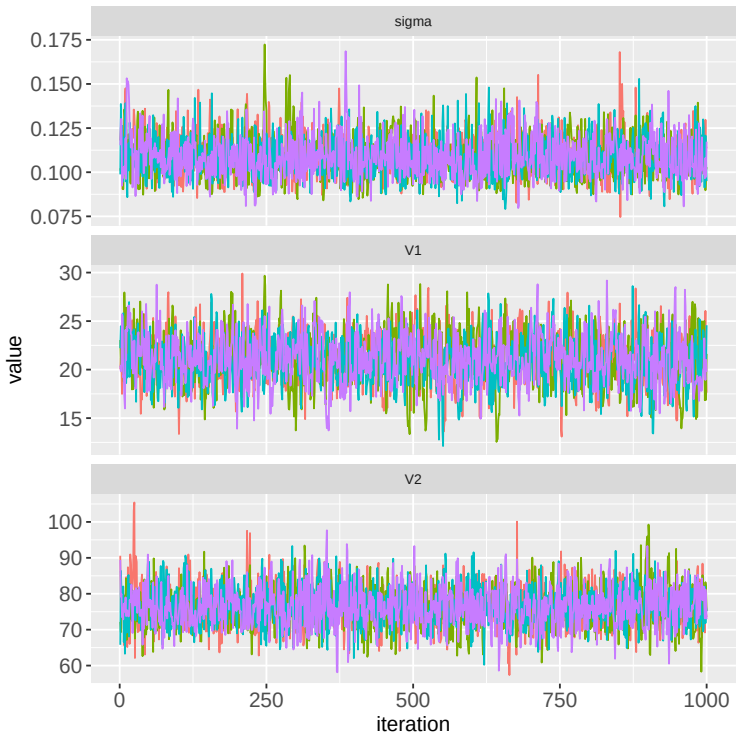

FIGURE 3.3. MCMC history plots for the parameters of a two compartment model with first order absorption (each color corresponds to a different chain)

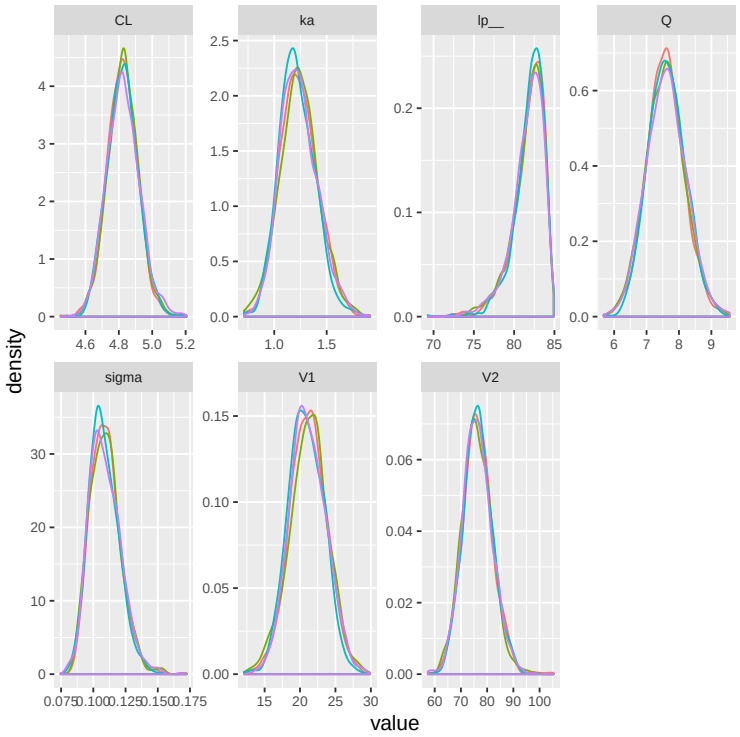

FIGURE 3.4. Posterior Marginal Densities of the Model Parameters of a two compartment model with first order absorption (each color corresponds to a different chain)

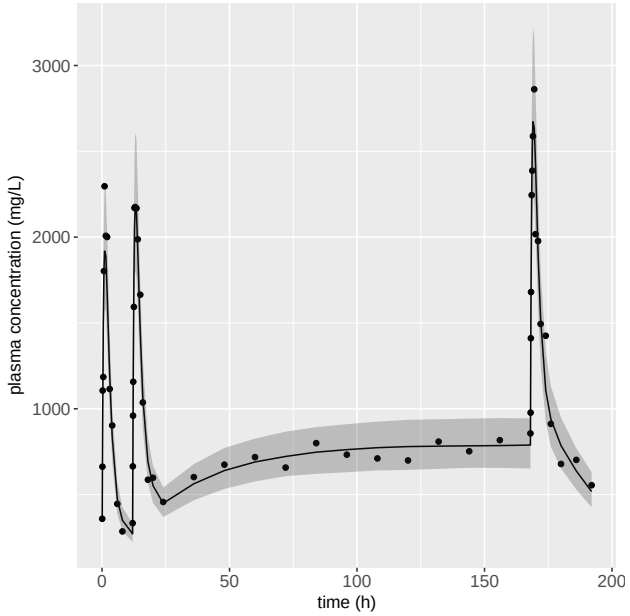

FIGURE 3.5. Predicted (posterior median and 90% credible intervals) and observed plasma drug concentrations of a two compartment model with first order absorption

3. General Linear ODE Model Function

```
matrix linOdeModel(real[] time, real[] amt, real[] rate,
                   real[] ii, int[] evid, int[] cmt,
                   real[] addl, int[] ss,
                   matrix K, real[] biovar, real[] tlag)
```

The Torsten function `linOdeModel` solves a (piecewise) linear ODEs model with coefficients in form of a matrix K

$$y'(t) = Ky(t) \tag{3.3}$$

For example, for a two-compartment model with first order absorption, K would be

$$K = \begin{bmatrix} -k_a & 0 & 0 \\ k_a & -(k_{10} + k_{12}) & k_{21} \\ 0 & k_{12} & -k_{21} \end{bmatrix} \tag{3.4}$$

where $k_{10} = CL/V_2$, $k_{12} = Q/V_2$, and $k_{21} = Q/V_3$.

- `K` contains ODE coefficients and replaces `theta`. If `K` is constant across events, we can pass it as a matrix. If it varies between events, `K` may be passed as an array of matrices. The i^{th} element corresponds to the matrix in the interval `[time[i-1], time[i]]`. The number of elements in the array is the number of events.
- See section 1 regarding the model arguments `biovar`, and `tlag`.
- The function returns a matrix with `nt` rows and `n` columns, where `nt` is the number of time steps and `n` is the size of the square matrix `K`.

3.1. Example. Using `linOdeModel`, the following example fits a two-compartment model with first order absorption.

```
// LinTwoCptModelExample.stan
// Run two compartment model using matrix exponential solution
// Heavily anotated to help new users

data{
  int<lower = 1> nt;  // number of events
  int<lower = 1> nObs;  // number of observations
  int<lower = 1> iObs[nObs];  // index of observation

  // NONMEM data
  int<lower = 1> cmt[nt];
  int evid[nt];
  int addl[nt];
  int ss[nt];
  real amt[nt];
  real time[nt];
  real rate[nt];
  real ii[nt];

  vector<lower = 0>[nObs] cObs;  // observed concentration (dependent variable)
}

transformed data{
  vector[nObs] logCObs = log(cObs);
```

```
  int nCmt = 3;
  real biovar[nCmt];
  real tlag[nCmt];

  for (i in 1:nCmt) {
    biovar[i] = 1;
    tlag[i] = 0;
  }
}

parameters{
  real<lower = 0> CL;
  real<lower = 0> Q;
  real<lower = 0> V1;
  real<lower = 0> V2;
  real<lower = 0> ka;
  real<lower = 0> sigma;

}

transformed parameters{
  matrix[3, 3] K;
  real k10 = CL / V1;
  real k12 = Q / V1;
  real k21 = Q / V2;
  vector<lower = 0>[nt] cHat;
  vector<lower = 0>[nObs] cHatObs;
  matrix<lower = 0>[nt, 3] x;

  K = rep_matrix(0, 3, 3);

  K[1, 1] = -ka;
  K[2, 1] = ka;
  K[2, 2] = -(k10 + k12);
  K[2, 3] = k21;
  K[3, 2] = k12;
  K[3, 3] = -k21;

  // linModel takes in the constant rate matrix, the object theta which
  // contains the biovariability fraction and the lag time of each compartment,
  // and the NONMEM data.
  x = linOdeModel(time, amt, rate, ii, evid, cmt, addl, ss,
                  K, biovar, tlag);
```

4. General ODE Model Function

```
matrix generalOdeModel_rk45(ODE_system, int nCmt,
                            real[] time, real[] amt, real[] rate, real[] ii,
                            int[] evid, int[] cmt, real[] addl, int[] ss,
                            real[] theta, real[] biovar, real[] tlag,
                            real rel_tol, real abs_tol, int max_step);
```

```
matrix generalOdeModel_bdf(ODE_system, int nCmt,
                           real[] time, real[] amt, real[] rate, real[] ii,
                           int[] evid, int[] cmt, real[] addl, int[] ss,
```

```
                                real[] theta, real[] biovar, real[] tlag,
                                real rel_tol, real abs_tol, int max_step);
```

The Torsten functions `generalOdeModel_rk45` and `generalOdeModel_bdf` solve first-order ODEs with user-specified first-order right-hand-side (RHS):

$$y'(t) = f(t, y(t), \theta)$$

where θ corresponds to coefficients in the ODE, which can be passed using `theta`. In the case where the `rate` vector r is non-zero, this equation becomes:

$$y'(t) = f(t, y(t), \theta) + r$$

- User specifies $f(t, y(t))$ by defining `ODE_system` inside the `functions` block (see section 19.2 of the Stan reference manual for details and code below for an example). The user does NOT include the rates in their definition of f. Torsten automatically corrects the derivatives when the rates are non-zero.
- `nCmt` is the number of compartments (or, equivalently, the number of ODEs) in the model.
- `rel_tol`, `abs_tol`, and `max_step` are tuning parameters for the ODE integrator: respectively the relative tolerance, the absolute tolerance, and the maximum number of steps.
- `generalOdeModel_rk45` solves ODEs with Stan's Runge-Kutta ODE solver function `integrate_ode_rk45`.
- `generalOdeModel_rk45` solves ODEs with Stan's Backward Differentiation(BDF) ODE solver function `integrate_ode_bdf`,
- The values to use for the tuning parameters depends on the integrator and the specifics of the ODE system. Reducing the tolerance parameters and increasing the number of steps make for a more robust integrator but can significantly slow down the algorithm. The following can be used as a starting point:
  - `rel_tol = 1e-6`
  - `abs_tol = 1e-6`
  - `max_step = 1e+6`

  for rk45 integrator and
  - `rel_tol = 1e-10`
  - `abs_tol = 1e-10`
  - `max_step = 1e+8`

  for the BDF integrator[1]. Users should be prepared to adjust these values. For additional information, see the Stan User's Manual [8].
- In the case of a multiple truncated infusion rate dosing regimen, the bioavailability `biovar` and the amount `amt` cannot be passed as parameters.
- See section 1 regarding model arguments `theta`, `biovar`, and `tlag`.

4.1. Example. Using `generalOdeModel_rk45`, the following example fits a two-compartment model with first order absorption. User-defined function `twoCptModelODE` describes the RHS of the ODEs.

```
// GenTwoCptModelExample.stan
// Run two compartment model using numerical solution
// Heavily anotated to help new users

functions{
```

[1]These are the default tuning parameters for the integrators. Torsten functions do not have a default values for these parameters. The user must explicitly pass the tuning parameters to `generalOdeModel_*()` .

```
  // define ODE system for two compartmnt model
  real[] twoCptModelODE(real t,
                        real[] x,
                        real[] parms,
                        real[] rate,  // in this example, rate is treated as data
                        int[] dummy){

    // Parameters
    real CL = parms[1];
    real Q = parms[2];
    real V1 = parms[3];
    real V2 = parms[4];
    real ka = parms[5];

    // Re-parametrization
    real k10 = CL / V1;
    real k12 = Q / V1;
    real k21 = Q / V2;

    // Return object (derivative)
    real y[3];  // 1 element per compartment of
                // the model

    // PK component of the ODE system
    y[1] = -ka*x[1];
    y[2] = ka*x[1] - (k10 + k12)*x[2] + k21*x[3];
    y[3] = k12*x[2] - k21*x[3];

    return y;
  }

}
data{
  int<lower = 1> nt;  // number of events
  int<lower = 1> nObs;  // number of observations
  int<lower = 1> iObs[nObs];  // index of observation

  // NONMEM data
  int<lower = 1> cmt[nt];
  int evid[nt];
  int addl[nt];
  int ss[nt];
  real amt[nt];
  real time[nt];
  real rate[nt];
  real ii[nt];

  vector<lower = 0>[nObs] cObs;  // observed concentration (dependent variable)
}

transformed data{
  vector[nObs] logCObs = log(cObs);
  int nTheta = 5;   // number of parameters
  int nCmt = 3;   // number of compartments
  real biovar[nCmt];
  real tlag[nCmt];

  for (i in 1:nCmt) {
```

```
    biovar[i] = 1;
    tlag[i] = 0;
  }
}

parameters{
  real<lower = 0> CL;
  real<lower = 0> Q;
  real<lower = 0> V1;
  real<lower = 0> V2;
  real<lower = 0> ka;
  real<lower = 0> sigma;
}

transformed parameters{
  real theta[nTheta];
  vector<lower = 0>[nt] cHat;
  vector<lower = 0>[nObs] cHatObs;
  matrix<lower = 0>[nt, 3] x;

  theta[1] = CL;
  theta[2] = Q;
  theta[3] = V1;
  theta[4] = V2;
  theta[5] = ka;

  // generalCptModel takes in the ODE system, the number of compartment
  // (here we have a two compartment model with first order absorption, so
  // three compartments), the parameters matrix, the NONEM data, and the tuning
  // parameters (relative tolerance, absolute tolerance, and maximum number of
  ↪  steps)
  // of the ODE integrator. The user can choose between the bdf and the rk45
  ↪  integrator.
  // Returns a matrix with the predicted amount in each compartment
  // at each event.

//  x = generalOdeModel_bdf(twoCptModelODE, 3,
//                          time, amt, rate, ii, evid, cmt, addl, ss,
//                          theta, biovar, tlag,
//                          1e-8, 1e-8, 1e8);

   x = generalOdeModel_rk45(twoCptModelODE, 3,
                           time, amt, rate, ii, evid, cmt, addl, ss,
                           theta, biovar, tlag,
                           1e-6, 1e-6, 1e6);

  cHat = col(x, 2) ./ V1;

  for(i in 1:nObs){
    cHatObs[i] = cHat[iObs[i]];  // predictions for observed data records
  }
}
```

5. Mixed ODE Model Function

```
matrix mixOde1CptModel_rk45(reduced_ODE_system, int nOde,
                            real[] time, real[] amt, real[] rate,
```

```
                              real[] ii, int[] evid, int[] cmt, real[]
                              addl, int[] ss,
                              real[] theta, real[] biovar, real[] tlag,
                              real rel_tol, real abs_tol, real max_step)
```

```
matrix mixOde1CptModel_bdf(reduced_ODE_system, int nOde,
                              real[] time, real[] amt, real[] rate,
                              real[] ii, int[] evid, int[] cmt, real[]
                              addl, int[] ss,
                              real[] theta, real[] biovar, real[] tlag,
                              real rel_tol, real abs_tol, real max_step)
```

```
matrix mixOde2CptModel_rk45(reduced_ODE_system, int nOde,
                              real[] time, real[] amt, real[] rate,
                              real[] ii, int[] evid, int[] cmt, real[]
                              addl, int[] ss,
                              real[] theta, real[] biovar, real[] tlag,
                              real rel_tol, real abs_tol, real max_step)
```

```
matrix mixOde2CptModel_bdf(reduced_ODE_system, int nOde,
                              real[] time, real[] amt, real[] rate,
                              real[] ii, int[] evid, int[] cmt, real[]
                              addl, int[] ss,
                              real[] theta, real[] biovar, real[] tlag,
                              real rel_tol, real abs_tol, real max_step)
```

When the ODE system consists of two subsystems in form of

$$y_1' = f_1(t, y_1),$$
$$y_2' = f_2(t, y_1, y_2),$$

with y_1, y_2, f_1, and f_2 being vector-valued functions, and y_1' independent of y_2, the solution can be accelerated if y_1 admits an analytical solution which can be introduced into the ODE for y_2 for numerical integration. This structure arises in PK/PD models, where y_1 describes a forcing PK function and y_2 the PD effects. In the example of a Friberg-Karlsson semi-mechanistic model [5] (see below), we observe an average speedup of $\sim 47 \pm 18\%$ when using the mix solver in lieu of the numerical integrator [7]. Torsten supports the mixed solver for cases where y_1 solves the ODEs for a One or Two Compartment model with a first-order absorption.

The `reduced_ODE_system` specifies the system we numerically solve, y_2 in the above discussion, also called the *reduced system*. `nOde` is the number of equations in the reduced system. The function that defines a reduced system has an almost identical signature to that used for a full system, but takes one additional argument: y_1, the PK states, i.e. solution to the PK ODEs.

```
real[] reducedODE(real t,       // time
                  real[] y,     // reduced state
                  real[] y1,    // PK states
                  real[] theta, // parameters
                  real[] x_r,   // data (real)
                  int[] x_int)  // data (integer)
```

The four functions of mixed solver correspond to all the permutations Torsten provides when using a forcing One or Two Compartment function, and the Runge-Kutta 4th/5th order (`rk45`) or Backward Differentiation (`bdf`) integration scheme. The mixed ODE functions can be used to compute the steady state solutions supported by the general ODE model functions.

Restrictions regarding which arguments may be passed as parameters for general ODE solvers also apply to mixed solvers.

6. Example

A Friberg-Karlsson Semi-Mechanistic model [5] couples a PK model with a PD effect. In the current example, we use the two compartment model in section 2 for the PK model.

Neutropenia is observed in patients receiving an ME-2 drug. Our goal is to model the relation between neutrophil counts and drug exposure. Using a feedback mechanism, the body maintains the number of neutrophils at a baseline value (Figure 3.6). While in the patient's blood, the drug impedes the production of neutrophils. As a result, the neutrophil count goes down. After the drug clears out, the feedback mechanism kicks in and brings the neutrophil count back to baseline.

$$\log(ANC_i) \sim N(\log(Circ), \sigma^2_{ANC}) \tag{3.5}$$

$$Circ = f_{FK}(MTT, Circ_0, \alpha, \gamma, c) \tag{3.6}$$

$$(MTT, Circ_0, \alpha, \gamma, ktr) = (125, 5.0, 3 \times 10^{-4}, 0.17) \tag{3.7}$$

$$\sigma^2_{ANC} = 0.001 \tag{3.8}$$

where c is the drug concentration in the blood we get from the Two Compartment model, and $Circ$ is obtained by solving the following system of nonlinear ODEs:

$$y'_{\text{prol}} = k_{\text{prol}} y_{\text{prol}} (1 - E_{\text{drug}}) \left(\frac{Circ_0}{y_{\text{circ}}} \right)^{\gamma} - k_{\text{tr}} y_{\text{prol}} \tag{3.9a}$$

$$y'_{\text{trans1}} = k_{\text{tr}} y_{\text{prol}} - k_{\text{tr}} y_{\text{trans1}} \tag{3.9b}$$

$$y'_{\text{trans2}} = k_{\text{tr}} y_{\text{trans1}} - k_{\text{tr}} y_{\text{trans2}} \tag{3.9c}$$

$$y'_{\text{trans3}} = k_{\text{tr}} y_{\text{trans2}} - k_{\text{tr}} y_{\text{trans3}} \tag{3.9d}$$

$$y'_{\text{circ}} = k_{\text{tr}} y_{\text{trans3}} - k_{\text{tr}} y_{\text{circ}} \tag{3.9e}$$

where $E_{drug} = \alpha c$.

The ODEs specifying the Two Compartment Model (Equation (3.2a)) do not depend on the PD ODEs (Equation (3.9)) and can be solved analytically using Torsten's `PKModelTwoCpt` function. We therefore specify our model using a mixed solver function. We do not expect our system to be stiff and use the Runge-Kutta 4th/5th order integrator.

```
functions{
  real[] FK_ODE(real t,
                real[] y,
                real[] y_pk,
                real[] theta,
                real[] rdummy,
                int[] idummy){
    /* PK variables */
    real VC = theta[3];

    /* PD variable */
    real mtt      = theta[6];
    real circ0    = theta[7];
```

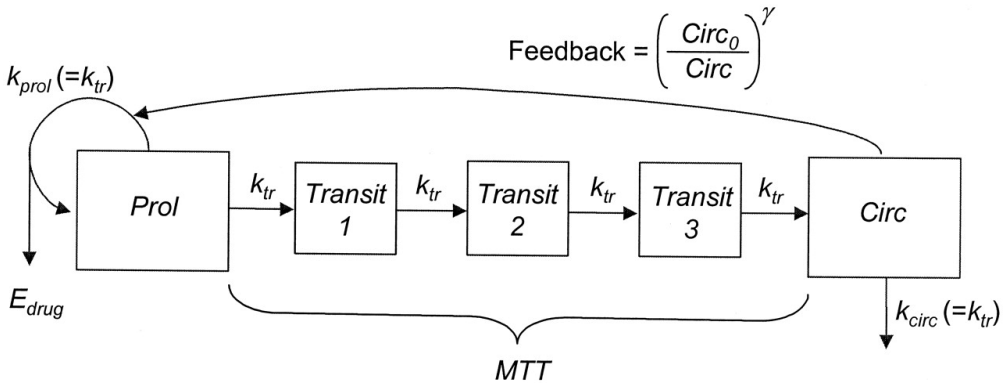

FIGURE 3.6. Friberg-Karlsson semi-mechanistic Model.

```
    real alpha    = theta[8];
    real gamma    = theta[9];
    real ktr      = 4.0 / mtt;
    real prol     = y[1] + circ0;
    real transit1 = y[2] + circ0;
    real transit2 = y[3] + circ0;
    real transit3 = y[4] + circ0;
    real circ     = fmax(machine_precision(), y[5] + circ0);
    real conc     = y_pk[2] / VC;
    real EDrug    = alpha * conc;

    real dydt[5];

    dydt[1] = ktr * prol * ((1 - EDrug) * ((circ0 / circ)^gamma) - 1);
    dydt[2] = ktr * (prol - transit1);
    dydt[3] = ktr * (transit1 - transit2);
    dydt[4] = ktr * (transit2 - transit3);
    dydt[5] = ktr * (transit3 - circ);

    return dydt;
  }
}

data{
  int<lower = 1> nt;
  int<lower = 1> nObsPK;
  int<lower = 1> nObsPD;
  int<lower = 1> iObsPK[nObsPK];
  int<lower = 1> iObsPD[nObsPD];
  real<lower = 0> amt[nt];
  int<lower = 1> cmt[nt];
  int<lower = 0> evid[nt];
  real<lower = 0> time[nt];
  real<lower = 0> ii[nt];
  int<lower = 0> addl[nt];
  int<lower = 0> ss[nt];
  real rate[nt];
  vector<lower = 0>[nObsPK] cObs;
  vector<lower = 0>[nObsPD] neutObs;

  real<lower = 0> circ0Prior;
```

```
  real<lower = 0> circ0PriorCV;
  real<lower = 0> mttPrior;
  real<lower = 0> mttPriorCV;
  real<lower = 0> gammaPrior;
  real<lower = 0> gammaPriorCV;
  real<lower = 0> alphaPrior;
  real<lower = 0> alphaPriorCV;
}

transformed data{
  int nOde = 5;
  vector[nObsPK] logCObs;
  vector[nObsPD] logNeutObs;
//  int idummy[0];
//  real rdummy[0];

  int nTheta;
  int nIIV;

  int n;                          /* ODE dimension */
  real rtol;
  real atol;
  int max_step;

  n = 8;
  rtol = 1e-8;
  atol = 1e-8;
  max_step = 100000;

  logCObs = log(cObs);
  logNeutObs = log(neutObs);

  nIIV = 7; // parameters with IIV
  nTheta = 9; // number of parameters
}

parameters{

  real<lower = 0> CL;
  real<lower = 0> Q;
  real<lower = 0> VC;
  real<lower = 0> VP;
  real<lower = 0> ka;
  real<lower = 0> mtt;
  real<lower = 0> circ0;
  real<lower = 0> alpha;
  real<lower = 0> gamma;
  real<lower = 0> sigma;
  real<lower = 0> sigmaNeut;

  // IIV parameters
  cholesky_factor_corr[nIIV] L;
  vector<lower = 0>[nIIV] omega;
}

transformed parameters{
  vector[nt] cHat;
  vector<lower = 0>[nObsPK] cHatObs;
```

```
  vector[nt] neutHat;
  vector<lower = 0>[nObsPD] neutHatObs;
  real<lower = 0> theta[nTheta];
  matrix[nt, nOde + 3] x;
  real biovar[nTheta];
  real tlag[nTheta];

  for (i in 1:nTheta) {
    biovar[i] = 1.0;
    tlag[i] = 0.0;
  }

  theta[1] = CL;
  theta[2] = Q;
  theta[3] = VC;
  theta[4] = VP;
  theta[5] = ka;
  theta[6] = mtt;
  theta[7] = circ0;
  theta[8] = alpha;
  theta[9] = gamma;

  x = mixOde2CptModel_rk45(FK_ODE, nOde, time, amt, rate, ii, evid, cmt, addl, ss,
  ↪  theta, biovar, tlag, rtol, atol, max_step);

  cHat = col(x, 2) / VC;
  neutHat = col(x, 8) + circ0;

  for(i in 1:nObsPK) cHatObs[i]    = cHat[iObsPK[i]];
  for(i in 1:nObsPD) neutHatObs[i] = neutHat[iObsPD[i]];

}

model {

  // Priors
  CL    ~ normal(0, 20);
  Q     ~ normal(0, 20);
  VC    ~ normal(0, 100);
  VP    ~ normal(0, 1000);
  ka    ~ normal(0, 5);
  sigma ~ cauchy(0, 1);

  mtt       ~ lognormal(log(mttPrior), mttPriorCV);
  circ0     ~ lognormal(log(circ0Prior), circ0PriorCV);
  alpha     ~ lognormal(log(alphaPrior), alphaPriorCV);
  gamma     ~ lognormal(log(gammaPrior), gammaPriorCV);
  sigmaNeut ~ cauchy(0, 1);

  // Parameters for Matt's trick
  L ~ lkj_corr_cholesky(1);
  omega ~ cauchy(0, 1);

  // observed data likelihood
  logCObs ~ normal(log(cObs), sigma);
  logNeutObs ~ normal(log(neutObs), sigmaNeut);
}
```

7. Univariate integral

```
real univariate_integral_rk45(f, t0, t1, theta, x_r, x_i)
```

```
real univariate_integral_bdf(f, t0, t1, theta, x_r, x_i)
```

Based on the ODE solver capability in Stan, Torsten provides functions calculating the integral of a univariate function. The integrand function f must follow the signature

```
  real f(real t, real[] theta, real[] x_r, int[] x_i) {
    /* ... */
}
```

7.1. Example. This example shows how to use `univariate_integral_rk45` to calculate the integral of a quadratic function.

```
functions {
  real fun_ord2(real t, real[] theta, real[] x_r, int[] x_i) {
    real a = 2.3;
    real b = 2.0;
    real c = 1.5;
    real res;
    res = a + b * t + c * t * t;
    return res;
  }
}
data {
  real t0;
  real t1;
  real dtheta[2];
  real x_r[0];
  int x_i[0];
}
transformed data {
  real univar_integral;
  univar_integral = univariate_integral_rk45(func, t0, t1, dtheta,
                          x_r, x_i);
}
/* ... */
```

8. Piecewise linear interpolation

```
real linear_interpolation(real xout, real[] x, real[] y)
```

```
real[] linear_interpolation(real[] xout, real[] x, real[] y)
```

Torsten also provides a `linear_interpolation` function for piecewise linear interpolation over a set of x, y pairs. It returns the values of a piecewise linear function at specified values `xout` of the first function argument. The function is specified in terms of a set of x, y pairs. Specifically,

`linear_interpolation` implements the following function

$$y_{\text{out}} = \begin{cases} y_1, & x_{\text{out}} < x_1 \\ y_i + \frac{y_{i+1}-y_i}{x_{i+1}-x_i}\left(x_{\text{out}} - x_i\right), & x_{\text{out}} \in [x_i, x_{i+1}) \\ y_n, & x_{\text{out}} \geq x_n \end{cases}$$

- The x values must be in increasing order, i.e. $x_i < x_{i+1}$.
- All three arguments may be data or parameters.

8.1. Example. This example illustrates how to use `linear_intepolationi` to fit a piecewise linear function to a data set consisting of (x, y) pairs.

```
data{
  int nObs;
  real xObs[nObs];
  real yObs[nObs];
  int nx;
  int nPred;
  real xPred[nPred];
}

transformed data{
  real xmin = min(xObs);
  real xmax = max(xObs);
}

parameters{
  real y[nx];
  real<lower = 0> sigma;
  simplex[nx - 1] xSimplex;
}

transformed parameters{
  real yHat[nObs];
  real x[nx];

  x[1] = xmin;
  x[nx] = xmax;
  for(i in 2:(nx-1))
    x[i] = x[i-1] + xSimplex[i-1] * (xmax - xmin);

  yHat = linear_interpolation(xObs, x, y);
}

model{
  xSimplex ~ dirichlet(rep_vector(1, nx - 1));
  y ~ normal(0, 25);
  yObs ~ normal(yHat, sigma);
}

generated quantities{
  real yHatPred[nPred];
  real yPred[nPred];

  yHatPred = linear_interpolation(xPred, x, y);
  for(i in 1:nPred)
    yPred[i] = normal_rng(yHatPred[i], sigma);
}
```

Additional examples

1. Effect Compartment Population Model

Here we expand the example in 2 to a population model fitted to the combined data from phase I and phase IIa studies. The parameters exhibit inter-individual variations (IIV), due to both random effects and to the patients' body weight, treated as a covariate and denoted bw.

1.1. Population Model for Plasma Drug Concentration c.

$$\log\left(c_{ij}\right) \sim N\left(\log\left(\widehat{c}_{ij}\right), \sigma^2\right),$$
$$\widehat{c}_{ij} = f_{2cpt}\left(t_{ij}, D_j, \tau_j, CL_j, Q_j, V_{1j}, V_{2j}, k_{aj}\right),$$
$$\log\left(CL_j, Q_j, V_{ssj}, k_{aj}\right) \sim N\left(\log\left(\widehat{CL}\left(\frac{bw_j}{70}\right)^{0.75}, \widehat{Q}\left(\frac{bw_j}{70}\right)^{0.75}, \widehat{V}_{ss}\left(\frac{bw_j}{70}\right), \widehat{k}_a\right), \Omega\right),$$
$$V_{1j} = f_{V_1} V_{ssj},$$
$$V_{2j} = \left(1 - f_{V_1}\right) V_{ssj},$$
$$\left(\widehat{CL}, \widehat{Q}, \widehat{V}_{ss}, \widehat{k}_a, f_{V_1}\right) = \left(10 \text{ L/h}, 15 \text{ L/h}, 140 \text{ L}, 2 \text{ h}^{-1}, 0.25\right),$$
$$\Omega = \begin{pmatrix} 0.25^2 & 0 & 0 & 0 \\ 0 & 0.25^2 & 0 & 0 \\ 0 & 0 & 0.25^2 & 0 \\ 0 & 0 & 0 & 0.25^2 \end{pmatrix},$$
$$\sigma = 0.1$$

Furthermore we add a fourth compartment in which we measure a PD effect(Figure 4.1).

1.2. Effect Compartment Model for PD response R.

$$R_{ij} \sim N\left(\widehat{R}_{ij}, \sigma_R^2\right),$$
$$\widehat{R}_{ij} = \frac{E_{max} c_{eij}}{EC_{50j} + c_{eij}},$$
$$c'_{e\cdot j} = k_{e0j}\left(c_{\cdot j} - c_{e\cdot j}\right),$$
$$\log\left(EC_{50j}, k_{e0j}\right) \sim N\left(\log\left(\widehat{EC}_{50}, \widehat{k}_{e0}\right), \Omega_R\right),$$
$$\left(E_{max}, \widehat{EC}_{50}, \widehat{k}_{e0}\right) = \left(100, 100.7, 1\right),$$
$$\Omega_R = \begin{pmatrix} 0.2^2 & 0 \\ 0 & 0.25^2 \end{pmatrix}, \quad \sigma_R = 10.$$

The PK and the PD data are simulated using the following treatment.

- Phase I study
  - Single dose and multiple doses
  - Parallel dose escalation design
  - 25 subjects per dose

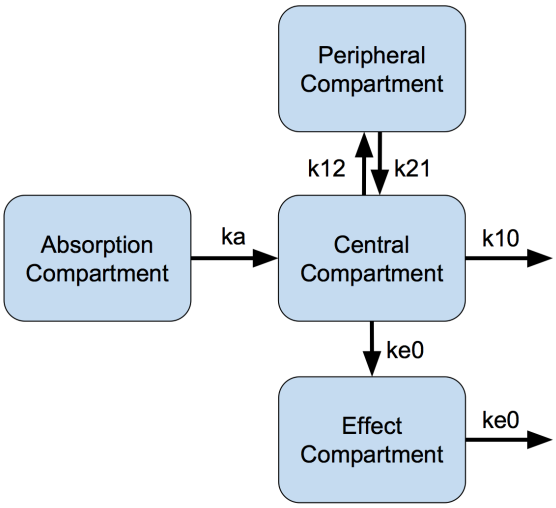

FIGURE 4.1. Effect Compartment Model

  - Single doses: 1.25, 5, 10, 20, and 40 mg
  - PK: plasma concentration of parent drug (c)
  - PD response: Emax function of effect compartment concentration (R)
  - PK and PD measured at 0.083, 0.167, 0.25, 0.5, 0.75, 1, 2, 3, 4, 6, 8, 12, 18, and 24 hours
- Phase IIa trial in patients
  - 100 subjects
  - Multiple doses: 20 mg
  - sparse PK and PD data (3-6 samples per patient)

The model is simultaneously fitted to the PK and the PD data. For this effect compartment model, we construct a constant rate matrix and use `linOdeModel`. Correct use of Torsten requires the user pass the entire event history (observation and dosing events) for an individual to the function. Thus the Stan model shows the call to `linOdeModel` within a loop over the individual subjects rather than over the individual observations.

```
transformed parameters{
  vector<lower = 0>[nRandom] thetaHat;
  cov_matrix[nRandom] Omega;
  real<lower = 0> CL[nSubjects];
  real<lower = 0> Q[nSubjects];
  real<lower = 0> V1[nSubjects];
  real<lower = 0> V2[nSubjects];
  real<lower = 0> ka[nSubjects];
  real<lower = 0> ke0[nSubjects];
  real<lower = 0> EC50[nSubjects];
  matrix[nCmt, nCmt] K;
  real k10;
  real k12;
  real k21;
  vector<lower = 0>[nt] cHat;
  vector<lower = 0>[nObs] cHatObs;
```

```
  vector<lower = 0>[nt] respHat;
  vector<lower = 0>[nObs] respHatObs;
  vector<lower = 0>[nt] ceHat;
  matrix[nt, nCmt] x;

  thetaHat[1] = CLHat;
  thetaHat[2] = QHat;
  thetaHat[3] = V1Hat;
  thetaHat[4] = V2Hat;
  thetaHat[5] = kaHat;

  Omega = quad_form_diag(rho, omega); ## diag_matrix(omega) * rho *
  ↪  diag_matrix(omega)

  for(j in 1:nSubjects){
    CL[j] = exp(logtheta[j, 1]) * (weight[j] / 70)^0.75;
    Q[j] = exp(logtheta[j, 2]) * (weight[j] / 70)^0.75;
    V1[j] = exp(logtheta[j, 3]) * weight[j] / 70;
    V2[j] = exp(logtheta[j, 4]) * weight[j] / 70;
    ka[j] = exp(logtheta[j, 5]);
    ke0[j] = exp(logKe0[j]);
    EC50[j] = exp(logEC50[j]);

    k10 = CL[j] / V1[j];
    k12 = Q[j] / V1[j];
    k21 = Q[j] / V2[j];

    K = rep_matrix(0, nCmt, nCmt);

    K[1, 1] = -ka[j];
    K[2, 1] = ka[j];
    K[2, 2] = -(k10 + k12);
    K[2, 3] = k21;
    K[3, 2] = k12;
    K[3, 3] = -k21;
    K[4, 2] = ke0[j];
    K[4, 4] = -ke0[j];

    x[start[j]:end[j],] = linOdeModel(time[start[j]:end[j]],
                                      amt[start[j]:end[j]],
                                      rate[start[j]:end[j]],
                                      ii[start[j]:end[j]],
                                      evid[start[j]:end[j]],
                                      cmt[start[j]:end[j]],
                                      addl[start[j]:end[j]],
                                      ss[start[j]:end[j]],
                                      K, biovar, tlag);

    cHat[start[j]:end[j]] = 1000 * x[start[j]:end[j], 2] ./ V1[j];
    ceHat[start[j]:end[j]] = 1000 * x[start[j]:end[j], 4] ./ V1[j];
    respHat[start[j]:end[j]] = 100 * ceHat[start[j]:end[j]] ./
       (EC50[j] + ceHat[start[j]:end[j]]);
  }

  cHatObs = cHat[iObs];
  respHatObs = respHat[iObs];

}
```

1.3. Results. We use the same diagnosis tools as for the previous examples. The MCMC history plots (Figure 4.2) suggest the 4 chains have converged to common distributions. We note some minor auto-correlations for $lp\_$ (the log posterior) and for IIV parameters: specifically Ω_{ke_0} and ρ. The correlation matrix ρ does not explicitly appear in the model, but it is used to construct Ω, which parametrizes the PK IIV. The fits to the plasma concentration (Figure 4.4) are in close agreement with the data, notably for the sparse data case (phase IIa study). The fits to the PD data (Figure 4.5) look good, though the data is more noisy. The model reflects the noise by producing larger credible intervals. The estimated values of the parameters are consistent with the values used to simulate the data (Table 4.1) and Figure 4.3).

TABLE 4.1. Summary of the MCMC simulations of the marginal posterior distributions of the model parameters for the effect compartment model example.

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
lp__	-201.282	10.073	84.189	-333.764	-259.017	-213.416	-154.381	8.549	69.850	1.044
CLHat	10.095	0.003	0.201	9.712	9.958	10.096	10.231	10.483	4000.000	0.999
QHat	14.867	0.014	0.357	14.182	14.620	14.862	15.106	15.563	678.208	1.007
V1Hat	34.188	0.067	1.089	31.940	33.494	34.214	34.918	36.251	267.748	1.016
V2Hat	103.562	0.076	2.925	98.031	101.600	103.454	105.472	109.583	1488.296	1.001
kaHat	1.930	0.004	0.077	1.771	1.880	1.933	1.982	2.076	334.888	1.014
ke0Hat	1.050	0.001	0.044	0.967	1.020	1.051	1.078	1.137	1164.741	1.000
EC50Hat	104.337	0.040	2.100	100.169	102.909	104.345	105.768	108.351	2744.041	1.000
sigma	0.099	0.000	0.002	0.095	0.097	0.099	0.100	0.103	1906.342	1.002
sigmaResp	10.156	0.003	0.197	9.779	10.023	10.154	10.286	10.552	4000.000	1.000
omega[1]	0.270	0.000	0.016	0.241	0.259	0.269	0.280	0.302	4000.000	1.001
omega[2]	0.231	0.001	0.021	0.192	0.217	0.230	0.245	0.275	531.512	1.006
omega[3]	0.219	0.002	0.031	0.158	0.199	0.218	0.238	0.281	158.198	1.017
omega[4]	0.267	0.001	0.026	0.218	0.249	0.266	0.284	0.319	684.870	1.001
omega[5]	0.285	0.002	0.037	0.214	0.259	0.284	0.309	0.361	284.545	1.009
omegaKe0	0.271	0.003	0.047	0.183	0.239	0.271	0.303	0.363	217.350	1.007
omegaEC50	0.213	0.001	0.021	0.174	0.199	0.213	0.227	0.255	1190.193	1.000
rho[1,2]	0.194	0.003	0.100	-0.011	0.127	0.195	0.265	0.379	1000.772	1.004
rho[1,3]	-0.157	0.005	0.126	-0.395	-0.243	-0.157	-0.072	0.088	677.709	1.001
rho[2,3]	0.079	0.012	0.155	-0.227	-0.024	0.082	0.181	0.384	180.306	1.021
rho[1,4]	-0.107	0.003	0.112	-0.319	-0.183	-0.110	-0.032	0.118	1081.932	1.002
rho[2,4]	0.194	0.005	0.126	-0.062	0.110	0.199	0.282	0.428	623.035	1.007
rho[3,4]	0.796	0.008	0.094	0.592	0.737	0.808	0.867	0.940	152.112	1.033
rho[1,5]	0.023	0.006	0.135	-0.232	-0.068	0.024	0.115	0.285	564.687	1.003
rho[2,5]	0.119	0.011	0.160	-0.188	0.008	0.118	0.224	0.438	226.174	1.014
rho[3,5]	-0.246	0.018	0.202	-0.663	-0.382	-0.237	-0.105	0.133	119.465	1.021
rho[4,5]	-0.288	0.009	0.155	-0.576	-0.396	-0.291	-0.183	0.014	275.549	1.009

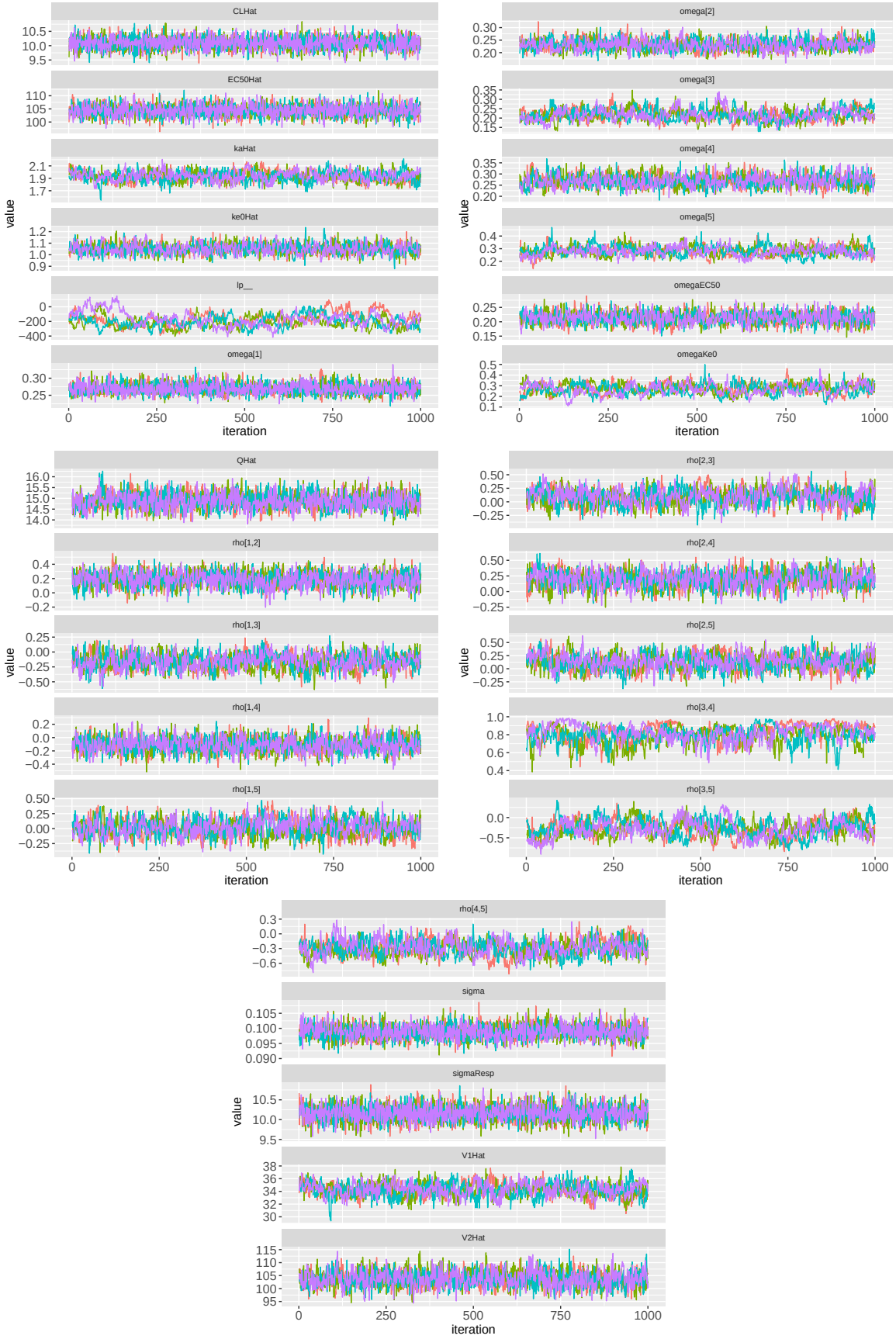

FIGURE 4.2. MCMC history plots for the parameters of an Effect Compartment Model (each color corresponds to a different chain) for example 2

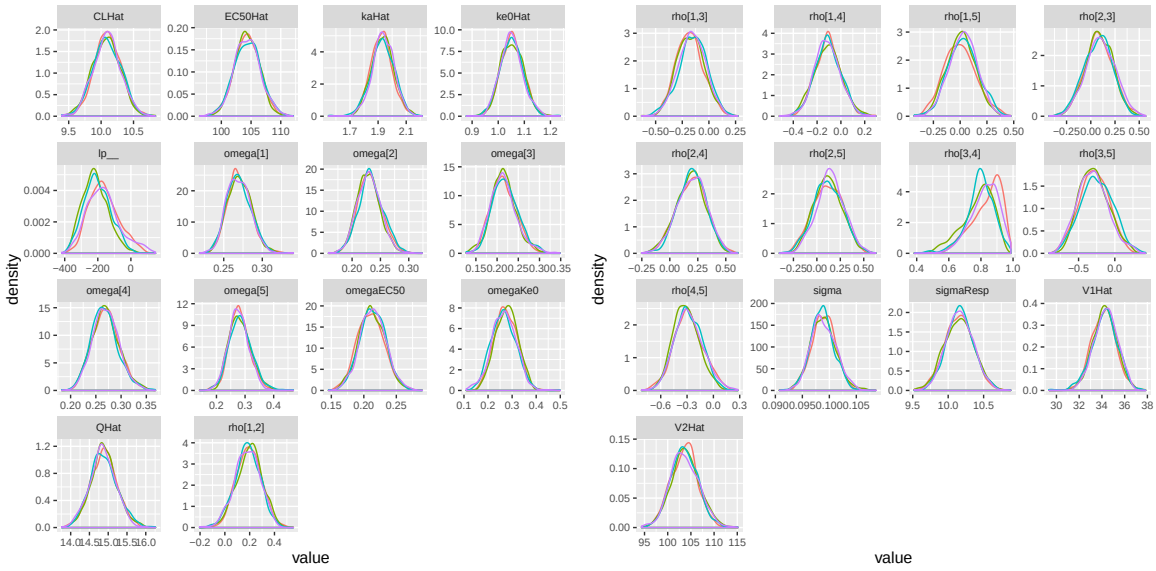

FIGURE 4.3. Posterior Marginal Densities of the Model Parameters of an Effect Compartment Model (each color corresponds to a different chain) for example 2

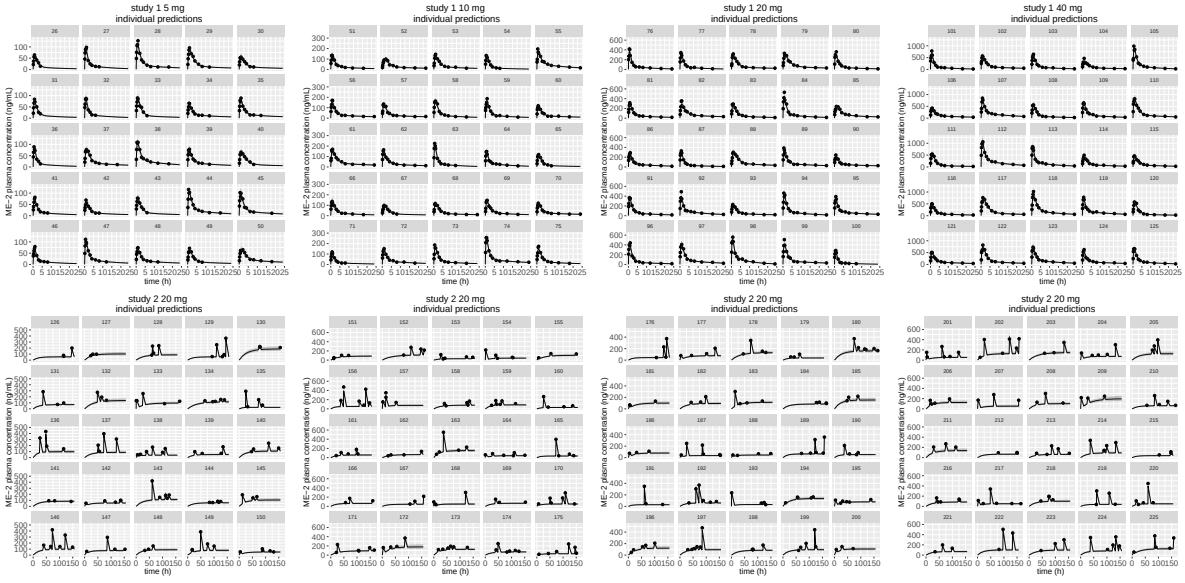

FIGURE 4.4. Predicted (posterior median and 90 % credible intervals) and observed plasma drug concentrations for example 2 for an Effect Compartment Model

FIGURE 4.5. Predicted (posterior median and 90 % credible intervals) and observed PD Response for example 2

2. Friberg-Karlsson Semi-Mechanistic Population Model

We now return to the example in Section 5 and extend it to a population model. While we recommend using the mixed solver, for completeness we show how to specify the model with the `generalOdeModel` function. We leave it as an exercise to the reader to rewrite the model with `mixOde2CptModel`.

2.1. Friberg-Karlsson Population Model for drug-induced myelosuppression (ANC).

$$\log(ANC_{ij}) \sim N(Circ_{ij}, \sigma^2_{ANC}),$$

$$\log\left(MTT_j, Circ_{0j}, \alpha_j\right) \sim N\left(\log\left(\widehat{MTT}, \widehat{Circ_0}, \widehat{\alpha}\right), \Omega_{ANC}\right),$$

$$\left(\widehat{MTT}, \widehat{Circ_0}, \widehat{\alpha}, \gamma\right) = (125, 5, 2, 0.17),$$

$$\Omega_{ANC} = \begin{pmatrix} 0.2^2 & 0 & 0 \\ 0 & 0.35^2 & 0 \\ 0 & 0 & 0.2^2 \end{pmatrix},$$

$$\sigma_{ANC} = 0.1,$$

$$\Omega_{PK} = \begin{pmatrix} 0.25^2 & 0 & a0 & 0 & 0 \\ 0 & 0.4^2 & 0 & 0 & 0 \\ 0 & 0 & 0.25^2 & 0 & 0 \\ 0 & 0 & 0 & 0.4^2 & 0 \\ 0 & 0 & 0 & 0 & 0.25^2 \end{pmatrix}$$

The PK and the PD data are simulated using the following treatment.

- Phase IIa trial in patients
  - Multiple doses: 80,000 mg
  - Parallel dose escalation design
  - 15 subjects
  - PK: plasma concentration of parent drug (c)
  - PD response: Neutrophil count (ANC)
  - PK measured at 0.083, 0.167, 0.25, 0.5, 0.75, 1, 2, 3, 4, 6, 8, 12, 18, and 24 hours
  - PD measured once every two days for 28 days.

Once again, we simultaneously fit the model to the PK and the PD data. Note that from a computational perspective, this is a much more difficult problem than in the previous example. The nonlinear nature of the ODEs forces us to use a numerical solver, which is significantly slower than the linear methods we have employed so far. Because the ODE system of interest is non-stiff, we use `genOdeModel_rk45`.

The two code snippets below show the definition of the ODEs system and the skeleton of the solution process in Stan's `transformed parameters` block.

```
functions{

    real[] twoCptNeutModelODE(real t,
                          real[] x,
                          real[] parms,
                          real[] rdummy,
                          int[] idummy){
                real CL = parms[1];
                real Q = parms[2];
                real V1 = parms[3];
                real V2 = parms[4];
```

```
                real ka = parms[5];
                real mtt = parms[6];
                real circ0 = parms[7];
                real gamma = parms[8];
                real alpha = parms[9];

    real k10 = CL / V1;
    real k12 = Q / V1;
    real k21 = Q / V2;
    real ktr = 4 / mtt;

    real dxdt[8];
    real conc;
    real EDrug;
    real transit1;
    real transit2;
    real transit3;
    real circ;
    real prol;

    dxdt[1] = -ka * x[1];
    dxdt[2] = ka * x[1] - (k10 + k12) * x[2] + k21 * x[3];
    dxdt[3] = k12 * x[2] - k21 * x[3];
    conc = x[2] / V1;
    EDrug = alpha * conc;
    // x[4], x[5], x[6], x[7] and x[8] are differences from circ0.
    prol = x[4] + circ0;
    transit1 = x[5] + circ0;
    transit2 = x[6] + circ0;
    transit3 = x[7] + circ0;
    circ = fmax(machine_precision(), x[8] + circ0); // Device for implementing a
    ↪  modeled
                                                    // initial condition
    dxdt[4] = ktr * prol * ((1 - EDrug) * ((circ0 / circ)^gamma) - 1);
    dxdt[5] = ktr * (prol - transit1);
    dxdt[6] = ktr * (transit1 - transit2);
    dxdt[7] = ktr * (transit2 - transit3);
    dxdt[8] = ktr * (transit3 - circ);

    return dxdt;
  }

}
```

```
transformed parameters{
  vector[nt] cHat;
  vector[nObsPK] cHatObs;
  vector[nt] neutHat;
  vector[nObsPD] neutHatObs;
  matrix[nt, nCmt] x;
  real<lower = 0> parms[nTheta]; # The [1] indicates the parameters are constant

  ## variables for Matt's trick
  vector<lower = 0>[nIIV] thetaHat;
  matrix<lower = 0>[nSubjects, nIIV] thetaM;
```

```
  ## Matt's trick to use unit scale
  thetaHat[1] = CLHat;
  thetaHat[2] = QHat;
  thetaHat[3] = V1Hat;
  thetaHat[4] = V2Hat;
  thetaHat[5] = mttHat;
  thetaHat[6] = circ0Hat;
  thetaHat[7] = alphaHat;
  thetaM = (rep_matrix(thetaHat, nSubjects) .*
            exp(diag_pre_multiply(omega, L * etaStd)))';

  for(i in 1:nSubjects) {

    parms[1] = thetaM[i, 1] * (weight[i] / 70)^0.75; # CL
    parms[2] = thetaM[i, 2] * (weight[i] / 70)^0.75; # Q
    parms[3] = thetaM[i, 3] * (weight[i] / 70); # V1
    parms[4] = thetaM[i, 4] * (weight[i] / 70); # V2
    parms[5] = kaHat; # ka
    parms[6] = thetaM[i, 5]; # mtt
    parms[7] = thetaM[i, 6]; # circ0
    parms[8] = gamma;
    parms[9] = thetaM[i, 7]; # alpha

    x[start[i]:end[i]] = generalOdeModel_rk45(twoCptNeutModelODE, nCmt,
                                              time[start[i]:end[i]],
                                              amt[start[i]:end[i]],
                                              rate[start[i]:end[i]],
                                              ii[start[i]:end[i]],
                                              evid[start[i]:end[i]],
                                              cmt[start[i]:end[i]],
                                              addl[start[i]:end[i]],
                                              ss[start[i]:end[i]],
                                              parms, biovar, tlag,
                                              1e-6, 1e-6, 1e6);

    cHat[start[i]:end[i]] = x[start[i]:end[i], 2] / parms[3];  ## divide by V1
    neutHat[start[i]:end[i]] = x[start[i]:end[i], 8] + parms[7];  ## Add baseline

  }

  cHatObs = cHat[iObsPK];
  neutHatObs = neutHat[iObsPD];

}
```

It pays off to construct informative priors, if available. The PD parameters are drug independent, so we can use information from the neutropenia literature. In this example, we choose to use weakly informative priors on the PK parameters and strongly informative priors on the PD parameters.

Since it takes a long time to run the model, we only use 100 iterations per chain, and study what we can learn from this less than optimal scenario. It is worth noting that Stan, because of its highly efficient MCMC sampler, still does a reasonable job estimating the posterior distribution.

2.2. Results. The MCMC history plots are not as convincing as in the previous examples, mostly because the number of iterations is small (100 versus 1000 in the previous example) (Figure 4.6). It does however look as though the chains are converging to a common distribution, and we see little auto-correlation (in particular, we expect that if we had run the model for 1000 iterations, we would obtain the desired "fuzzy caterpillar" look). The model fits the data, and the credible

interval reflect the noise in the data (Figure 4.8). The parameters estimation reflects the real value of the parameters (Table 4.2 and Figure 4.7).

TABLE 4.2. Summary of the MCMC simulations of the marginal posterior distributions of the model parameters for the Friberg-Karlsson model example.

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
CL	9.986	0.009	0.174	9.641	9.872	9.982	10.107	10.331	400.000	0.997
Q	14.633	0.055	1.106	12.505	13.992	14.623	15.296	16.948	400.000	0.996
V1	32.909	0.174	2.439	28.203	31.186	32.836	34.762	37.750	195.828	1.008
V2	106.631	0.311	6.226	95.234	102.269	106.403	111.000	118.533	400.000	0.999
ka	1.882	0.012	0.175	1.582	1.756	1.871	2.006	2.223	196.052	1.007
sigma	0.106	0.001	0.010	0.089	0.098	0.105	0.112	0.132	259.693	1.009
alpha	3.3e-04	1.4e-06	2.2e-05	2.9e-04	3.2e-04	3.3e-04	3.5e-04	3.8e-04	247	1.01
mtt	132.763	0.515	6.498	120.843	128.082	132.223	136.694	146.845	159.372	1.024
circ0	5.014	0.009	0.172	4.711	4.888	5.000	5.138	5.334	400.000	1.000
gamma	0.190	0.002	0.022	0.153	0.175	0.187	0.202	0.239	139.485	1.025
sigmaNeut	0.092	0.001	0.014	0.068	0.082	0.090	0.100	0.125	161.199	1.010

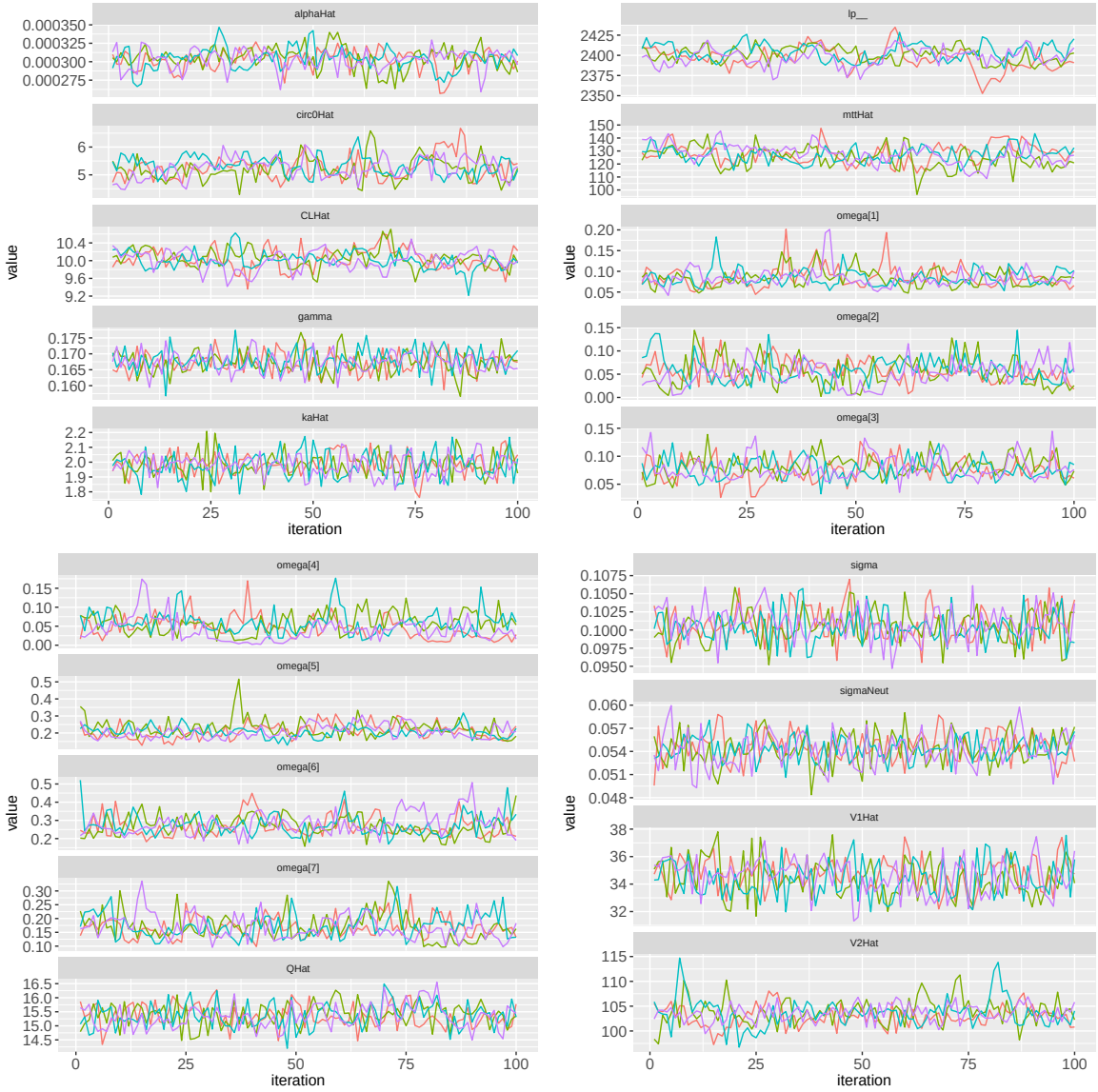

FIGURE 4.6. MCMC history plots for the parameters of a Friberg-Karlsson semi-mechanistic model (each color corresponds to a different chain) for example 3

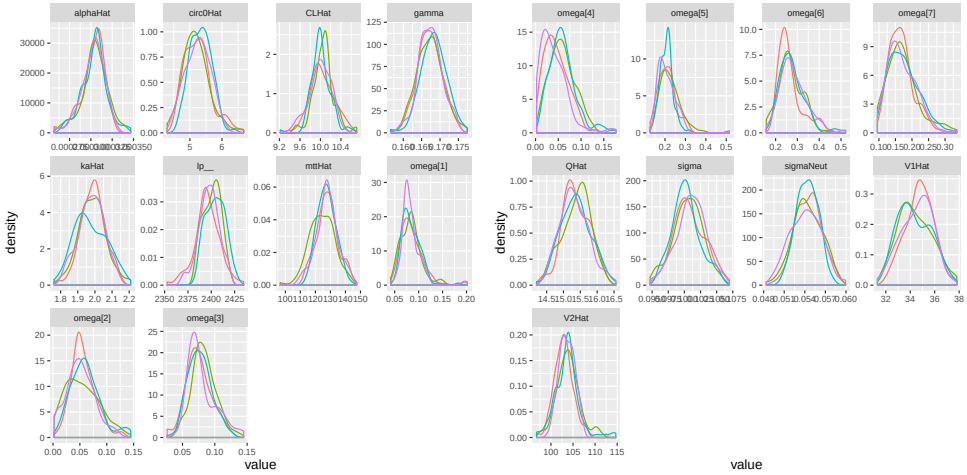

FIGURE 4.7. Posterior Marginal Densities of the Model Parameters of a Friberg-Karlsson semi-mechanistic model (each color corresponds to a different chain)

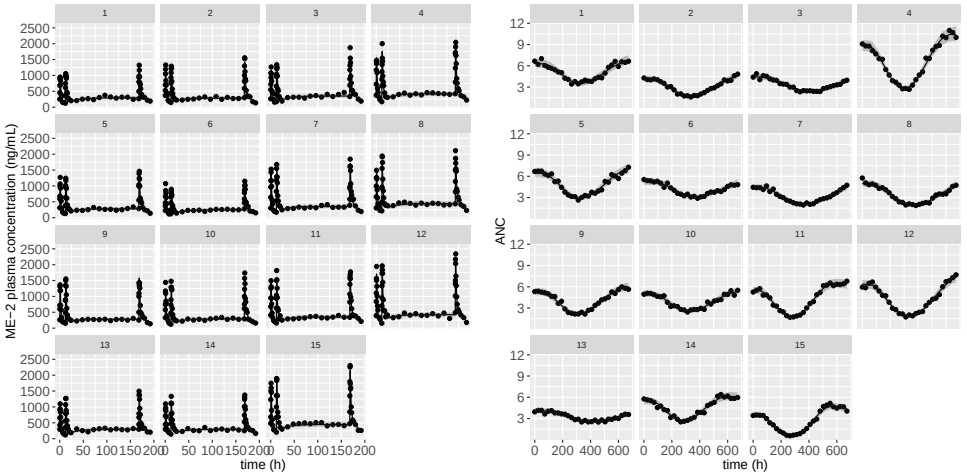

FIGURE 4.8. Predicted (posterior median and 90 % credible intervals) and observed plasma drug concentrations, and Neutrophil counts, for a Friberg-Karlsson semi-mechanistic model

Index

Bibliography

[1] Kyle T. Baron and Marc R. Gastonguay. Simulation from ode-based population pk/pd and systems pharmacology models in r with mrgsolve. *Journal of Pharmacokinetics and Pharmacodynamics*, 42(W-23):S84–S85, 2015.

[2] Michael Betancourt. A conceptual introduction to hamiltonian monte carlo. 2018. arXiv: 1701.02434.

[3] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A Probabilistic Programming Language. *Journal of Statistical software*, 76, 2017.

[4] Bob Carpenter, Matthew D. Hoffman, Marcus Brubaker, Daniel Lee, Peter Li, and Michael Betancourt. The Stan Math Library: Reverse-Mode Automatic Differentiation in C++. *arXiv:1509.07164 [cs]*, September 2015. arXiv: 1509.07164.

[5] Lena E. Friberg and Mats O. Karlsson. Mechanistic Models for Myelosuppression. *Investigational New Drugs*, 21(2):183–194, May 2003.

[6] Matthew D. Hoffman and Andrew Gelman. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, April 2014. arXiv: 1111.4246.

[7] Charles C Margossian and William R Gillespie. Gaining efficiency by combining analytical and numerical methods to solve odes: Implementation in stan and application to bayesian pk/pd. In *Journal of Pharmacokinetics and Pharmacodynamics*, volume 44, October 2017.

[8] Stan Development Team. *Stan Modeling Language Users Guide and Reference Manual*, 2017.