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 0.84, Stan version 2.17.1)

February 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.

1

Introduction

Stan is an open source probabilistic programing language designed primarily to do Bayesian data analysis [2]. Several of its features make it a powerful tool to specify and fit complex models. Notably, its language is extremely flexible and its No U-Turn Sampler (NUTS), an adaptative Hamiltonian Monte Carlo algorithm, has proven more efficient than commonly used Monte Carlo Markov Chains (MCMC) samplers for complex high dimensional problems [5]. 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 new mathematical tools, 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)!

Torsten is licensed under the BSD 3-clause license.

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.

1.1. 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:

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

- Recursive calculation of model predictions
  - 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 Stan 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:

- The RATE and TIME arguments must be fixed
- 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.

1.2. Installation

Installation files are available on GitHub

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

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 (v0.84) is compatible with Stan v2.17.1. Torsten is agnostic to which Stan interface you use. Here we provide command line and R interfaces.

First, download the project. The root path of the project is your `torsten_path`. Set the envionment variable `TORSTEN_PATH` to `torsten_path`. In bash this is

```
export TORSTEN_PATH=torsten_path
```

Command line version. There is no need to install command line version. To build a Stan model with `model_name` in `model_path` using command line, in `torsten_path/cmdstan`, do

```
make model_path/model_name
```

R version. The R version 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++11 support. In particular, R 3.3.0 and later is recommended as it contains toolchain based on gcc 4.9.3. On Windows platform, such a toolchain can be found in Rtools33 and later.

Please ensure CXXFLAGS in .R/Makevars constains flag -std=c++11.

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

Testing. To test after installation, run

```
./test-torsten.sh --unit --signature --model
```

1.3. Implementation details

- Stan version 2.17.1
- All functions are programmed in C++ and are compatible with the Stan math automatic differentiation library [3]
- One and two compartment models: hand-coded analytical solutions
- General linear compartment models with semi-analytical solutions using the built-in matrix exponential function
- General compartment models with numerical solutions 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.
- Mix compartment model: the PK forcing function is solved analytically and the forced ODE system is solved numerically.

1.4. 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 RATE, AMT, TIME, and the 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

1.5. Changelog

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.

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

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

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

- Added linCptModel (linear compartmental model) function

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

- Added check$_{\text{finite}}$ statements in pred$_1$ and pred$_2$ to reject metropolis proposal if initial conditions are not finite

2

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. It will be helpful to apply these functions to a simple example. We have uploaded code and data on

- https://github.com/metrumresearchgroup/example-models

We use the following example to demonstrate use of several functions in the Torsten library.

Two compartment model. 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
- PK: plasma concentrations of parent drug (c)
- PK measured at 0.083, 0.167, 0.25, 0.5, 0.75, 1, 1.5, 2, 4, 6, 8, 10 and 12 hours after 1st, 2nd, and 15th dose. In addition, the PK is measured every 12 hours throughout the trial.

The plasma concentration (c) are simulated according to the following equations:

$$\begin{aligned}
\log(c) &\sim N\left(\log(\widehat{c}), \sigma^2\right) \\
\widehat{c} &= f_{2cpt}\left(t, CL, Q, V_2, V_3, k_a\right) \\
(CL, Q, V_2, V_3, ka) &= \left(5 \text{ L/h}, 8 \text{ L/h}, 20 \text{ L}, 70 \text{ L}, 1.2 \text{ h}^{-1}\right) \\
\sigma^2 &= 0.01
\end{aligned}$$

and the drug concentration is given by $c = y_2/V_2$.

where the mass of drug in the central compartment (y_2) is obtained by solving the system of ordinary differential equations (ODEs):

$$\begin{aligned}
y_1' &= -k_a y_1 \\
y_2' &= k_a y_1 - \left(\frac{CL}{V_2} + \frac{Q}{V_2}\right) y_2 + \frac{Q}{V_3} y_3 \\
y_3' &= \frac{Q}{V_2} y_2 - \frac{Q}{V_3} y_3
\end{aligned}$$

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

2.1. One Compartment Model

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

Parameters in `theta` : CL, V_2, k_a.

The event arguments. `time`, `amt`, `rate`, `ii`, `evid`, `cmt`, `addl`, and `ss`, describe the event schedule of the clinical trial. All arrays have the same length, which corresponds to the number of events.

The model arguments, `theta` contains the ODE parameters, `biovar` the bioavailability fraction in each compartment, and `tlag` the lag time in each compartment. The model arguments may be either one or two dimensional arrays. If they are one dimensional arrays, the parameters are constant for all events. If they are two dimensional arrays then each row contains the parameters for the interval `[ time[i-1], time[i] ]`. The number of rows should equal the number of events.

Setting ka to 0 eliminates the first-order absorption.

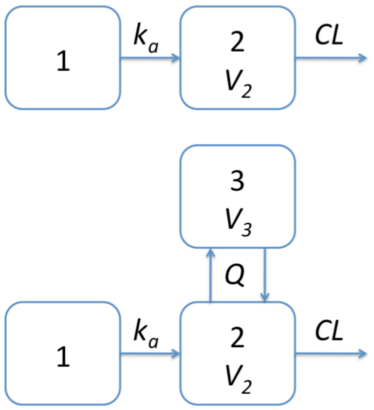

FIGURE 1. One and two compartment models with first order absorption implemented in Torsten.

2.2. Two Compartment Model

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

Parameters in `theta` : CL, Q, V_2, V_3, k_a

See section **??** for function description.

Code example below shows Stan example of using `TwoCptModel` to fit Two compartment model. Three MCMC chains of 2000 iterations were simulated. The first 1000 iteration of each chain were discarded. Thus 1000 MCMC samples per chain were used for the subsequent analyses.

```
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
```

The MCMC history plots(Figure 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 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 1 and Figure 4 are consistent with the values used for simulation.

Table 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

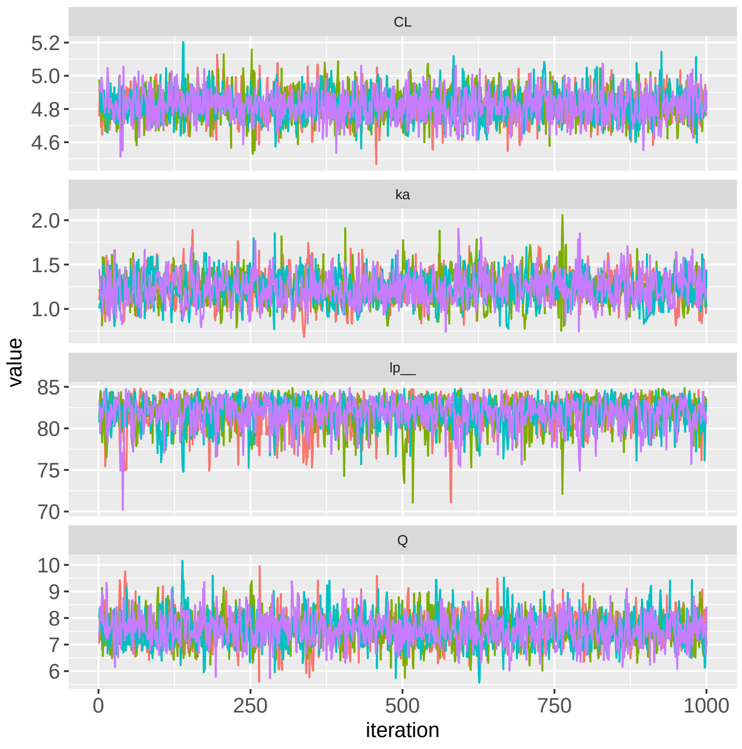

Figure 2. MCMC history plots for the parameters of a two compartment model with first order absorption (each color corresponds to a different chain)

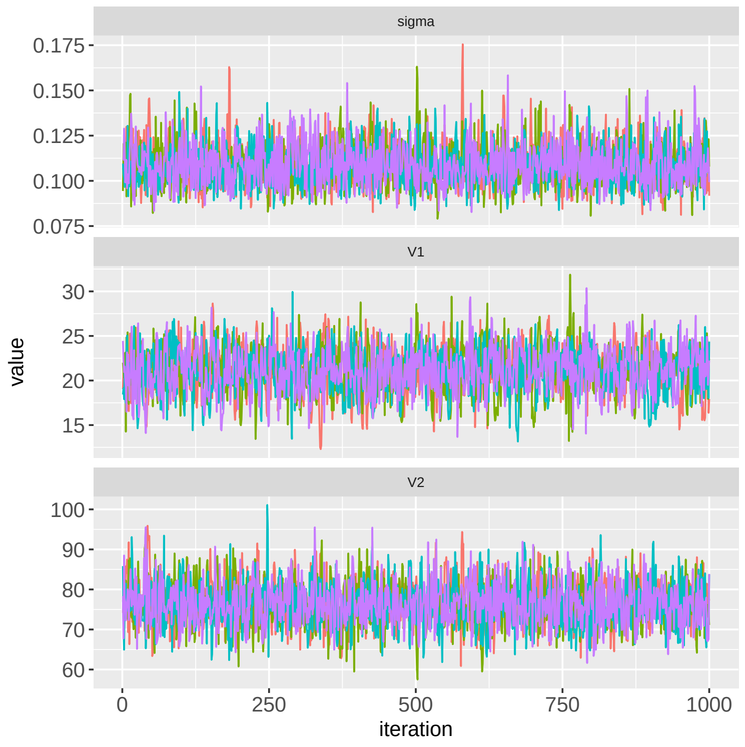

FIGURE 3. MCMC history plots for the parameters of a two compartment model with first order absorption (each color corresponds to a different chain)

2.3. General Linear ODE Model Function

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

A general linear ODE model refers to a model that may be described in terms of a system of first order linear differential equations with (piecewise) constant coefficients, i.e., a differential equation of the form:

$$y'(t) = Ky(t) \quad (1)$$

where K is a matrix. For example K for a two compartment model (equation (**??**)) with first order absorption is:

$$K = \begin{bmatrix} -k_a & 0 & 0 \\ k_a & -(k_{10}+k_{12}) & k_{21} \\ 0 & k_{12} & -k_{21} \end{bmatrix} \quad (2)$$

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

System parameters is in `K`, with `K` being the constant rate matrix is the same for all events or an array of constant rate matrices. The length of the array is the number of events and each element corresponds to the matrix at the interval `[ time[i-1], time[i] ]`. Note that `K` contains all the ODE parameters, so we no longer need `theta`.

The following Stan example illustrate the use of General linear model for fitting a two compartment model with first order absorption.

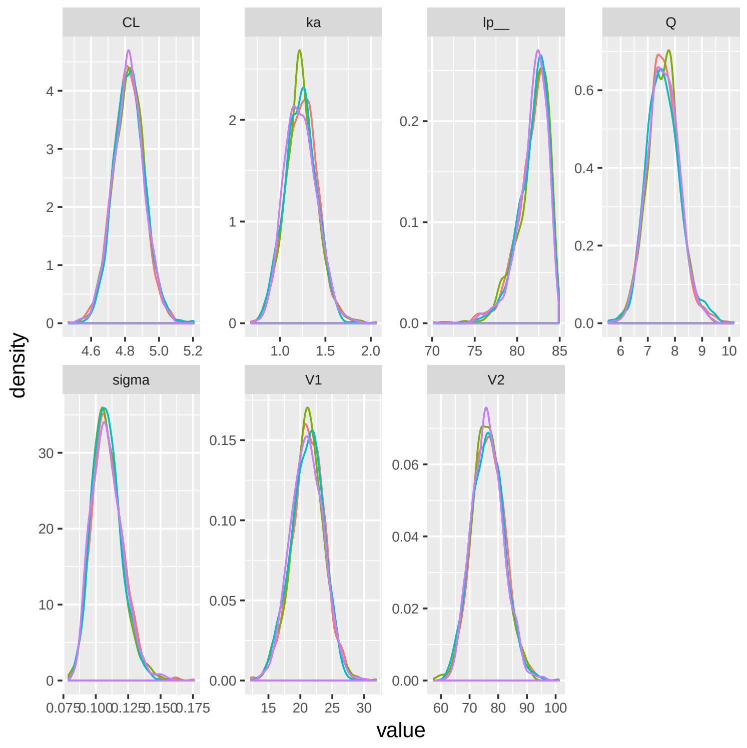

FIGURE 4. Posterior Marginal Densities of the Model Parameters of a two compartment model with first order absorption (each color corresponds to a different chain)

```
// 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];
```

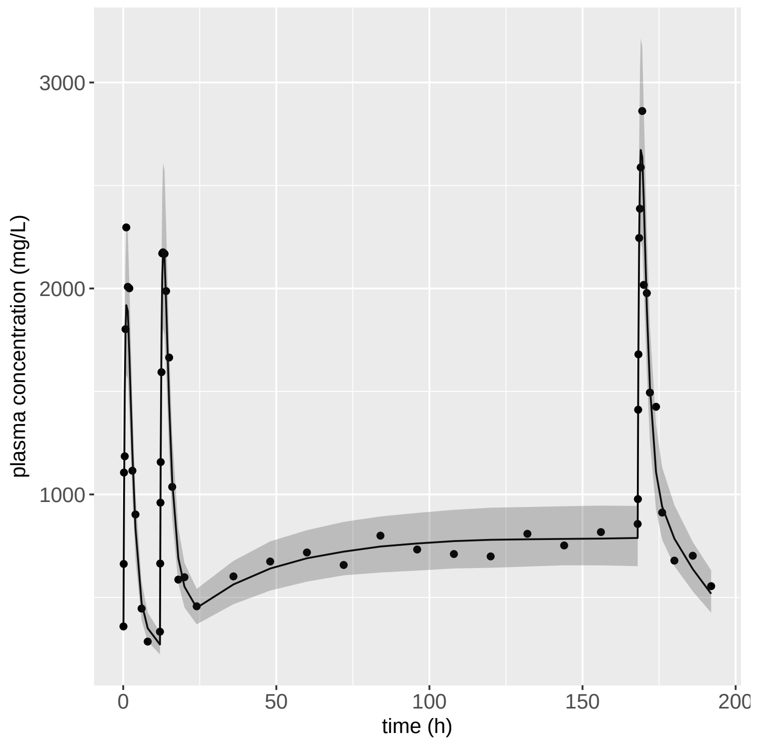

FIGURE 5. Predicted (posterior median and 90% credible intervals) and observed plasma drug concentrations of a two compartment model with first order absorption

```
  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);
```

2.4. General ODE Model Function

```
real[] model_name(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)
```

Torsten may be used to fit models described by a system of user-specified first-order ODEs, i.e., differential equations of the form:

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

In the case where the rate vector R is non-zero, this equation becomes:

$$y'(t) = f(t, y(t)) + R$$

`ODE_system` specifies $f(t, y(t))$, which the user defines inside the `functions` block (see section 19.2 of the Stan reference manual for details and Figure~**??** 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.

The options for `model_name` are:

- `generalOdeModel_rk45`
- `generalOdeModel_bdf`

They respectively call the built-in Runge-Kutta 4th/5th order (rk45) integrator, recommended for non-stiff ODEs, and the Backward Differentiation (BDF) integrator, recommended for stiff ODEs. Which value 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 [6].

A few notable restrictions apply to :

- `rate` and `time` cannot be passed as parameters.
- In the case of a multiple truncated infusion rate dosing regimen:
  - The bioavailability `biovar`) and the amount `amt`) cannot be passed as parameters.

These restrictions also apply to `mixOdeCpt_*` functions, discussed in the next section.

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

functions{

  // 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
```

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

```
  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
  }
}
```

2.5. Mixed ODE Model Function

```
real[] model_name(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)
```

In certain cases, an ODE system can be divided in two subsystems:

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

where y_1, y_2, f_1, and f_2 are vector-valued functions, and y_1' is independent of y_2. This structure arises in PK/PD models, where y_1 describes a forcing PK function and y_2 the PD effects. If y_1 has an analytical solution, we can construct a *mixed solver*, which analytically solves y_1 and numerically integrates y_2. This approach leads to an appreciable gain in computational efficiency. In the example of a Friberg-Karlsson semi-mechanistic model, we observe an average speedup of $\sim 47 \pm 18\%$ when using the mix solver in lieu of the numerical integrator. 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* and nOde 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 PKstates, 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 options for modelName are:

- mixOde1CptModel_rk45
- mixOde1CptModel_bdf

- `mixOde2CptModel_rk45`
- `mixOde2CptModel_bdf`

These four functions correspond to all the permutations we can obtain when using a forcing One or Two Compartment function, and the Runge-Kutta 4th/5th order (rk45) or Backward Differentiation (BDF) integration method. 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 `generalOdeModel_*` also apply to `mixOdeCptModel_*`.

We cannot apply the mixed solver to the Two Compartment example we have been using so far. Instead, we will consider the model which motivated the implementation of the method in the first place.

2.6. Friberg-Karlsson Semi-Mechanistic Model

In this second example, we add to our two Compartment model a PD effect, described by a system of nonlinear ODEs [4].

Friberg-Karlsson 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~**??**). 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.

$$\log(ANC_i) \sim N(\log(Circ), \sigma^2_{ANC}) \tag{3}$$
$$Circ = f_{FK}(MTT, Circ_0, \alpha, \gamma, c) \tag{4}$$
$$(MTT, Circ_0, \alpha, \gamma, ktr) = (125, 5.0, 3 \times 10^{-4}, 0.17) \tag{5}$$
$$\sigma^2_{ANC} = 0.001 \tag{6}$$

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:

$$\begin{aligned}
y'_{\text{prol}} &= k_{\text{prol}} y_{\text{prol}} (1 - E_{\text{drug}}) \left(\frac{Circ_0}{y_{\text{circ}}}\right)^{\gamma} - k_{\text{tr}} y_{\text{prol}} \\
y'_{\text{trans1}} &= k_{\text{tr}} y_{\text{prol}} - k_{\text{tr}} y_{\text{trans1}} \\
y'_{\text{trans2}} &= k_{\text{tr}} y_{\text{trans1}} - k_{\text{tr}} y_{\text{trans2}} \\
y'_{\text{trans3}} &= k_{\text{tr}} y_{\text{trans2}} - k_{\text{tr}} y_{\text{trans3}} \\
y'_{\text{circ}} &= k_{\text{tr}} y_{\text{trans3}} - k_{\text{tr}} y_{\text{circ}}
\end{aligned} \tag{7}$$

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

The ODEs specifying the Two Compartment Model (equation~**??**) do not depend on the PD ODEs (equation~7) and can be solved analytically by Torsten. 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 (Figures **??** and **??**).

```
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];
    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);
}
```

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. *J Pharmacokinet Pharmacodyn*, 42(W-23):S84–S85, 2015.

[2] 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.

[3] 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.

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

[5] Matthew D. Hoffman and Andrew Gelman. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *arXiv:1111.4246 [cs, stat]*, November 2011. arXiv: 1111.4246.

[6] Stan Development Team. Stan modeling language users guide and reference manual. 2017.