# Torsten: A PKPD Model Library for Stan

*Charles Margossian, Bill Gillespie, and Metrum Research Group, LLC*

*Updated: August 3rd 2016*

## 1 Introduction

Metrum Research Group, LLC has developed a prototype PKPD model library for Stan. The current version includes:

- One compartment model with first order absortpion
- Two compartment model with elimination from and first order absorption into the central compartment
- General compartmental model described by a system of first order Ordinary Differential Equations (ODEs)

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

- Recursive calculation of model predictions, which permits piecewise constant covariate values
- Bolus or constant rate inputs into any compartment
- Handling of single dose, multiple doses and steady-state dosing histories
- Implemented NMTRAN data items: TIME, EVID, CMT, AMT, RATE, ADDL, II, SS

The library provides Stan language functions that calculate amounts in each compartment.

### Implementation Details

- Stan (http://mc-stan.org/)
- An extensible object-oriented programing approach is used to facilitate development of additional models
- The core functions are programmed in C++
- One and two compartment models: analytical calculations programmed in C++
- General compartment models with numerical solutions to ODE using built-in ODE integrators in Stan (Runge-Kutta 4th/5th and CVODES). NOTE: currently, general compartment models do not handle steady state approximations

**Warning:** The current version of Torsten is a *prototype*. It is being released for review and comment. It is still being developed and should not be used for critical applications without further testing or cross-checking by comparison with other methods.

### Development

This project is open-source and forsters collaboration. We have recieved extensive help and advice from researchers at various institutions, notably the Stan Group. See https://groups.google.com/forum/#!forum/stan-dev.

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

### Install Stan and Torsten

Examples and installation files are available on GitHub: https://github.com/charlesm93/example-models/tree/feature/issue-70-PKPDexamples-torsten/PKPD/torsten

Torsten has been tested with cmdStan-2.11, the command line interface of Stan. I recommend using cmdStan-dev and downloading Torsten from gitHub. The easiest way to do so is to run the bash file `setupTorsten.sh` posted on example-models (see link above). You can also do this step by step on terminal.

Download cmdstan-dev:

```
git clone https://github.com/stan-dev/cmdstan.git
```

Add the version of Stan that exposes the Torsten functions to the language, and switch to the branch `feature/issue-1953-exposing-torsten-pharmacometrics`:

```
cd cmdstan
git clone https://github.com/charlesm93/stan.git
cd stan
git checkout feature/issue-1953-exposing-torsten-pharmacometrics
```

Next, install the version of stan-math that contains the Torsten library, and switch to the branch `feature/issue-314-torsten-pharmacometrics`:

```
cd lib
rm -r stan_math
git clone https://github.com/charlesm93/math.git
cd math
git checkout feature/issue-314-torsten-pharmacometrics
cd ..
mv math stan_math
```

## 2 Using Torsten

### Linear One and Two compartment model

The one and two compartment model functions have the form:

```
<modelName>(theta, time, amt, rate, ii, evid, cmt, addl, ss)
```

where `time`, `amt`, `rate`, `ii` are arrays of real and `evid`, `cmt`, `addl`, and `ss` arrays of integers. All arrays have the same length, which corresponds to the number of events. `theta`, also known as the *parameter matrix*, is an array of vectors. If the parameters are constant for all events, `theta` should be an array of length 1, else it should have the same length as `time`, which amounts to having one vector of parameters per event. In the latter case, the *ith* row contains a vector of parameters for the time interval [`time[i-1]`, `time[i]`].

The options for *modelName* are:

- PKModelOneCpt
- PKModelTwoCpt

which respectively correspond to the one and two compartment model with first order absorption (see figure 1).

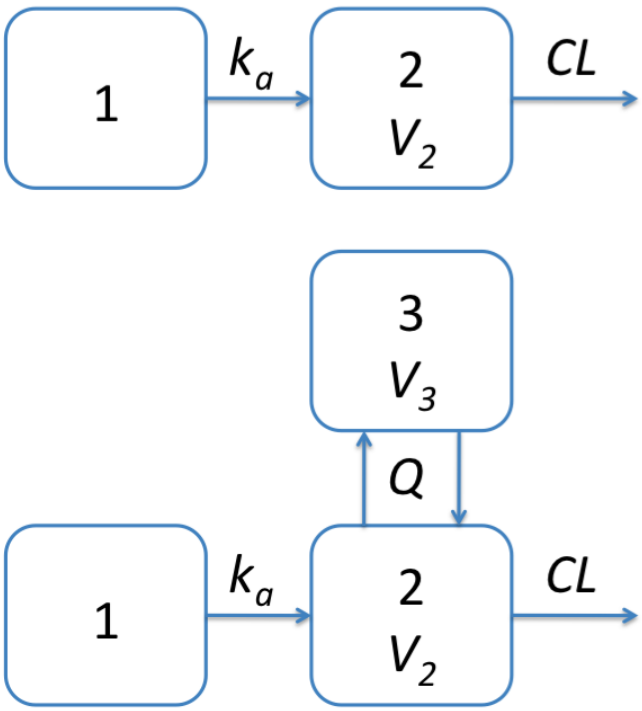


Figure 1: One and Two Compartment Model with First Order Absorption

## General compartment model

The general compartment model functions have the form:

```
<modelName>(ODE_System, nCmt,
            theta, time, amt, rate, ii, evid, cmt, addl, ss,
            rel_tol, abs_tol, max_step)
```

where `ODE_System` is a System of first order ODEs defined in the function block of Stan (see section 19.2 of the Stan reference manual for instructions on how to code such a system) and `nCmt` the number of compartments in the model. `rel_tol`, `abs_tol`, and `max_step` are the tuning parameters of the ODE integrator, respectively the relative tolerance, the absolute tolerance, and the maximum number of steps.

The options for *modelName* are:

- generalCptModel_bdf
- generalCptModel_rk45

These respectively call the built-in CVODES and Runge-Kutta 4th/5th ODE integrators.

## Example 1: Two Compartment Model

To illustrate use of the model library, let's start with a fairly simple example. We modeled drug absorption in a single patient and simulated plasma drug concentrations:

- Single Dose: 5000 mg
- PK: plasma concentrations of parent drug
- PK measured at 0, 0.25, 0.50, 0.75, 1, 1.25, 1.50, 1.75, 2, 4, 6, 8, 10, 12, 14, 16, 18, and 20 hours after

The plasma drug concentration ($c$) is simulated according to the following equations:

$$log(c) \sim N(log(\hat{c}), \sigma)$$
$$\hat{c} = f_{2cpt}(t, CL, Q, V_1, V_2, ka)$$
$$(CL, Q, V_1, V_2, ka) = (5 \text{ L/h}, 8 \text{ L/h}, 20 \text{ L}, 70 \text{ L}, 1.2 \text{ h}^{-1}), \sigma = 0.05$$

The simulations are generated using the R package *mrgsolve*. Four MCMC chains of 2000 iterations were simulated. The first 1000 iteration of each chain were discarded. Thus 1000 MCMC samples were used for the subsequent analyses. To fit this model, we used `TwoCptModelExample.stan` and the data and initial estimates simulated by `TwoCptModelSimulation.R`, respectively saved in `TwoCptModelExample.data.R` and `TwoCptModelExample.init.R`.

This example uses the function `PKModelTwoCpt`, which analytically computes the solution for a two compartment model. Alternatively, `GenTwoCptModelExample.stan` calls `genCptModel_bdf` and makes a numerical approximation of the solution. Since we are looking at relatively simple model and data, the fitting only takes a few seconds in both cases.

Additionally, we coded some priors in the model block:

```
CL ~ lognormal(log(10), 0.25);
Q ~ lognormal(log(15), 0.5);
V1 ~ lognormal(log(35), 0.25);
V2 ~ lognormal(log(105), 0.5);
ka ~ lognormal(log(2.5), 1);
sigma ~ cauchy(0, 1);
```

These priors are not essential to fit this particular model, but we included them to make the example more complete.

**Result**

The MCMC history plots (Figure 2) and the posterior marginal densities of the model parameters (figure 3) suggest that the 4 chains have converged to a common distribution for all of the key model parameters. The predicted data agrees with the simulated data (not surprisingly so since the fitted model is identical to the one used to simulate data; figure 4). Similarly the parameter estimates summarized in table 1 are consistent with the values used for the 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 | 5.21 | 0.01 | 0.25 | 4.65 | 5.06 | 5.22 | 5.37 | 5.65 | 1460.46 | 1.00 |
| Q | 7.93 | 0.01 | 0.48 | 6.97 | 7.62 | 7.93 | 8.23 | 8.89 | 1737.66 | 1.00 |
| V1 | 20.13 | 0.06 | 2.20 | 15.99 | 18.67 | 20.13 | 21.55 | 24.55 | 1226.46 | 1.00 |
| V2 | 63.72 | 0.19 | 6.98 | 51.92 | 59.20 | 62.96 | 67.39 | 79.63 | 1337.96 | 1.00 |
| ka | 1.24 | 0.00 | 0.17 | 0.95 | 1.13 | 1.23 | 1.34 | 1.58 | 1299.38 | 1.00 |

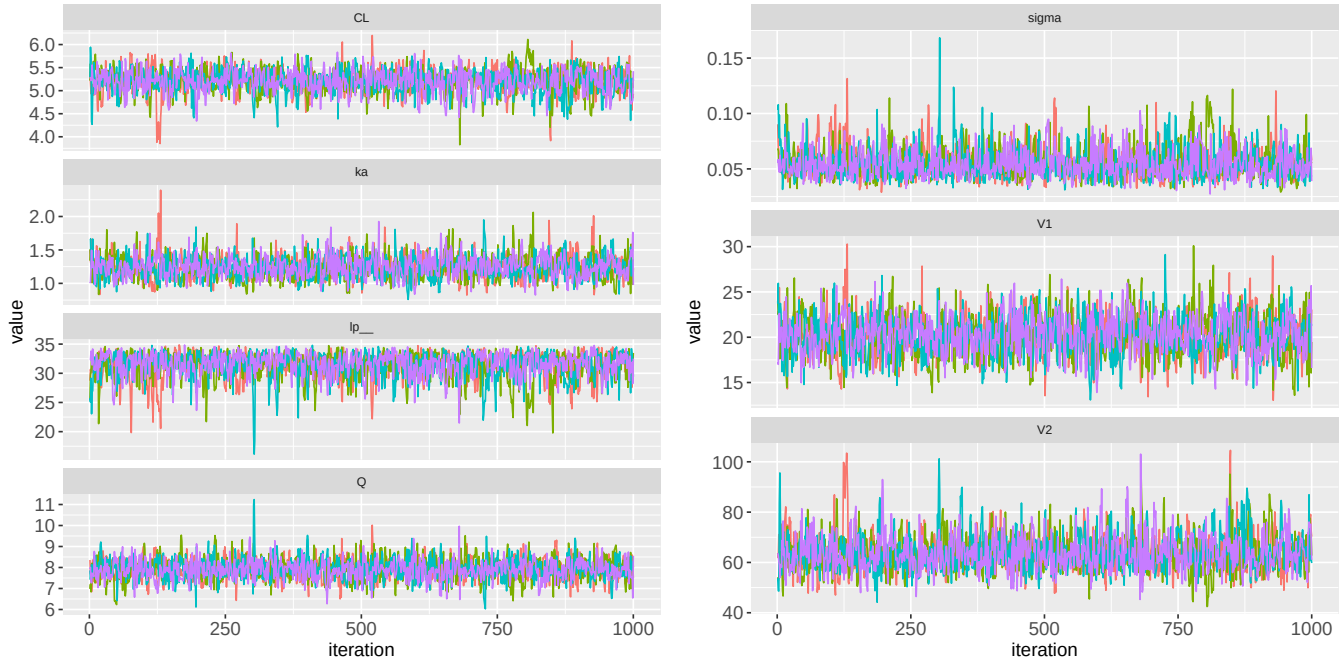


Figure 2: MCMC history plots for the parameters of a two compartment model with first order absorption (each color corresponds to a different chain)

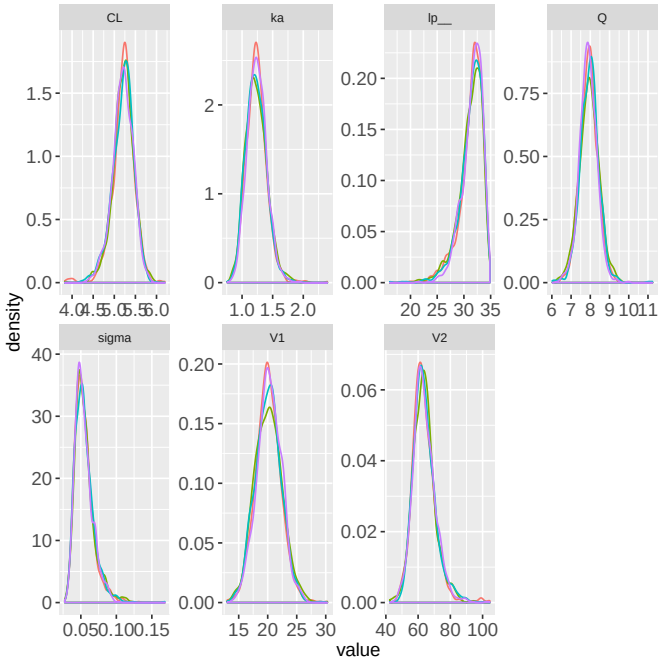


Figure 3: Posterior Marginal Densities of the Model Parameters (each color corresponds to a different chain)

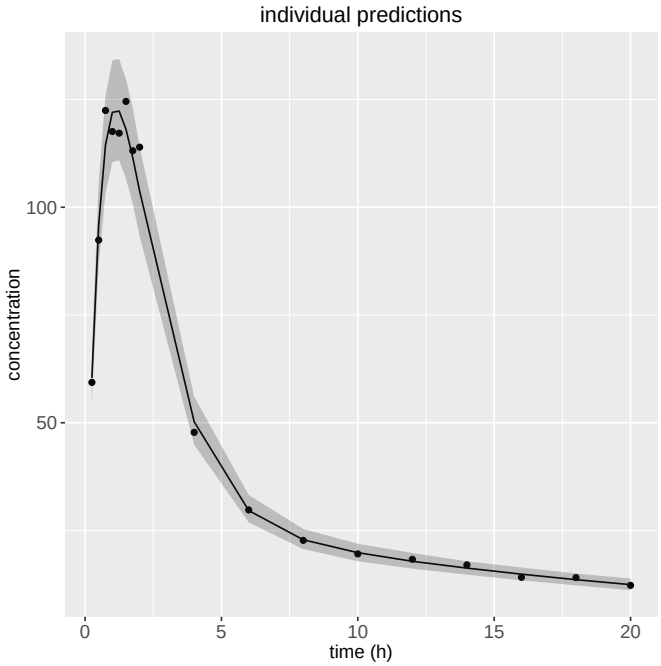


Figure 4: Predicted (posterior median and 90 % credible intervals) and observed plasma drug concentrations

## Example 2: Population PK Model

The above example can be extended to a population model, in which we administer the drug to multiple patients, and where the PK parameters vary from one individual to the other:

$$log(c_i) \sim N(log(\hat{c_i}), \sigma)$$

$$\hat{c_i} = f_{2cpt}(t_i, CL_i, Q_i, V_{1i}, V_{2i}, ka_i)$$

$$log(CL_i, Q_i, V_{1i}, V_{2i}, k_i) \sim N(log(\hat{CL}, \hat{Q}, \hat{V_1}, \hat{V_2}, \hat{ka}, \Omega)$$

$$(\hat{CL}, \hat{Q}, \hat{V_1}, \hat{V_2}, \hat{ka}) = (5 \text{ L/h}, 8 \text{ L/h}, 20 \text{ L}, 70 \text{ L}, 1.2 \text{ h}^{-1}), \sigma = 0.05$$

$$\Omega = \begin{matrix} 0.05^2 & 0 & 0 & 0 \\ 0 & 0.05^2 & 0 & 0 \\ 0 & 0 & 0.05^2 & 0 \\ 0 & 0 & 0 & 0.05^2 \end{matrix}$$

This model can be fitted using `TwoCptModelPopulation.stan` and the corresponding `.data.R` and `.init.R` files.

## Example 3: Population PK of ME-2

This example was presented at the *Getting Started with Bayesian PKPD Modeling Using Stan* workshop at the PAGE 2016 conference. Relevant pages from the workshop slides that describe the model (see `MetrumBayesianStanPAGE_example3.pdf`) are available on gitHub. This model is more elaborate, and has the merit of demonstrating how Stan and Torsten can be used to tackle more difficult problems. Just a heads up: it took me 7.5 hours to run the model with 2000 MCMC iterations, but the result was rather good (MCMC chains converge, and data predicted by fitted model agrees with real data).

### Further Examples

`TwoCptModelSimulation.R` and `TwoCptModelSimulationPopulation.R` can be used as templates to simulate PKPD data. Users are welcomed to experiment with different parameter values and different events (such as dosing at regular intervals or using steady state approximations). Examples on how to simulate PKPD data sets with *mrgsolve* can be found on https://github.com/metrumresearchgroup/mrgsolve/wiki/gallery. Changing the model itself is also a possibility, though it may result in model misspecification.

## 3 Under the Hood Design

At the end of the day, Torsten is four functions that can be called from Stan:

- PKModelOneCpt()
- PKModelTwoCpt()
- generalCptModel_rk45()
- generalCptModel_bdf()

We here discuss the "under the hood" design that supports these functions. This design is based on that of BUGSModelLibrary, a prototype PKPD model library for WinBUGS 1.4.3 (https://bitbucket.org/metrumrg/bugsmodellibrary/wiki/Home). From now on, we will refer to the four functions listed above as Torsten functions.

### Computing Amounts in an ODE-based Compartment Model

Given an event schedule and a compartment model, a Torsten function sequentially calculates the amount in each compartment of the model at each event. A PKPD compartment model is based on an ODE system, that describes how the amounts in each compartment change over time. For example, the following system describes the evolution of drug amounts in a one compartment model with first-order absorption from the gut:

$$\frac{dGUT}{dt} = -kaGUT$$
$$\frac{dCENT}{dt} = kaGUT - \frac{CL}{V_1}CENT$$

Our goal is to solve such an ODE system at each event. The initial conditions are specified by the amounts at the previous event, and the ODE is integrated from $t_{previous}$ to $t_{current}$.

All four functions call the C++ function `pred`, which does three things:

1. Augment the Event Schedule to include all the Events that will change the system
2. Calculate the amounts in each compartment at each of the events in the augmented schedule
3. Return the amounts at each event of the original schedule

The Event schedule depends on the user's input (TIME, EVID, CMT, AMT, RATE, ADDL, II, SS). The event schedule may need to be augmented if, for example, an event specifies a patient recieves multiple doses at a regular time interval. For example:

```
TIME = 0, EVID = 1, CMT = 1, AMT = 1500, RATE = 0, ADDL = 4, II = 10, SS = 0
```

specifes that a time 0 (TIME=0), a patient recieves a 1500 mg (AMT=1500) drug dose (EVID=1) in the gut (CMT=1), and will recieve an additional dose every 10 hours (II=10) until the patient has recieved a total of five doses (ADDL=4, being the number of additional doses, + 1, the original dose). Such an event really corresponds to 5 dosing events, and the event schedule will be appropriately augmented.

This is what the first half of `pred` does. To do so, it relies on the C++ classes (defined in Torsten): Events, Rates, and ModelParameters, which respectively correspond to the Event Schedule, the Rates at each time at which an event occurs, and the Model Parameters at each event.

In a second time, `pred` computes the amount at each event, by sequentially solving the ODE system the model is based on. To do so, it calls the function `pred1` (prediction for one event) or `predSS` (prediction for one event under the steady state approximation). `pred1` and `predSS` differ from one Torsten function to the other, and are functors. For example, under `PKModelOneCpt`, `pred1` analytically computes the solution for a one compartment model, while `generalCptModel_rk45` uses `pred1` to numerically solves an ODE system specified by the user in Stan.

Under this structural scheme, a Torsten function performs a very simple set of actions:

1. Consistent with Stan practices, check the validity of the arguments and of the parameter values
2. Construct the PKModel object, which contains basic information about the model, such as the number of compartments
3. Construct the pred1 and predSS functions
4. Call pred

The sample code below is from `PKModelOneCpt`:

```
template <typename T0, typename T1, typename T2, typename T3, typename T4>
Matrix <typename promote_args<T0, T1, T2, T3, T4>::type, Dynamic, Dynamic>
PKModelOneCpt(const vector< Matrix<T0, Dynamic, 1> >& pMatrix,
              const vector<T1>& time,
              const vector<T2>& amt,
              const vector<T3>& rate,
              const vector<T4>& ii,
              const vector<int>& evid,
              const vector<int>& cmt,
              const vector<int>& addl,
              const vector<int>& ss)
{
  PKModel model("OneCptModel");
  static const char* function("PKModelOneCpt");
  Matrix <typename promote_args<T0, T1, T2, T3, T4>::type, Dynamic, Dynamic> pred;

  pmetricsCheck(pMatrix, time, amt, rate, ii, evid, cmt, addl, ss, function, model);
  for(int i=0; i<pMatrix.size(); i++) {
      stan::math::check_positive_finite(function, "PK parameter CL", pMatrix[i](0,0));
      stan::math::check_positive_finite(function, "PK parameter V2", pMatrix[i](1,0));
      stan::math::check_positive_finite(function, "PK parameter ka", pMatrix[i](2,0));
  }

  //Construct Pred functions for the model.
  Pred1_structure new_Pred1("OneCptModel");
  PredSS_structure new_PredSS("OneCptModel");
  Pred1 = new_Pred1;
```

```
  PredSS = new_PredSS;

  pred = Pred(pMatrix, time, amt, rate, ii, evid, cmt, addl, ss, model, dummy_ode());

  return pred;
}
```

## Implementing Torsten in Stan

### Modifications in Stan-math

All four Torsten functions are located under the `torsten` directory, under `stan/math`, along side the other directories that contain C++ functions (`rev`, `fwd`, etc.). We modified `rev/mat.hpp` to include the `torsten/mat.hpp` header file which in turn includes the Torsten files Hpp files supporting Torsten functions, such as `pred.hpp`, are located under the `PKModel` directory.

The code can be found on GitHub: https://github.com/charlesm93/math/tree/feature/issue-314-torsten-pharmacometrics/stan/math/torsten

### Modifications in Stan

Furhter modifications are done in stan to expose the Torsten functions to the Stan language. Notably, we edited `function_signatures.h` for `PKModelOneCpt` and `PKModelTwoCpt`. The general compartment functions are higher-order functions, in that they take another function as one of their arguments. They were exposed by directly modifying the grammar files, following very close the example of `integrate_ode_rk45` and `integrate_ode_bdf`.

The code can be found on GitHub: https://github.com/charlesm93/stan/tree/feature/issue-1953-exposing-torsten-pharmacome