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

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

Torsten: A Prototype Model Library for Bayesian PKPD Modeling in Stan

User Manual: Version 0.81

Charles Margossian and Bill Gillespie

Development

This project is open-source. We have received extensive help and advice from Stan's development team, as well as insightful feedbacks from the broader Stan community. We are active contributors to Stan's core language, and our focus is on tools for differential equations based models, with the goal to support applications in pharmacometrics. Metrum Research Group's collaboration with Columbia University for the development of tools for *Fast and Flexible Differential Equation Model Fitting with Application to Pharmacometrics* is supported by the Small Business Technology Transfer grant from the Office of Naval Research.

1. Introduction

Metrum Research Group has developed a prototype Pharmacokinetic/Pharmacodynamic (PKPD) model library for use in Stan 2.12. The current version includes:

- Specific linear compartmental models:
  - One compartment model with first order absorption
  - Two compartment model with elimination from and first order absorption into central compartment
- General linear compartmental model described by a matrix exponential
- General compartmental model described by a system of first order Ordinary Differential Equations (ODE's)

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

- 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 for specific and general linear models
- Implemented NMTRAN data items include: TIME, EVID, CMT, AMT, RATE, ADDL, II, SS

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

Implementation details.

- Stan (http://mc-stan.org/)
- All functions are programmed in C++ and are compatible with the Stan-math autodiff library
- All functions can be called directly in a Stan file in a manner identical to other built-in functions
- One and two compartment models: coded analytical caclulations
- General linear compartment models with semi-analytical solutions using built-in (still in development) Matrix Exponential function (Pade approximation coupled with scaling and squaring)
- General compartment models with numerical solutions to ODE's using built-in ODE integrators in Stan (Runge-Kutta 4th/5th and CVODES), with adjustable tuning parameters

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.

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

- A system to easily share packages of Stan functions (written in C++ or in the Stan language)
- Extend formal tests
  - C++ Google unit tests
  - Comparison with simulations from the R package *mrgsolve*
- Steady state calculation for the General compartment model
  - This will require the developer of a solver for nonlinear algebraic equations (aka root solver)
- Allow user to provide time-varying constant rate matrix in `linCptModel`

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

- Optimize Matrix exponential functions
  - Function for action of Matrix Exponential on a vector
  - Coded gradients
  - Special algorithm for matrices with special properties
- Google C++ unit tests
- R tests comparing simulations with the R package *mrgsolve*
- Conduct beta testing

2. Install Stan and Torsten

Installation files are available on GitHub: `https://github.com/charlesm93/example-models/tree/feature/issue-70-PKPDexamples-torsten/PKPD/torsten`
Torsten uses a development version of Stan, that follows the 2.12 release, in order to implement the matrix exponential function required for the general linear compartment model. The easiest way to install Stan with the CmdStan interface is to run the bash file `setupTorsten.sh`. CmdStan-dev is set to the last version Torsten was tested on.

Download CmdStan:

```
git clone https://github.com/stan-dev/cmdstan.git
cd cmdstan
git reset --hard 53b4041
```

Download Stan with Torsten:

```
git clone https://github.com/charlesm93/stan.git
```

Download Stan_math with Torsten:

```
cd stan
cd lib
rm -r stan_math git clone https://github.com/charlesm93/math.git
mv math stan_math
```

3. Using Torsten

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.

Example 1: Two Compartment Model. We modeled drug absorption in a single patient and simulated plasma drug concentrations:

- Single Dose: 5000 mg
- PK: plasma concentration of parent drug (c)
- 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 dosing

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

$$\begin{aligned} \log(c) &\sim N\left(\log(\widehat{c}), \sigma^2\right) \\ \widehat{c} &= f_{2cpt}(t, CL, Q, V_2, V_3, k_a) \\ (CL, Q, V_2, V_3, ka) &= \left(5 \text{ L/h}, 8 \text{ L/h}, 20 \text{ L}, 70 \text{ L}, 1.2 \text{ h}^{-1}\right) \\ \sigma &= 0.05 \end{aligned}$$

The data are generated using the R package *mrgsolve*, see `TwoCptModelSimulation.R`. We show the results obtained when using the function `PKModelTwoCpt`, which computes solutions to the ODE's analytically.

Linear One and Two Compartment Model. The one and two compartment model functions have the form:

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

where `time, amt, rate, ii` are arrays of real and `evid, cmt, addl, 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 its length should be the number of events. In the latter case, the *ith* row contains a vector parameters for the time interval [`time[i-1], time[i]`].

The options for *model name* are:

- PKModelOneCpt
- PKModelTwoCpt

which respectively correspond to the one and two compartment model with first order absorption (see figure 1). A vector in `thetha` is expected to contain parameters CL, $V2$, and ka for the one compartment case, and CL, Q, $V2$, $V3$, and ka for the two compartments case, in this order, followed by the bioavailability fraction of each compartment (non-effective if set to 1) and the lag time in each compartment (non-effective if set to 0). Note that setting ka to 0 eliminates the fist-order absoprtion.

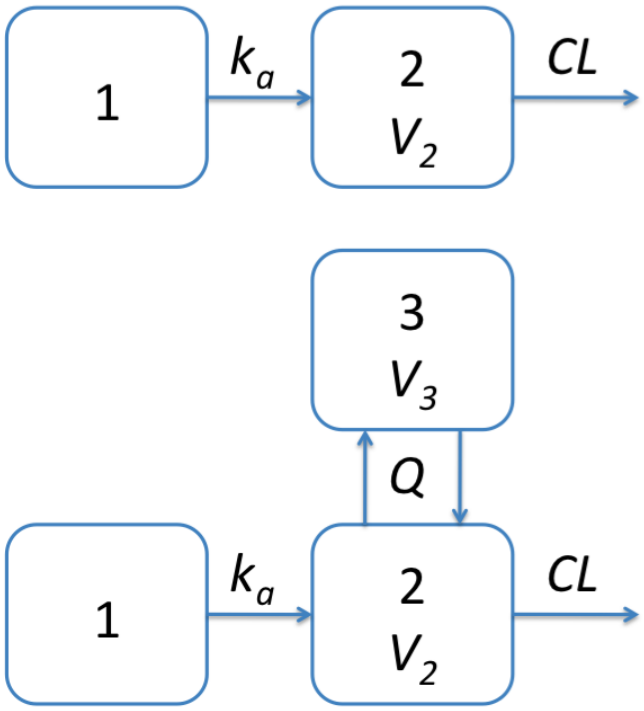

FIGURE 1. One and two compartment models with first order absorption implemented in Torsten.

`PKModelTwoCpt` can be used to fit example 1, see `TwoCptModelExample.stan`. 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.

Result. The MCMC history plots (Figure 3) suggest that the 4 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 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	5.19	0.01	0.25	4.60	5.05	5.21	5.35	5.62	971.04	1.00
Q	7.90	0.01	0.47	6.97	7.58	7.93	8.21	8.79	1662.02	1.00
V2	20.25	0.08	2.27	15.71	18.74	20.23	21.76	24.67	896.67	1.00
V3	64.10	0.22	6.98	52.60	59.48	63.38	67.74	80.65	1032.66	1.00
ka	1.25	0.00	0.17	0.94	1.13	1.24	1.35	1.62	940.99	1.00

General Linear Compartment Model. A general linear compartment 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:

$$x'(t) = Kx(t)$$

FIGURE 2. Stan language for fitting a two compartment model using the `PKModelTwoCpt` function (abstract)

```
data {
  int<lower = 1> nt; # number of events
  int<lower = 1> nObs; # number of observation
  int<lower = 1> iObs[nObs]; # index of observation
  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)
}
                              ⋮
parameters {
  real<lower = 0> CL;
  real<lower = 0> Q;
  real<lower = 0> V2;
  real<lower = 0> V3;
  real<lower = 0> ka;
  real<lower = 0> sigma;
}

transformed parameters {
                              ⋮
  theta[1][1] = CL;
  theta[1][2] = Q;
  theta[1][3] = V2;
  theta[1][4] = V3;
  theta[1][5] = ka;
  theta[1][6] = 1; # F1
  theta[1][7] = 1; # F2
  theta[1][8] = 1; # F3
  theta[1][9] = 0; # tlag1
  theta[1][10] = 0; # tlag2
  theta[1][11] = 0; # tlag3

  x = PKModelTwoCpt(theta, time, amt, rate, ii, evid, cmt, addl, ss);

  cHat = col(x, 2) ./ V2; # get concentration in the central compartment

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

model {
  # priors
  CL ~ lognormal(log(10), 0.25);
  Q ~ lognormal(log(15), 0.5);
  V2 ~ lognormal(log(35), 0.25);
  V3 ~ lognormal(log(105), 0.5);
  ka ~ lognormal(log(2.5), 1);
  sigma ~ cauchy(0, 1);

  logCObs ~ normal(log(cHatObs), sigma);
}
```

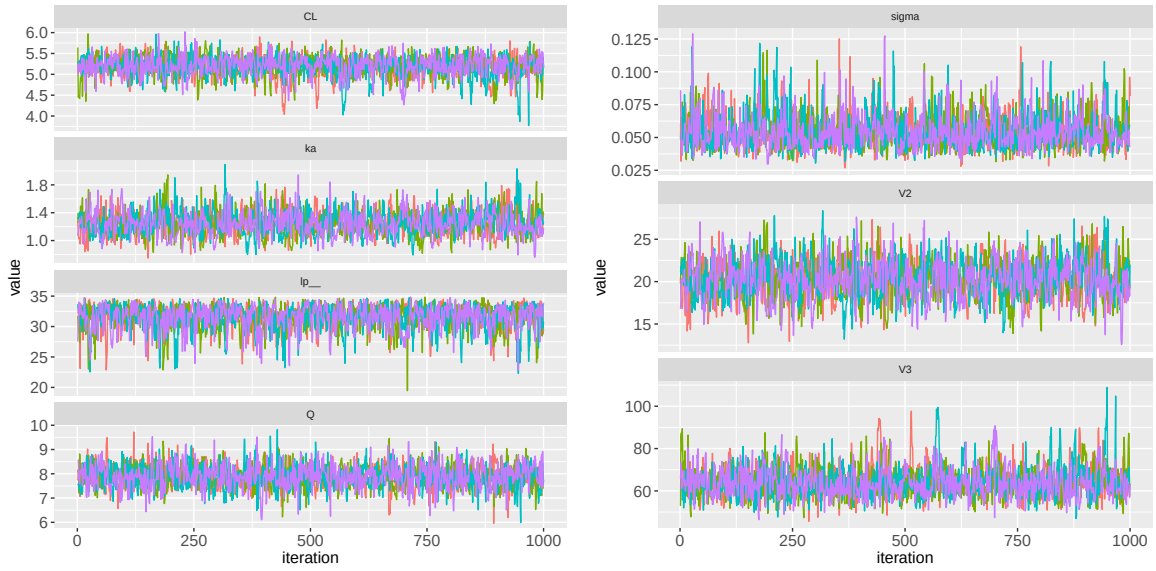

FIGURE 3. MCMC history plots for the parameters of a two compartment model with first order absorption (each color corresponds to a different chain)

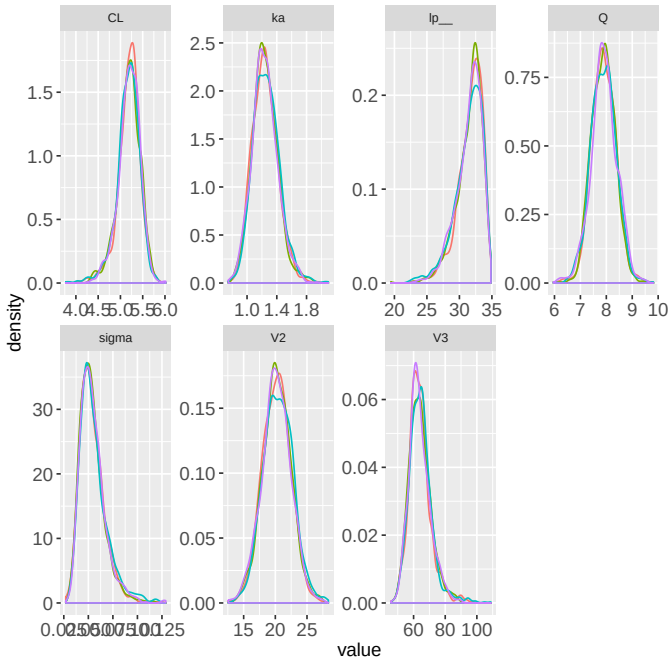

FIGURE 4. Posterior Marginal Densities of the Model Parameters (each color corresponds to a different chain)

where K is a matrix. For example K for a two compartment model with first order absorption is:

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

where $k_{10} = CL/V2$, $k_{12} = Q/V2$, and $k_{21} = Q/V2$.

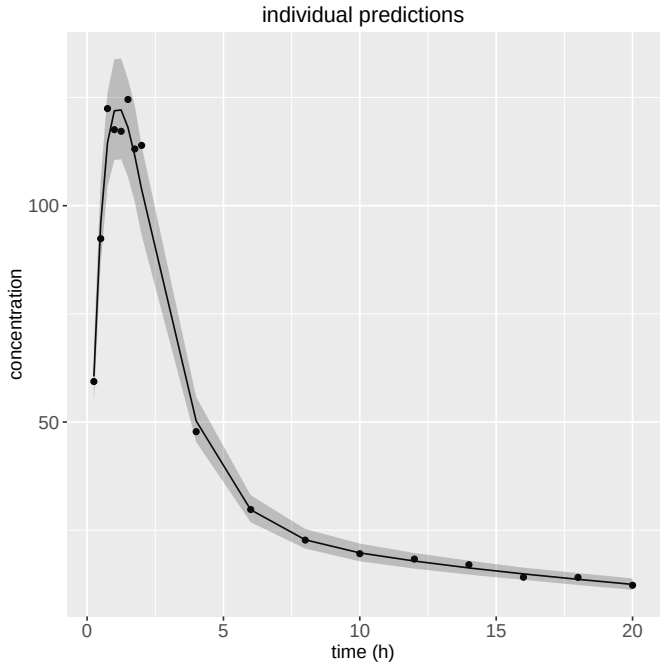

FIGURE 5. Predicted (posterior median and 90 % credible intervals) and observed plasma drug concentrations

The linear compartment model has the form:

```
linCptModel(K, theta, time, amt, rate, ii, evid, cmt, addl, ss)
```

where `K` is the matrix describing the ODE system. Since most of the parameters appear in `K`, it is not necessary to specify them again in `theta`. Instead, we only specify in `theta` parameters left out of `K`, namely the bioavailability fractions and the lag times.

General Compartment Model. Torsten may be used to fit models described by a system of first-order ODE's, i.e., differential equations of the form:

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

where x and f are vector-valued functions.

The general compartment model functions have the form:

```
<model_name>(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 ODE's defined in the function block of Stan (see section 19.2 of the Stan reference manual) and `nCmt` is the number of compartments in the model. `rel_tol`, `abs_tol`, and `max_step` are the tuning parameters for the ODE integrator, respectively the relative tolerance, the absolute tolerance, and the maximum number of steps. It is difficult to recommend values for the tuning parameters, though the authors typically use $rel\_tol = 1e-8$, $abs\_tol = 1e-8$ and $max\_step = 1e+8$. For more details see Stan's reference manual.

FIGURE 6. Stan language for fitting a two compartment model using the `linCptModel` function (abstract)

```
transformed parameters {
  matrix[3, 3] K;
  real k10;
  real k12;
  real k21;
  vector<lower = 0>[nTheta] theta[1];
  vector<lower = 0>[nt] cHat;
  vector<lower = 0>[nObs] cHatObs;
  matrix<lower = 0>[nt, 3] x;

  k10 = CL / V2;
  k12 = Q / V2;
  k21 = Q / V3;

  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;

  theta[1][1] = 1; # F1
  theta[1][2] = 1; # F2
  theta[1][3] = 1; # F3
  theta[1][4] = 0; # tlag1
  theta[1][5] = 0; # tlag2
  theta[1][6] = 0; # tlag3

  x = linCptModel(K, theta, time, amt, rate, ii, evid, cmt, addl, ss);

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

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

model{
  logCObs ~ normal(log(cHatObs), sigma);
}
```

The options for `model_name` are:

- generalCptModel_rk45
- generalCptModel_CVODES

They respectively call the built-in Runge-Kutta 4th/5th integrator, recommended for non-stiff ODE's, and CVODES integrator, recommended for stiff ODE's.

FIGURE 7. Stan language for fitting a two compartment model using the genCptModel_rk45 function (abstract)

```
functions{
  # define ODE system for two compartment model
  real[] twoCptModelODE(real t,
                        real[] x,
                        real[] theta,
                        real[] rate, # in this example, rate is treated as data
                        int[] dummy){
    real Q;
    real CL;
    real V1;
    real V2;
    real ka;
    real k12;
    real k21;
    real k10;
    real y[3];

    CL = theta[1];
    Q = theta[2];
    V1 = theta[3];
    V2 = theta[4];
    ka = theta[5];
    k10 = CL / V1;
    k12 = Q / V1;
    k21 = Q / V2;

    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;
  }
}
                              ⋮
transformed parameters {
                              ⋮
  theta[1][1] = CL;
  theta[1][2] = Q;
  theta[1][3] = V1;
  theta[1][4] = V2;
  theta[1][5] = ka;
  theta[1][6] = 1; # F1
  theta[1][7] = 1; # F2
  theta[1][8] = 1; # F3
  theta[1][9] = 0; # tlag1
  theta[1][10] = 0; # tlag2
  theta[1][11] = 0; # tlag3

  x = generalCptModel_bdf( twoCptModelODE, 3,
                    theta, time, amt, rate, ii, evid, cmt, addl, ss,
                    1e-8, 1e-8, 1e8);
                              ⋮
```

TABLE 2. Arguments of Torsten functions.

model	function name	argument names	model parameters in `theta`
one compartment model with first order absorption $\left(k_a > \frac{CL}{V_2}\right)$	`PKModelOneCpt`	`time`, `amt`, `rate`, `ii`, `evid`, `cmt`, `addl`, `ss`	CL, V_2, k_a, F_1, F_2, t_{lag1}, t_{lag2}
two compartment model with first order absorption ($k_a > \lambda_1$ [a])	`PKModelTwoCpt`	`time`, `amt`, `rate`, `ii`, `evid`, `cmt`, `addl`, `ss`	CL, Q, V_2, V_3, k_a, F_1, F_2, F_3, t_{lag1}, t_{lag2}, t_{lag2}
general linear compartment model	`linCptModel`	`K`, `time`, `amt`, `rate`, `ii`, `evid`, `cmt`, `addl`, `ss`	F_1 to F_n, t_{lag1} to t_{lagn}
general compartment models	`genCptModel_`$\star$	`ODE_system`, `nCmt`, `time`, `amt`, `rate`, `ii`, `evid`, `cmt`, `addl`, `ss`, `rel_tol`, `abs_tol`, `max_num_steps`	Parameters in ODE system, F_1 to F_n, t_{lag1} to t_{lagn}

[a] $\lambda_1 = \frac{k_{10}+k_{12}+k_{21}-\sqrt{(k_{10}+k_{12}+k_{21})^2-4k_{10}k_{21}}}{2}$ where $k_{10} = \frac{CL}{V_2}$, $k_{12} = \frac{Q}{V_2}$ and $k_{21} = \frac{Q}{V_3}$.

Summary. Table 2 summarizes how to call Torsten functions.

It is key to understand which type of models each function works for, and which method optimizes model fitting. An analytical method will always be fastest, but only applies to a few simple cases. Numerical methods that use an ODE integrator are generally applicable, but are orders of magnitude slower. The matrix exponential solution falls, speed-wise, between the analytical and the numerical solution, and can be used for all linear ODE's system.

4. Additional Example

Example 2: Effect Compartment Model. We now expand example 1 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*:

Population Model for Plasma Drug Concentration (c).

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

Furthermore we add a fourth compartment in which we measure a PD effect (see figure 8).

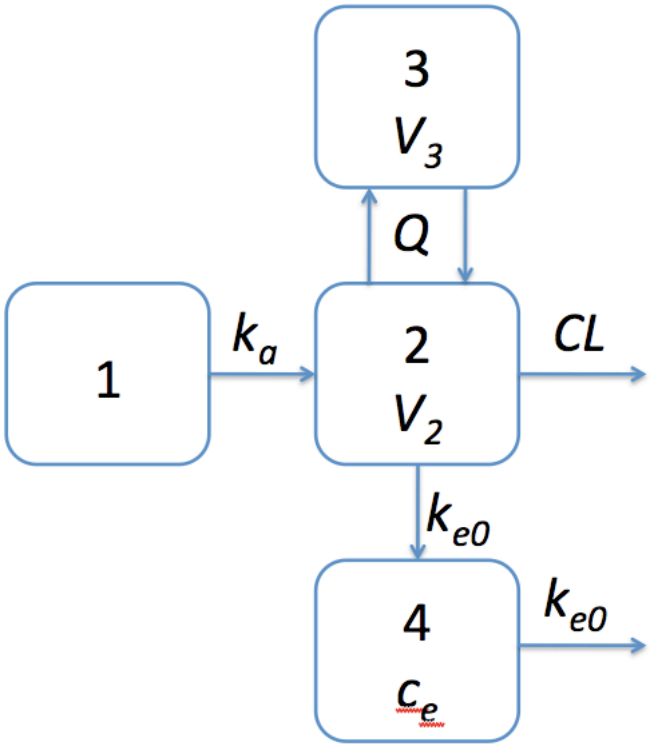

Figure 8. Effect Compartment Model

Effect Compartment Model for PD response (R).

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

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
  - Single doses: 1.25
  - 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 now simultaneously fitted to the PK and the PD data! For this four compartment model, we construct a constant rate matrix and use `linCptModel`. Correct use of Torsten requires that 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 `linCptModel` within a loop over the individual subjects rather than over the individual observations.

Results. We use the same diagnosis tools as for the previous example. The MCMC history plots (Figure 10) suggest the 4 chains have converge 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 data (Figure 12) are in close agreement with the data, notably for the sparse data case (phase IIa study). The fits to the PD data (figure 13) look good, though the data is more noisy. The model reflects the noise by producing larger confidence intervals. The estimated values of the parameters are consistent with the values used to simulate the data.

FIGURE 9. Stan language for fitting an effect compartment model using `linCptModel` (abstract)

```
transformed parameters {
  for(j in 1:nSubjects){
                        :
                        :
  Omega = quad_form_diag(rho, 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, 4, 4);
    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];

    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, 4, 4);

    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],] = linCptModel(K, parms, 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]]);

    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]]);
  }

  for(i in 1:nObs){
    cHatObs[i] = cHat[iObs[i]];
    respHatObs[i] = respHat[iObs[i]];
  }
}
                        :
                        :
}
```

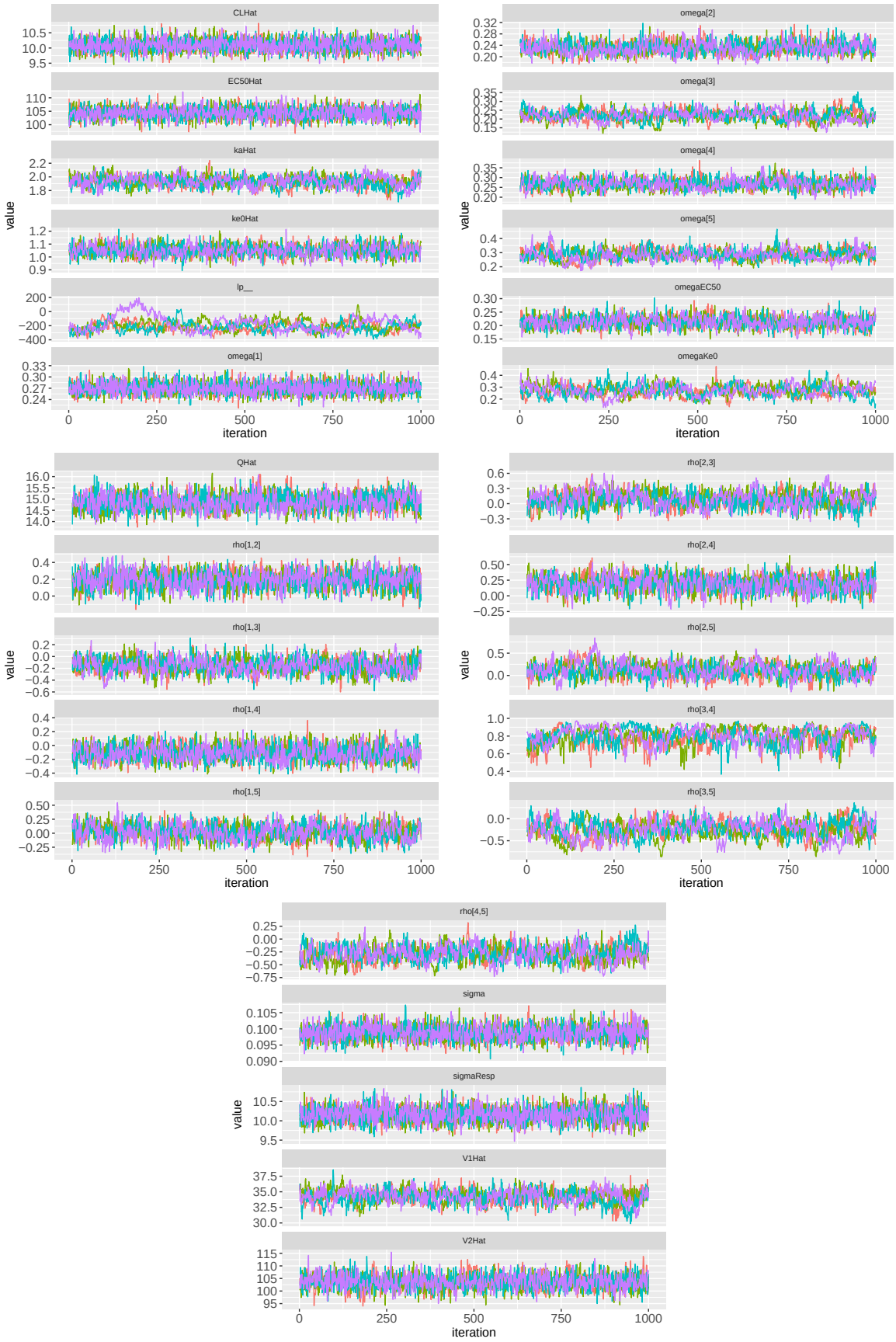

FIGURE 10. MCMC history plots for the parameters of a two compartment model with first order absorption (each color corresponds to a different chain)

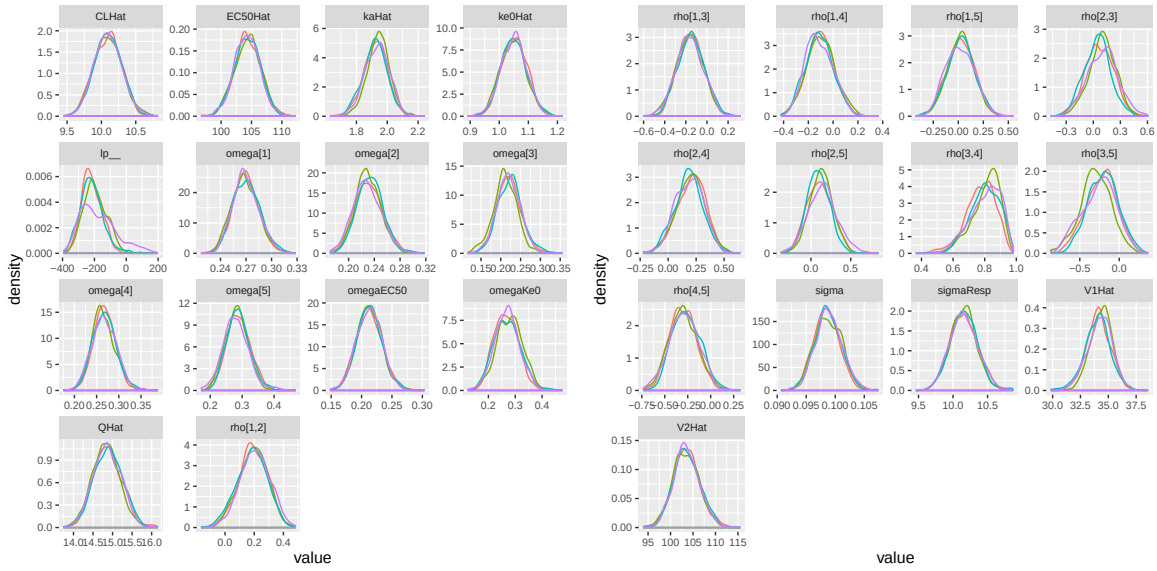

FIGURE 11. Posterior Marginal Densities of the Model Parameters (each color corresponds to a different chain)

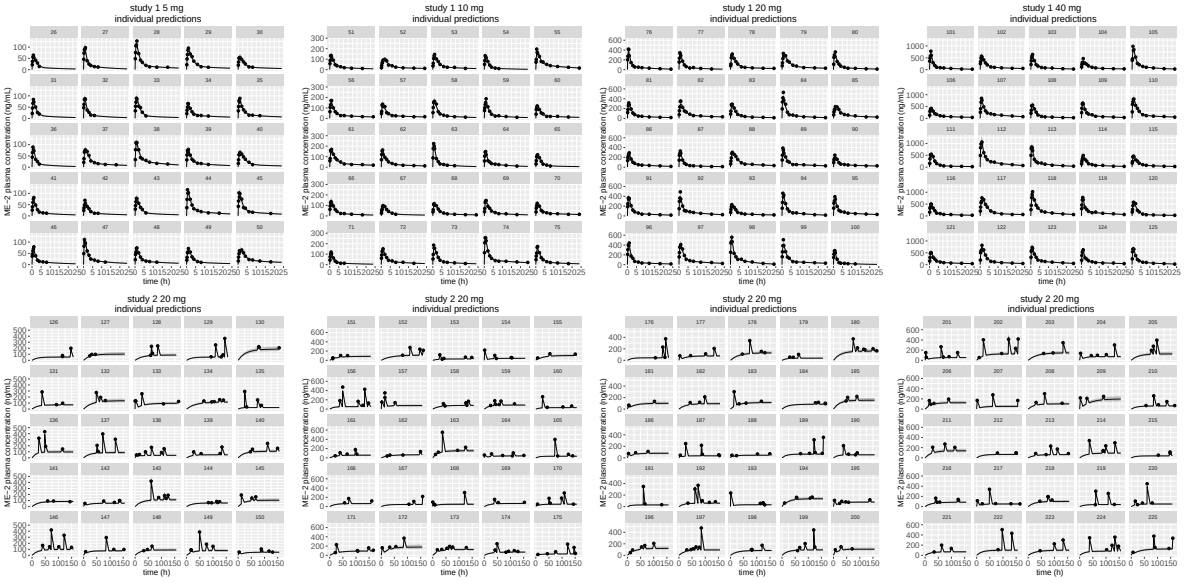

FIGURE 12. Predicted (posterior median and 90 % credible intervals) and observed plasma drug concentrations

FIGURE 13. Predicted (posterior median and 90 % credible intervals) and observed PD Response

TABLE 3. 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
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	103.455	105.472	109.583	488.296	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	164.741	1.000
EC50Hat	104.337	0.040	2.100	100.169	102.909	104.345	105.768	108.351	744.041	1.000
sigma	0.099	0.000	0.002	0.095	0.097	0.099	0.100	0.103	906.342	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	190.193	1.000

5. Under the Hood Design

We here discuss some background theory and the design of Torsten at a C++ level.

Our approach is heavily based on the *BUGS model library*, a prototype PKPD model library for WinBUGS (`https://bitbucket.org/metrumrg/bugsmodellibrary/wiki/Home`), developed by Metrum Research Group in 2009.

This section is intended for developers. No knowledge of C++ is required for users.

Computing Amounts in an ODE-based model. Most PKPD model are based on ODE's that describe how PK and PD amounts evolve over time. For instance, the following ODE's describe drug diffusion in a one compartment model with a first-order absorption from the gut:

$$\begin{aligned} \frac{dGUT}{dt} &= -kaGUT \\ \frac{dCENT}{dt} &= kaGUT - \frac{CL}{V_1}CENT \end{aligned}$$

If the ODE's fully describe the PKPD system, knowing the state y_0 at time t_0 fully defines the solution at finite times. Exploiting this property, Torsten calculates the evolution of amounts in each compartment from one event to the other. The initial conditions of the ODE system are specified by the previous event, and the ODE's are integrated from $t_{previous}$ to $t_{current}$.

Note we cannot simply integrate from t_{first} to t_{last} because the ODE's do not describe exterior interventions, such as additional dosing. Torsten treats these interventions independently. Most importantly, Torsten only integrates between t_0 and t_1 if no exterior interventions occur during this interval. It is key to properly handle the *event schedule*.

All five functions in Torsten call the C++ function `pred`, which does three things:

(1) Augment the event schedule to include all events that alter the system
(2) Calculate the amounts in each compartment at each event of the augmented schedule
    - Compute the *natural* evolution of the system by integrating ODE's
    - Compute alterations due to exterior interventions
(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 receives multiple doses at a regular time interval. Consider:

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

This Event specifies that a time 0 (TIME = 0), a patient receives a 1500 mg (AMT = 1500) drug dose (EVID = 1) in the gut (CMT = 1), and will receive an additional dose every 10 hours (II = 10) until the patient has received a total of 5 doses (ADDL = 4, being the number of additional doses, + 1, the original dose). Such an Event really corresponds to 5 dosing events.

To integrate the ODE's, `pred` calls `pred1` (prediction for one event) or `predSS` (prediction for one event if the system is in a steady state, i.e SS = 1). `pred1` and `predSS` are functors and get constructed differently by each Torsten function. Under `PKModelOneCpt`, `pred1` analytically computes the solution, while under `generalCptModel_rk45` it numerically solves the ODE's.

Structure of a Torsten Function. Under the structural scheme described above, 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

Figure 14 shows the C++ code for `PKModelOneCpt`.

Implementing Torsten in Stan.

Modifications in Stan-math. All five Torsten functions are located under the `Torsten` directory, under `stan/math`. We modified `rev/math` to include the `torsten/torsten.hpp` header file. The code can be found on GitHub: `https://github.com/charlesm93/math`

Modifications in Stan. Further modification are done in Stan to expose the Torsten functions to the Stan language. We edited `function_signatures.h` to expose `PKModelOneCpt`, `PKModelTwoCpt`, and `linCptModel`. The general compartment model 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 closely the example of `integrate_ode_rk45` and `integrate_ode_bdf`.

The code can be found on GitHub: `https://github.com/charlesm93/stan`

FIGURE 14. C++ code for the PKModelOneCpt function (abstract)

```
template <typename T0, typename T1, typename T2, typename T3, typename T4>
Eigen::Matrix <typename promote_args<T0, T1, T2, T3, T4>::type, Eigen::Dynamic,
  Eigen::Dynamic>
PKModelOneCpt(const std::vector< Eigen::Matrix<T0, Eigen::Dynamic, 1> >& pMatrix,
                    const std::vector<T1>& time,
                    const std::vector<T2>& amt,
                    const std::vector<T3>& rate,
                    const std::vector<T4>& ii,
                    const std::vector<int>& evid,
                    const std::vector<int>& cmt,
                    const std::vector<int>& addl,
                    const std::vector<int>& ss) {

  using std::vector;
  using Eigen::Dynamic;
  using Eigen::Matrix;
  using boost::math::tools::promote_args;
  using stan::math::check_positive_finite;

  PKModel model("OneCptModel");

  // Check arguments
  static const char* function("PKModelOneCpt");
  pmetricsCheck(pMatrix, time, amt, rate, ii, evid, cmt, addl, ss, function, model);
  for(int i=0; i<pMatrix.size(); i++) {
    check_positive_finite(function, "PK parameter CL", pMatrix[i](0,0));
    check_positive_finite(function, "PK parameter V2", pMatrix[i](1,0));
    check_positive_finite(function, "PK parameter ka", pMatrix[i](2,0));
  }
  std::string message4 = ", but must equal the number of parameters in the model: "
    + boost::lexical_cast<string>(model.GetNParameter()) + "!";
  const char* length_error4 = message4.c_str();
  if (!(pMatrix[0].size() == model.GetNParameter()))
    stan::math::invalid_argument(function,
    "The number of parameters per event (length of a vector in the first argument) is",
    pMatrix[0].size(), "", length_error4);

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

  // Construct dummy matrix for last argument of pred
  Eigen::Matrix<double, Dynamic, Dynamic> dummy_system(0,0);

  Matrix <typename promote_args<typename promote_args<T0, T1, T2, T3, T4>::type,
    double>::type, Dynamic, Dynamic> pred;
  pred = Pred(pMatrix, time, amt, rate, ii, evid, cmt, addl, ss, model,
    dummy_ode(), dummy_system);

  return pred;
}
```