Received: 16 July 2022 | Revised: 5 January 2023 | Accepted: 11 January 2023

DOI: 10.1002/psp4.12926

TUTORIAL

Bayesian PBPK modeling using R/Stan/Torsten and Julia/SciML/Turing.Jl

Ahmed Elmokadem[1] | **Yi Zhang**[2] | **Timothy Knab**[1] | **Eric Jordie**[1] | **William R. Gillespie**[1]

[1]Metrum Research Group, Tariffville, Connecticut, USA

[2]Sage Therapeutics, Inc., Cambridge, Massachusetts, USA

Correspondence
Ahmed Elmokadem, Metrum Research Group, 2 Tunxis Rd. Suite 112, Tariffville, CT 06081, USA.
Email: ahmede@metrumrg.com

Abstract

Physiologically-based pharmacokinetic (PBPK) models are mechanistic models that are built based on an investigator's prior knowledge of the in vivo system of interest. Bayesian inference incorporates an investigator's prior knowledge of parameters while using the data to update this knowledge. As such, Bayesian tools are well-suited to infer PBPK model parameters using the strong prior knowledge available while quantifying the uncertainty on these parameters. This tutorial demonstrates a full population Bayesian PBPK analysis framework using R/Stan/Torsten and Julia/SciML/Turing.jl.

OVERVIEW

Physiologically-based pharmacokinetic (PBPK) models are mechanistic models that describe the pharmacokinetic (PK) properties of the drug of interest. The parameters of PBPK models can be classified into physiological (e.g., tissue volumes and blood flow rates) and drug-specific (e.g., tissue to plasma partition coefficients and clearance) parameters. Usually, there is strong prior knowledge of the physiological parameters because these are accumulated through years of experimentation. Additionally, in vitro methods could provide prior knowledge of the drug-specific parameters, however, there is much more uncertainty around these parameters, primarily tissue to plasma partition coefficients, due to limited data or the unreliability of the indirect in silico methods that use the drug's physicochemical properties to calculate partition coefficients.[1]

Bayesian inference in its essence depends on Bayes rule:

$$p(\theta|d) = \frac{p(d|\theta).p(\theta)}{p(d)} \quad (1)$$

where d is the data and θ represents the model parameters. Bayesian inference provides a framework to leverage prior knowledge of the parameters of interest ($p(\theta)$) and the likelihood of the data given the chosen parameters ($p(d|\theta)$). It also allows for the quantification of the uncertainty around the parameter values.

As such, Bayesian frameworks are well-suited to make inferences on PBPK model parameters as they make use of the relatively informed prior knowledge of the physiological and drug-specific parameters and can quantify the uncertainty around those parameters that are not known with enough precision, such as clearance or drug:plasma partition coefficients.[2–6] This approach is also useful during the incremental acquisition of new data in the drug development process because Bayesian inference has the capability to sequentially update the initially developed PBPK model with the new data while quantifying the uncertainty at each stage.[7]

Open-source, domain-specific tools, such as MCSim, BUGS, and Stan, have made Bayesian analysis much more accessible to scientists.[8–10] Torsten is a library of C++ functions that aid in the Bayesian analysis of pharmacometric

data using Stan https://github.com/metrumresearchgroup/Torsten.[11,12] Additionally, the open-source Julia packages, Turing.jl and Soss.jl, make Bayesian analysis possible while taking advantage of the fact that they are fully written in Julia. This circumvents the two-language problem that tools like Stan/Torsten suffer from, because these are written in C++ for efficiency but models are usually run and analyzed using a more convenient language with simpler syntax like R or Python, putting the burden of learning two different languages on the investigator.[13-15] Unlike Torsten, however, these open-source Julia tools are general purpose tools and not pharmacometric-oriented so they cannot handle complex pharmacometric events.

In this tutorial, we demonstrate workflows for Bayesian analysis of a population hierarchical PBPK model using R/Stan/Torsten and Julia/SciML/Turing.jl. The modeled drug was mavoglurant, a small molecule drug candidate for the treatment of fragile-X-syndrome that was discontinued in 2014 after disappointing trial results.[16] This drug was chosen as a case study mainly because of the availability of open access individual level data and because the purpose of this tutorial is to demonstrate workflows with general applicability rather than focusing on a single drug, choosing this particular drug, even though it was discontinued, was deemed acceptable. All models and run scripts are found in the associated Github repository https://github.com/metrumresearchgroup/BayesPBPK-tutorial.

DATA

The mavoglurant PK data that were used for this case study were part of study A2121 that characterized the PK of mavoglurant following a 10 min i.v. infusion dosing in healthy volunteers.[6] The data were freely available as a comma separated variable (csv) file publicly shared by the nlmixr[17] team in support of an application for nonlinear mixed effects modeling with nlmixr. The dataset contained PK observations from 120 subjects who were administered i.v. infusions of mavoglurant. For the purpose of this tutorial, only the first 20 subjects were analyzed. These subjects were administered single 25 or 37.5 mg doses of mavoglurant that were infused with rates ranging from 75 to 225 mg/h. The dataset included a weight covariate (WT), which was used to calculate an individual's physiological parameters in the PBPK model.

PBPK MODEL

The mavoglurant PBPK model was implemented as previously described.[6] The model was composed of 16-well stirred compartments representing different body organs and drug distribution was dictated by blood flow rate to each organ (Figure 1). Mavoglurant clearance was assumed to be solely hepatic. The model parameters that were fixed (v) were physiological parameters (organ volumes OV and cardiac output CO) and several tissue:plasma partition coefficients (Kb). The physiological parameters were allometrically scaled based on weight (see the file model/mavoPBPKGenODE.stan in the associated Github repository for reference). Organ flow rates (Q) were calculated as fractions of CO. Estimated parameters (θ) were intrinsic clearance ($CLint$) and tissue:plasma partition coefficients for the brain ($KbBR$), muscle ($KbMU$), adipose ($KbAD$), bone ($KbBO$), and the rest of body ($KbRB$).

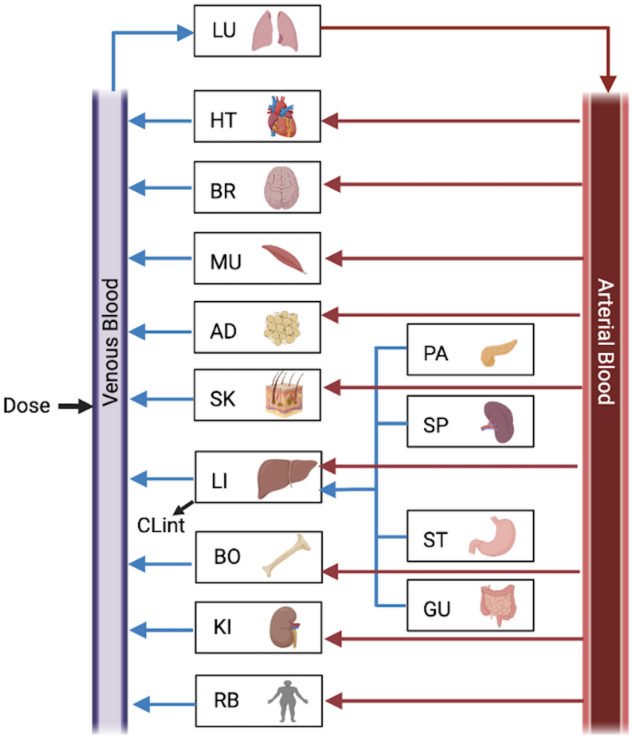

FIGURE 1 PBPK model structure. LU, HT, BR, MU, AD, SK, SP, PA, ST, GU, LI, BO, KI, and RB represent lungs, heart, brain, muscle, adipose, skin, spleen, pancreas, stomach, gut, liver, bone, kidneys, and rest of body compartments, respectively. CLint, intrinsic clearance; PBPK, physiologically-based pharmacokinetic.

STATISTICAL MODEL

The statistical model is illustrated in Equations 2–14. The drug's observed plasma concentration data for subject i at time j (c_{ij}) was assumed to be lognormally distributed around the predicted geometric mean concentrations $\hat{c}_{ij}$ with a variance σ^2 of the log-transformed concentrations. This represented the first level of hierarchy in random effects. The $\hat{c}_{ij}$ was predicted by the PBPK model (f_{PBPK}) that

took timepoints (t_{ij}), doses (D_i), and parameters (p_i) as inputs. The variable parameter vector, θ_i, contained partition coefficient values that were assumed to be independent of subjects, whereas *CLint* was assumed to be variable between subjects with a lognormal distribution around a geometric mean $\widehat{CLint}$ and a variance ω^2 of the log-transformed parameter. This represented the second level of random effects hierarchy.

$$\log(c_{ij}) \sim N(\log(\hat{c}_{ij}), \sigma^2) \tag{2}$$

$$\hat{c}_{ij} = f_{PBPK}(t_{ij}, D_i, p_i) \tag{3}$$

$$p_i = [\theta_i, v_i] \tag{4}$$

$$\theta_i = [CLint_i, KbBR, KbMU, KbAD, KbBO, KbRB] \tag{5}$$

$$\log(CLint_i) \sim N(\log(\widehat{CLint}), \omega^2) \tag{6}$$

$$\widehat{CLint} \sim lognormal(7.1, 0.25^2) \tag{7}$$

$$KbBR \sim lognormal(1.1, 0.25^2) \tag{8}$$

$$KbMU \sim lognormal(0.3, 0.25^2) \tag{9}$$

$$KbAD \sim lognormal(2, 0.25^2) \tag{10}$$

$$KbBO \sim lognormal(0.03, 0.25^2) \tag{11}$$

$$KbRB \sim lognormal(0.3, 0.25^2) \tag{12}$$

$$\omega \sim half - Caucy(0, 0.5) \tag{13}$$

$$\sigma \sim half - Caucy(0, 0.5) \tag{14}$$

Equations 7–14 represent the prior distributions of the parameters to be estimated. Prior distributions are crucial for Bayesian inference and represent the investigator's prior knowledge that gets updated by the data. Informative prior distributions can help alleviate data biases but they become less important when the sample size gets larger.[18] For this tutorial, we used relatively informative priors because the means of the prior distributions for intrinsic clearance and partition coefficients were based on the final estimates (intrinsic clearance) or initial values (partition coefficients) used in the previous nonlinear mixed effects run in nlmixr. The variances for the same prior distributions were chosen to represent a reasonable 25% relative standard deviation around the means. The prior distributions for ω and σ were chosen to have large enough tails to encompass a relatively larger range of potential values.

BAYESIAN INFERENCE USING R/STAN/TORSTEN

Stan is an open-source probabilistic programming language that can be used for full Bayesian statistical inference.[9] Stan has multiple interfaces, like command line (CmdStan), R (RStan), and Python (PyStan) interfaces. The Stan language is expressed as program blocks where variables can be declared. For example, the model block can be declared as:

```
model {
// … declarations
}
```

The main program blocks of interest are the data (where data items are declared), parameters (where parameters are declared), and model (where the model is defined) blocks. Additionally, transformed data, transformed parameters, and generated quantities blocks were used in this tutorial to do further manipulations to data, parameters, and to generate outputs, respectively. Stan also allows for user-defined functions to be added to a functions block and this was used to define the PBPK model.

Torsten is a library of Stan functions built to facilitate analysis of pharmacometric data.[11,12] It can handle scheduled events that follow NMTRAN conventions with data items, such as AMT, DV, TIME, and EVID.[19] Torsten contains functions to build specific linear compartmental models as one and two-compartment models with first-order absorption into the central compartment, general linear models that can be expressed as a system of linear ordinary differential equations (ODEs), general compartmental models that can be expressed as a system of ODEs. For this tutorial, the PBPK model was built using Torsten's general ODE and general linear ODE functions.

GENERAL ODE APPLICATION

Torsten's general ODE function is pmx_solve_[solver] where solver can be adams, rk45, or bdf. This function takes a number of arguments that include an ODE function that defines the system of ODEs as well as other arguments that describe the pharmacometric data. The version of this function that handles population data is pmx_solve_group_[solver]. A full definition of the function is:

```
pmx_solve_group_[adams || rk45 || bdf]
(ODE_rhs, int nCmt, time, amt, rate, ii,
evid, cmt, addl, ss, theta, [ biovar, tlag,
real[,] x_r, int [,] x_i, real rel_tol,
real abs_tol, int max_step, real as_rel_
tol, real as_abs_tol, int as_max_step ] )
```

The general ODE Stan/Torsten PBPK model can be found in the associated Github repository under model/mavoPBPKGenODE.stan. The model included an ODE function PBPKModelOD that was built using the user-defined functions block and it took mainly the model parameters as inputs and returned the system of PBPK model ODEs.

The data block defined the data items that included the number of subjects, doses, infusion rates, times, observations, body weights, etc... The parameters to be estimated were defined in the parameters block:

```
parameters{
  real<lower = 0> CLintHat;
  real<lowerc= 0> KbBR;
  real<lower = 0> KbMU;
  real<lower = 0> KbAD;
  real<lower = 0> KbBO;
  real<lower = 0> KbRB;
  // residual error
  real<lower = 0> sigma;
  // IIV parameters
  vector<lower = 0>[nIIV] omega;
  matrix[nIIV, nSubject] etaStd;
}
```

As can be seen, the size and boundaries on the parameters can be specified in this block.

The transformed parameters block was where the general ODE function was called using the rk45 solver as follows:

```
  x = pmx_solve_group_rk45(PBPKModelODE,
nCmt, len, time, amt, rate, ii, evid,
cmt, addl, ss, parms, biovar, tlag, WT,
idummy, 1e-6, 1e-6, 1e6);
```

The prior distributions and the likelihood were defined in the model block as follows:

```
model{
  // Priors
  CLintHat ~ lognormal(7.1, 0.25);
  KbBR ~ lognormal(1.1, 0.25);
  KbMU ~ lognormal(0.3, 0.25);
  KbAD ~ lognormal(2, 0.25);
  KbBO ~ lognormal(0.03, 0.25);
  KbRB ~ lognormal(0.3, 0.25);
  sigma ~ cauchy(0, 0.5);
  omega ~ cauchy(0, 0.5);
  to_vector(etaStd)  ~ normal(0, 1);
  // observed data likelihood
  logCObs ~ normal(log(cHatObs), sigma);
}
```

Model outputs were defined in the generated quantities block which looks very similar to the transformed parameters block, but instead of estimating the parameters, this block can be used to run simulations based on the estimated parameters.

All data wrangling, setting up initial values, running the model, and post-processing exercises were carried out in R[20] using the R script script/mavoPBPKGenODE.R in the associated Github repository. The model was run using the R package, cmdstanr, which provides an R interface for CmdStan.

The sampler used was the No U-Turn Sampler (NUTS) and four Markov Chain Monte Carlo (MCMC) chains were run in parallel with 500 samples for each chain (250 warmup and 250 sampling) with an acceptance criterion of 0.8 on a local machine. These parameters were chosen as reasonable starting points but they can be revisited depending on the Bayesian model diagnostics. Multiple chains were run to confirm convergence to the same distributions for the estimated parameters and the model run time was about 58 h. Warmup samples are necessary for the initial parameter search before homing in on the real parameter distributions and were eventually discarded.

Model diagnostics showed good mixing of posterior samples. This is evident from the trace plots that show the "fuzzy caterpillar" appearance as opposed to the "wiggly snake" appearance that would indicate bad mixing (Figure 2). Additionally, the overlaid density plots from all four chains show that they converged to approximately the same distribution (Figure 2). Tables 1 and S1 show summaries of the posterior parameter samples. Table S1 also shows more diagnostics like the Gelman-Rubin statistic $\widehat{R}$[21] and effective sample size (ESS). The former is a measure of convergence and an acceptable convergence value would be close to 1 (not larger than 1.05), which was the case for all inferred parameters. The latter highlights uncertainty in parameter estimates as a result of autocorrelation within the chains and the larger this value is the better. In this example, the large ESS-bulk (bulk of the distribution) and ESS-tail (tails of the distribution) values indicate a large enough number of independent samples and impart confidence in model estimates. In case of poor sampling efficiency, increasing the acceptance criterion or the sample size can help resolve this issue.

A posterior predictive check (PPC) is often done as a diagnostic of how a model's predictions compare to the observed data. Stan makes it simple to get those model predictions using the generated quantities block where the sampled parameters can be used to run simulations. Figures 3 and S1 show the summary, and individual-level, PPC results that demonstrated good agreement between observed data and model predictions.

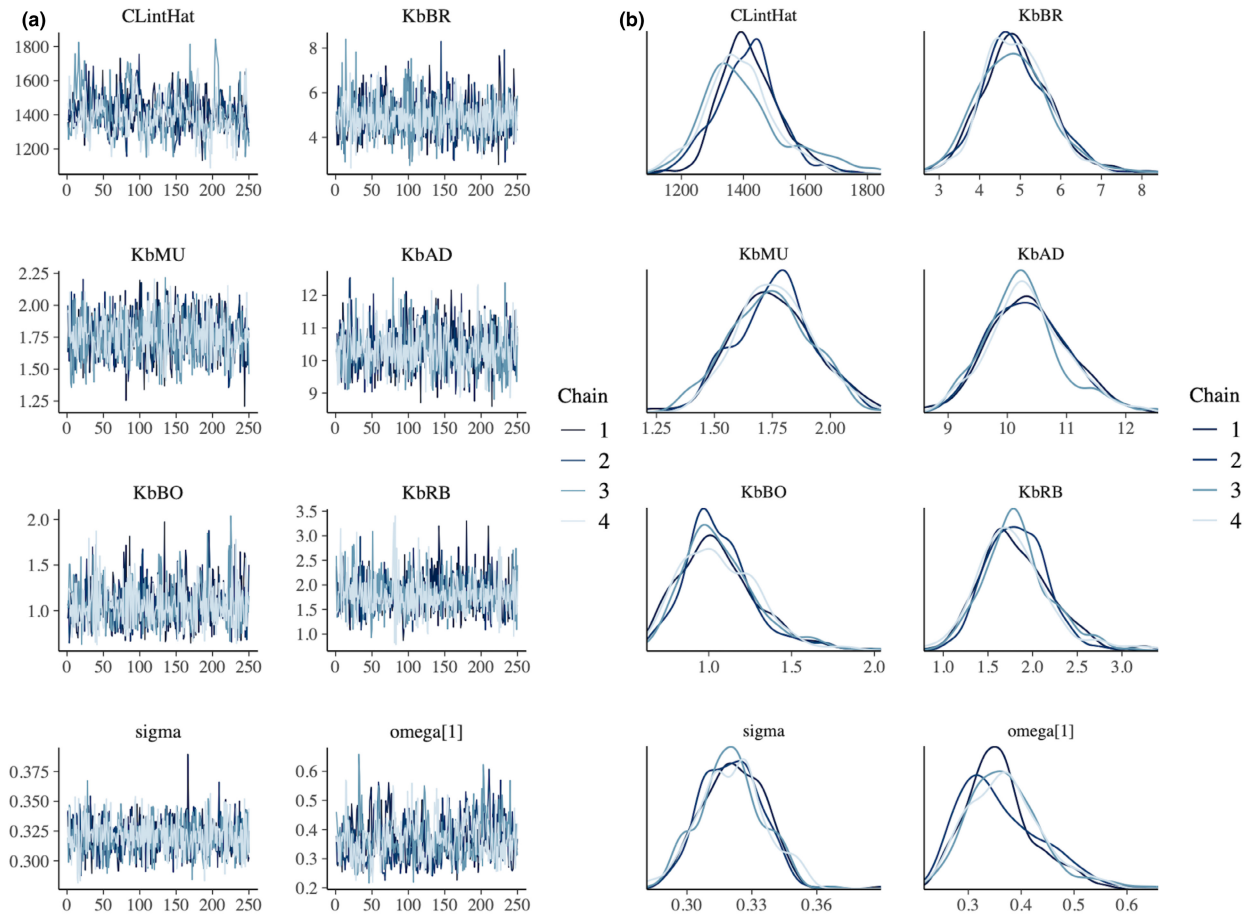

FIGURE 2 MCMC diagnostics for the Stan/Torsten general ODE application. (a) Trace plots of MCMC chains. (b) Density plots of posterior distributions. CLint, intrinsic clearance; KbBR, KbMU, KbAD, KbBO, and KbRB are the brain, muscle, adipose, bone, and rest of body tissue:plasma partition coefficients; omega[1], standard deviation of CLintHat intersubject variability; sigma, residual error; MCMC, Markov Chain Monte Carlo; ODE, ordinary differential equation.

GENERAL LINEAR ODE APPLICATION

Because the mavoglurant PBPK model ODEs are linear, i.e., they can be represented as $y' = Ky$, where y' represents the derivatives of the state variables y and K is the coefficient matrix, the much more efficient general linear ODE function pmx_solve_linode() can be used to build and solve the PBPK model. Instead of the ODE function, this function takes the coefficient matrix K as an input together with the other pharmacometric inputs.

```
pmx_solve_linode(time, amt, rate, ii,
evid, cmt, addl, ss, K, biovar, tlag)
```

The linear ODE Stan/Torsten model can be found in the associated Github repository under model/mavoPBPKLinODE.stan. A 16×16 K matrix was built under the transformed parameters block and populated with the coefficients of the PBPK model. The rest of the model code was mostly similar to the general ODE application.

Similar sampling conditions as the general ODE application were used (NUTS with 0.8 acceptance criterion, four parallel chains, and 500 samples for each chain). Model run time was about 1.4 h, which is about 41 times faster than the general ODE application.

Model MCMC diagnostics (Figure S2) and PPCs (Figures 4 and S3) showed similar results to the general ODE application and generally demonstrated good sample convergence and the model's ability to capture observed data. A summary of the posterior parameter samples is shown in Tables 1 and S2 together with the $\widehat{R}$ and ESS diagnostics and they demonstrated good convergence and ESS, respectively.

BAYESIAN INFERENCE USING JULIA/SCIML/TURING.JL

Turing.jl is an open-source Julia package for general-purpose probabilistic programming.[13] An advantage of Turing.jl is its use of regular Julia syntax and composability with other Julia packages, including the SciML ecosystem

TABLE 1 Parameter estimates summary

Parameter	Median (90% CI)		
	Stan/Torsten (general ODE solver)	Stan/Torsten (linear ODE solver)	Turing.jl
CLintHat/$\widehat{CLint}$ (L/h)	1395 (1239, 1604)	1393 (1230, 1582)	1390 (1216, 1585)
KbBR	4.83 (3.64, 6.36)	4.92 (3.66, 6.36)	4.9 (3.54, 6.54)
KbMU	1.75 (1.49, 2.04)	1.75 (1.49, 2.03)	1.75 (1.49, 2.04)
KbAD	10.3 (9.31, 11.51)	10.27 (9.23, 11.53)	10.3 (9.37, 11.46)
KbBO	1.03 (0.74, 1.47)	1.04 (0.735, 1.47)	1.04 (0.74, 1.42)
KbRB	1.77 (1.26, 2.46)	1.77 (1.23, 2.43)	1.8 (1.26, 2.43)
omega [1]/ω	0.358 (0.263, 0.499)	0.35 (0.258, 0.483)	0.352 (0.27, 0.484)
sigma/σ	0.32 (0.298, 0.345)	0.32 (0.297, 0.346)	0.32 (0.297, 0.345)

Abbreviations: CI, credible interval; CLintHat and $\widehat{CLint}$, population intrinsic clearance; KbBR, KbMU, KbAD, KbBO, and KbRB are the brain, muscle, adipose, bone, and rest of body tissue:plasma partition coefficients, respectively; ODE, ordinary differential equation; omega [1] and ω = standard deviation of $\widehat{CLint}$ intersubject variability; sigma and σ = residual error.

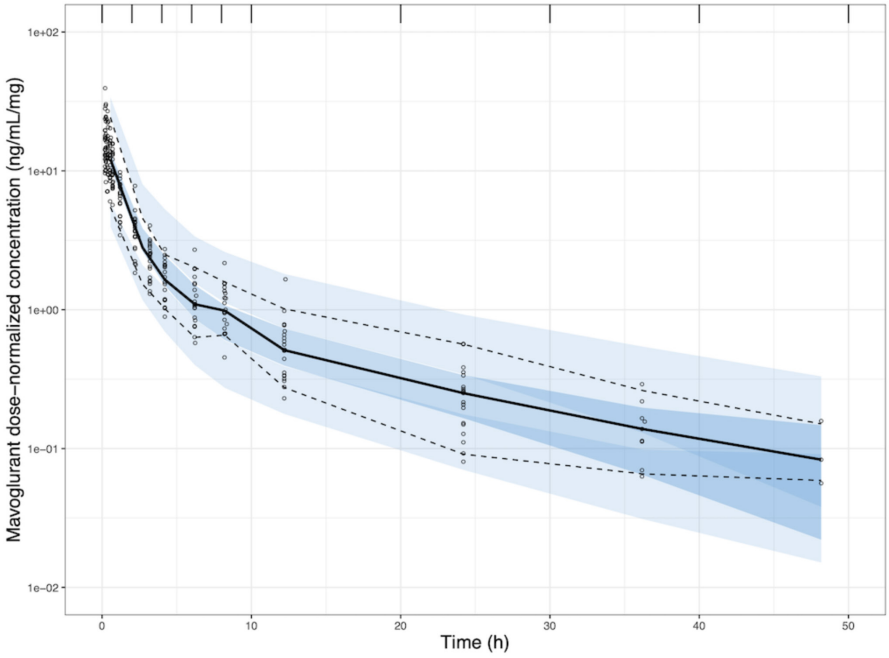

FIGURE 3 Summary of posterior predictive check for the Stan/Torsten general ODE application. Open circles represent observed data. Solid and dashed lines represent median, 5th, and 95th percentiles of observed data. Dark and light blue bands represent 95% credible intervals around the median, 5th, and 95th percentiles of model prediction. ODE, ordinary differential equation.

with its DifferentialEquations.jl package that provides state-of-the-art differential equation solvers.[22] As a result, the PBPK model was built using DifferentialEquations.jl by first defining an ODE function, which takes standard arguments (derivatives du, state variables u, parameters p, and time t) and returns the updated derivatives vector:

```
function PBPKODE!(du, u, p, t)
      du[1] = …
      du[2] = …
end
```

The p vector carried the parameters to be estimated, whereas all fixed parameters were explicitly defined in the model function body (see the model file model/mavoPBPKGenODE.jl in the associated Github repository). The p vector also carried the weight covariate and the mavoglurant infusion rate to be able to individualize those values.

The ODE problem was then defined using the ODEProblem() function:

```
prob = ODEProblem(PBPKODE!, u0, tspan, p)
```

where u0 is the vector of initial state values and tspan is the time span for simulation. This prob object was then used for the Bayesian inference in Turing.jl.

The composability of Julia packages allowed for the use of the prob object for running a global sensitivity analysis (GSA) on the model parameters using the

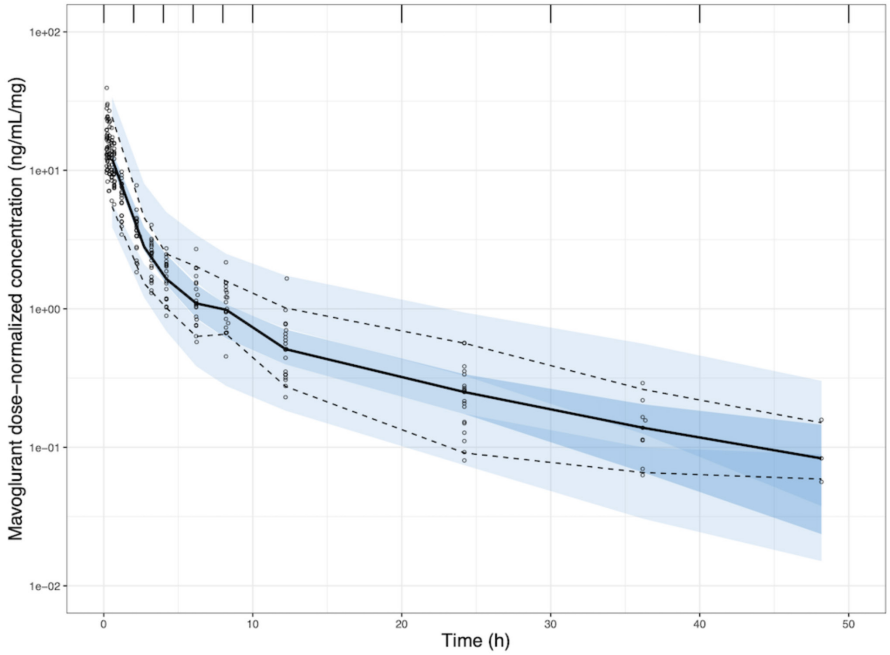

FIGURE 4 Summary of posterior predictive check for the Stan/Torsten linear ODE application. Open circles represent observed data. Solid and dashed lines represent median, 5th, and 95th percentiles of observed data. Dark and light blue bands represent 95% credible intervals around the median, 5th, and 95th percentiles of model prediction. ODE, ordinary differential equation.

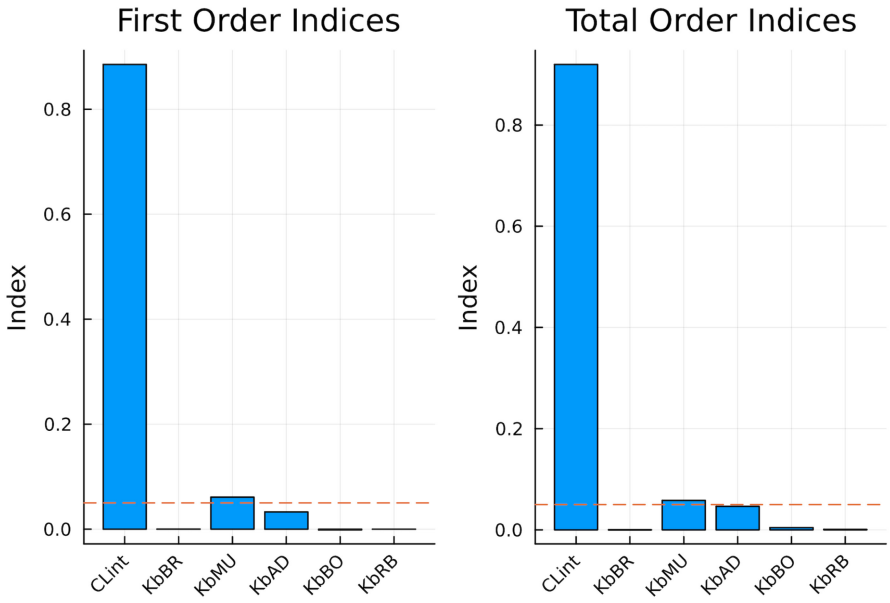

FIGURE 5 Global sensitivity analysis of the PBPK model parameters of interest using the Sobol method. Horizontal red dashed line represents the arbitrary cutoff value of 0.05. CLint, intrinsic clearance; KbBR, KbMU, KbAD, KbBO, and KbRB are the brain, muscle, adipose, bone, and rest of body tissue:plasma partition coefficients; PBPK, physiologically-based pharmacokinetic.

GlobalSensitivity.jl package.[23] GSA can be extremely helpful in the context of PBPK modeling as it can identify how the in vivo drug behavior is impacted by its physicochemical properties and, as such, can help in the drug development process.[24] GSA can also help assess the impact of the level of uncertainty in a drug-specific parameter on a model's output of interest and this would highlight the influential parameters among the many PBPK parameters that can be fine-tuned to improve model predictions.[24]

GSA evaluates the influence of a given parameter on a certain model output as well as the influence of its interactions with other parameters. This gives GSA an advantage over local parameter sensitivity analysis that only evaluates univariate parameter sensitivity. A clear disadvantage of GSA, however, is the relatively large computation time that is necessary to evaluate a large number of parameter sets. To run the GSA, a function f_globsens() was built that took the parameter vector p as an argument and returned the area under the curve for mavoglurant in venous blood as the model output to run the sensitivity on. GlobalSensitivity.jl package offers a number of methods and the one used in our example was the Sobol method.[25] This method was used to provide first-order and total sensitivity indices. The former quantifies the influence of varying an individual parameter alone, whereas the latter quantifies the influence of varying the parameter together

with its interaction with other parameters. To assess whether a parameter is influential or not, a widely used arbitrary cutoff value of 0.05 was used where a parameter was regarded as being influential if its indices were higher than this value and less influential if they fell below this value.[25] This value is considered acceptable for relatively complex models and might not be stringent enough for simpler models with fewer parameters.[25] The sensitivity analysis results shown in Figure 5 highlighted CLint and KbMU as being influential with KbAD falling right behind them with indices that are just below 0.05. KbBO, KbBR, and KbRB were shown to be non-influential given how far below 0.05 their indices fell. These parameters were still included in the Bayesian inference exercise to remain consistent with the previous analysis in Stan/Torsten and the original nlmixr implementation of the model.[17] It is worth noting that the first order and total indices were very similar for all parameters indicating that there is only a marginal role for these specific parameter interactions on the model output.

The Turing.jl model was defined by first building a model function fitPBPK() using the @model macro. The arguments to the model function included the observed data, the prob object, and vectors of infusion rates, timepoints, weights, and callback events. The callback events were created using the PresetTimeCallback() function from the DiffEqCallbacks package https://diffeq.sciml.ai/stable/features/callback_library/#PresetTimeCallback. These callbacks would set the infusion rate to 0 when infusion duration is complete. Within the model function, the prior distributions of the parameters of interest were defined including population and individual level parameters. For example:

```
KbBR ~ LogNormal(1.1, 0.25)
```

To get the population predictions for the likelihood calculation, the EnsembleProblem() function was used. The function takes the prob object and a prob_func() as arguments. The latter would remake the ODE problem by looping through and updating the individual parameters, weights, timepoints, rates, and callbacks. This process is described by the following code:

```
function prob_func(prob, i, repeat)
      ps = [CLinti[i], KbBR, KbMU, KbAD,
      KbBO, KbRB, wts[i], rates[i]]
      remake(prob, p=ps, saveat=times[i],
      callback=cbs[i])
end
tmp_ensemble_prob = EnsembleProblem(prob,
prob_func=prob_func)
tmp_ensemble_sol  =  solve(tmp_ensemble_
prob, Tsit5(), trajectories=nSubject)
```

The tmp_ensemble_sol would carry the predictions for each individual, which can be saved as a continuous vector object predicted then used for the likelihood calculation as follows:

```
for i = 1:length(predicted)
 data[i] ~LogNormal(log.(predicted[i]),σ)
end
```

A model object mod was created by calling the model function and feeding it the necessary arguments. The sample() function was called to run the Bayesian inference and create the MCMC chains as follows:

```
mcmcchains = sample(mod, NUTS(250, .8),
MCMCThreads(), 250, 4)
```

Similar to the Stan/Torsten implementations, the sampler used here was NUTS with 250 warmup and 250 post-warmup samples per chain for four chains. The acceptance criterion used for the sampler was 0.8. The MCMCThreads() argument parallelized the process over four cores (one chain per core). The run time was about 16 h, which is about 3.6 times faster than the corresponding general ODE application in Stan/Torsten. Caution must be taken here, however, since this is not a direct comparison because factors that could impact run times, such as solver tolerance, were not unified across methods.

Post-processing of the MCMC chains was carried out using the MCMCChains.jl and StatsPlots.jl packages. The describe() function has a method for the MCMC chains object where it would return summary statistics (including mean standard error, $\widehat{R}$, and ESS) and quantiles for each parameter. The StatsPlots.plot() function also has a method for the MCMC chains and would return MCMC traces and density plots for the parameters (Figure S4). To generate the parameter table and to remain consistent with the previous $\widehat{R}$ and ESS calculations done on the Stan/Torsten runs, the posterior samples of the parameters were exported to R where the samples were summarized using R's posterior package tools as before (Tables 1 and S3). The posterior samples were exported by first using the DataFrame() function on the MCMC chains object to get a dataframe of the posterior samples then this dataframe was saved as a csv file to be imported to R.

The PPCs were carried out as previously discussed where a new population was generated by randomly sampling the random effects on CLint while using the posterior samples for the partition coefficient parameters and the covariates from the analysis dataset. This creates a new ensemble problem that can be simulated similarly to the fitting run. The summary PPC plot was generated using the Gadfly.jl[26] package and is shown in Figure 6. The figure shows the observed and predicted 5th, 50th, and 95th

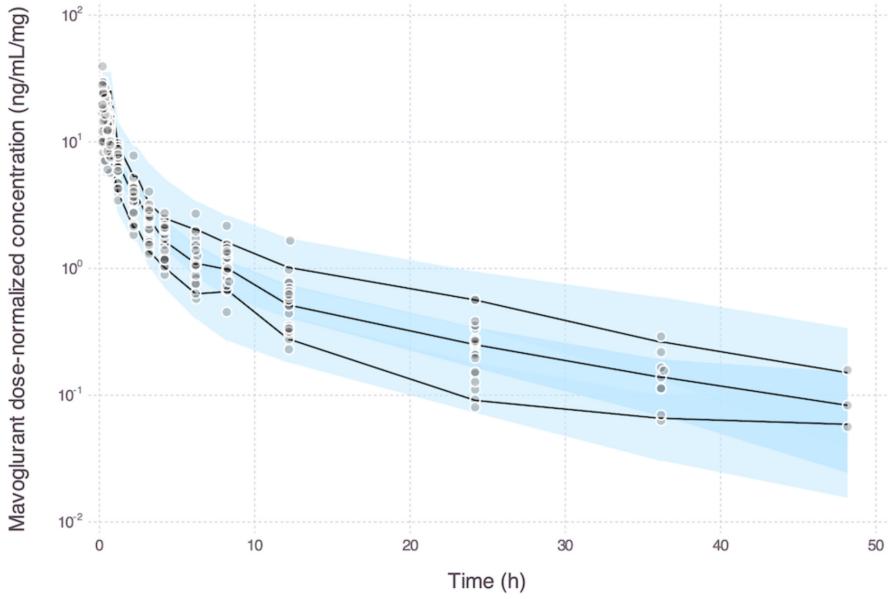

FIGURE 6 Summary of posterior predictive check for the Turing.jl application. Circles represent observed data. Solid lines represent median, 5th, and 95th percentiles of observed data. Dark and light blue bands represent 95% credible intervals around the median, 5th, and 95th percentiles of model prediction.

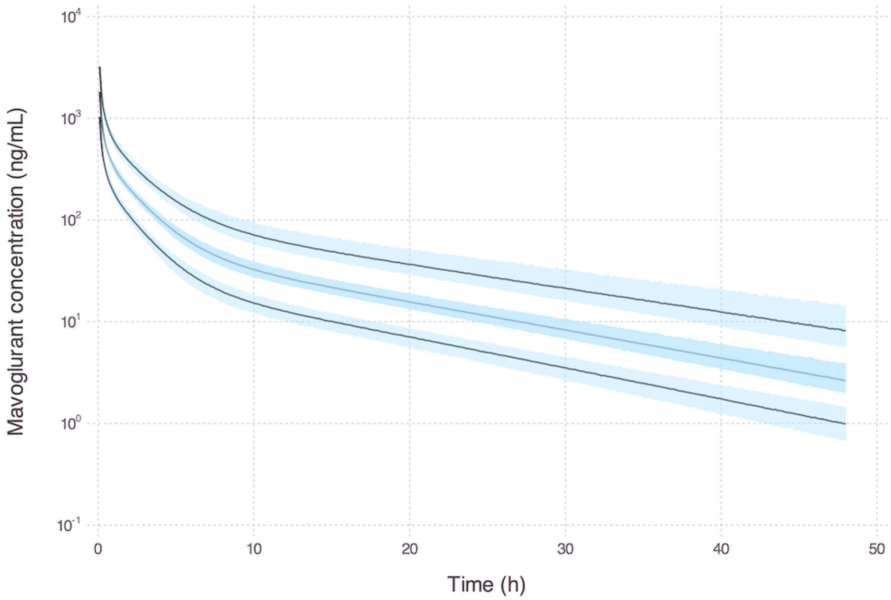

FIGURE 7 PBPK model simulation (500 subjects × 500 replicates). Dose administered was a single 50 mg i.v. infusion of mavoglurant with a rate of 300 mg/h. Solid lines represent median, 5th, and 95th percentiles of model predictions. Blue bands represent 95% credible intervals around the different percentiles. PBPK, physiologically-based pharmacokinetic.

percentiles with the 95% credible interval around the predicted percentiles. Figure S5 shows the PPCs stratified by individual that were generated as previously described by first exporting the simulation results to R and generating the plots using R tools.

Subsequently, the same Julia tools were used to explore a higher mavoglurant dose of 50 mg. A virtual population was generated (500 subjects × 500 samples) using randomly sampled weight covariates and CLint random effects. Simulation results are shown in Figure 7 and demonstrate expected higher exposures for the higher 50 mg dose.

CONCLUSIONS

This tutorial demonstrated a practical application of building population Bayesian PBPK models using two different open-source approaches in two different programming languages, namely R/Stan/Torsten and Julia/SciML/Turing.jl. In contrast to tools applying nonlinear mixed-effects modeling to calculate point estimates in pharmacometric analysis, like the open-source R package nlmixr[17] or the commercial software Monolix,[27] the approaches presented here carry the advantage of being able to run full Bayesian analyses to infer parameter posterior distributions that characterize the uncertainty around these parameters rather than just inferring the point estimates. Full Bayesian approaches also allow for the incorporation of the investigator's prior knowledge of the system, which can be substantial in PBPK applications. Other available tools that are capable of running full Bayesian analyses include the well-known software NONMEM and the newly developed Julia-based tool Pumas.[19,28] These tools, however, are commercial tools and hence, are

not readily accessible to all investigators, unlike the approaches presented in this tutorial that are based on open-source tools. The presented approaches offer an additional advantage over NONMEM in being flexible in choosing from a number of prior distributions for different parameters, whereas NONMEM is limited in prior distribution choices.[29] Furthermore, post-processing to generate predictive checks is much more convenient in Torsten or Turing.jl compared to NONMEM. Torsten carries the advantage over Turing.jl of being pharmacometric-oriented, which means it can handle very complex events in the input analysis dataset. In case the PBPK model ODEs can be represented as linear ODEs, Torsten offers the general linear ODE function, which is much more efficient than the general ODE solver options, however, this might not be an option for many PBPK applications that require nonlinearity in the model equations. A drawback of using Torsten is that it suffers from the two-language problem inherited from Stan since it is dependent on C++ libraries that are crucial for efficiency. This carries the burden of having to learn how to write Stan code on top of R that allows for convenient model running and analysis of results. Turing.jl, on the other hand, is written in Julia so it does not require learning a new language and is very easy to install while maintaining the C++ level efficiency associated with Julia. It also takes advantage of Julia composability that makes it easy to integrate with other Julia tools such as the SciML tools that provide the ODE solvers, sensitivity analysis tools, as well as the callback functionality that allows for handling relatively simple pharmacometric events. More complex pharmacometric events, however, might be cumbersome to handle with SciML/Turing.jl given that these are general purpose tools and not specifically built for pharmacometrics. Run times between the proposed approaches here cannot be directly compared because factors like solver tolerances and initial estimate generation were not unified across approaches, however, the anecdotal evidence from the runs in this tutorial suggest that the Julia/SciML/Turing.jl approach is about 3.6 times faster than the corresponding R/Stan/Torsten general ODE approach.

The general applicability of the approaches discussed here makes them valuable tools for investigators interested in building population Bayesian PBPK models in an efficient, flexible, and convenient way.

ACKNOWLEDGMENTS

Portions of this paper were presented as a poster at the American Conference of Pharmacometrics (AcoP13) in Aurora, CO, October 30 to November 2, 2022.

FUNDING INFORMATION

No funding was received for this work.

CONFLICT OF INTEREST STATEMENT

The authors declared no competing interests for this work.

ORCID

Ahmed Elmokadem https://orcid.org/0000-0002-2412-4986
William R. Gillespie https://orcid.org/0000-0002-4560-6482

REFERENCES

1. Utsey K, Gastonguay MS, Russell S, Freling R, Riggs MM, Elmokadem A. Quantification of the impact of partition coefficient prediction methods on PBPK model output using a standardized tissue composition. *Drug Metab Dispos*. 2020;48:903-916. doi:10.1124/dmd.120.090498
2. Krauss M, Burghaus R, Lippert J, et al. Using Bayesian-PBPK modeling for assessment of inter-individual variability and subgroup stratification. *In Silico Pharmacol*. 2013;1:6.
3. Kapraun DF, Schlosser PM, Nylander-French LA, Kim D, Yost EE, Druwe IL. A physiologically based pharmacokinetic model for naphthalene with inhalation and skin routes of exposure. *Toxicol Sci*. 2020;177:377-391.
4. Yang Y, Xu X, Georgopoulos PG. A Bayesian population PBPK model for multiroute chloroform exposure. *J Expos Sci Environ Epidemiol*. 2010;20:326-341. doi:10.1038/jes.2009.29
5. Bois FY, Gelman A, Jiang J, Maszle DR, Zeise L, Alexeef G. Population toxicokinetics of tetrachloroethylene. *Arch Toxicol*. 1996;70:347-355.
6. Wendling T, Dumitras S, Ogungbenro K, Aarons L. Application of a Bayesian approach to physiological modelling of mavoglurant population pharmacokinetics. *J Pharmacokinet Pharmacodyn*. 2015;42:639-657. doi:10.1007/s10928-015-9430-4
7. Carrara L, Magni P, Teutonico D, Pasotti L, Della Pasqua O, Kloprogge F. Ethambutol disposition in humans: challenges and limitations of whole-body physiologically-based pharmacokinetic modelling in early drug development. *Eur J Pharm Sci*. 2020;150:105359.
8. Bois FY. GNU MCSim: Bayesian statistical inference for SBML-coded systems biology models. *Bioinformatics*. 2009;25:1453-1454.
9. Stan Team. *Stan Modeling Language User's Guide and Reference Manual*. 2012. https://mc-stan.org
10. Ntzoufras I. *Bayesian Modeling Using WinBUGS*. John Wiley & Sons; 2011.
11. Margossian C, Gillespie WR. Stan functions for Bayesian pharmacometric modeling. *J Pharmacokinet Pharmacodyn*. 2016;43:S52.
12. Margossian CC, Zhang Y, Gillespie WR. Flexible and efficient Bayesian pharmacometrics modeling using Stan and Torsten, part I. *CPT-PSP*. 2022;11:1151-1169.
13. Ge H, Xu K, Ghahramani Z. Turing: a language for flexible probabilistic inference. In: Storkey A, Perez-Cruz F, eds. *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. PMLR; 2018:1682-1690.
14. Scherrer C. Soss. jl. 2019. https://github.com/cscherrer/Soss.jl
15. Bezanson J, Edelman A, Karpinski S, Shah VB. Julia: a fresh approach to numerical computing. *SIAM Rev*. 2017;59:65-98.
16. Hagerman R, Jacquemont S, Berry-Kravis E, et al. Mavoglurant in fragile X syndrome: results of two open-label, extension trials in adults and adolescents. *Sci Rep*. 2018;8:16970.

17. Fidler M, Wilkins JJ, Hooijmaijers R, et al. Nonlinear mixed-effects model development and simulation using nlmixr and related R open-source packages. *CPT Pharmacometrics Syst Pharmacol*. 2019;8:621-633.
18. Aczel B, Hoekstra R, Gelman A, et al. Discussion points for Bayesian inference. *Nat Hum Behav*. 2020;4:561-563.
19. Boeckmann AJ, Sheiner LB, Beal SL. *NONMEM Users Guide-Part V: Introductory Guide*. NONMEM Project Group University of California at San Francisco; 1994.
20. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing; 2018 https://www.R-project.org/
21. Gelman A, Rubin DB. Inference from iterative simulation using multiple sequences. *Statistical Science*. 1992;7:457-511. doi:10.1214/ss/1177011136
22. Rackauckas C, Nie Q. DifferentialEquations.jl – A performant and feature-rich ecosystem for solving differential equations in Julia. *Journal of Open Research Software*. 2017;5:15-25. doi:10.5334/jors.151
23. Dixit VK, Rackauckas C. GlobalSensitivity.jl: performant and parallel global sensitivity analysis with Julia. *J Open Source Softw*. 2022;7:4561. doi:10.21105/joss.04561
24. Melillo N, Grandoni S, Cesari N, Brogin G, Puccini P, Magni P. Inter-compound and intra-compound global sensitivity analysis of a physiological model for pulmonary absorption of inhaled compounds. *AAPS J*. 2020;22:116.
25. Zhang X-Y, Trame MN, Lesko LJ, Schmidt S. Sobol sensitivity analysis: a tool to guide the development and evaluation of systems pharmacology models. *CPT Pharmacometrics Syst Pharmacol*. 2015;4:69-79.
26. Jones DC, Arthur B, Nagy T, et al. Giovineitalia/Gadfly.Jl: V0.7.0. *Zenodo*. 2018. doi:10.5281/ZENODO.1284282
27. Platova AI. Population pharmacokinetics analysis in Lixoft Monolix softwares. *Pharmacokinet Pharmacodyn*. 2022;3:36-51. doi:10.37489/2587-7836-2021-3-36-51
28. Rackauckas C, Ma Y, Noack A, et al. Accelerated predictive healthcare analytics with pumas, a high performance pharmaceutical modeling and simulation platform. *bioRxiv*. 2022. doi:10.1101/2020.11.28.402297
29. Bauer RJ. NONMEM tutorial part II: estimation methods and advanced examples. *CPT Pharmacometrics Syst Pharmacol*. 2019;8:538-556. doi:10.1002/psp4.12422

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article: Elmokadem A, Zhang Y, Knab T, Jordie E, Gillespie WR. Bayesian PBPK modeling using R/Stan/Torsten and Julia/SciML/Turing.Jl. *CPT Pharmacometrics Syst Pharmacol*. 2023;12:300-310. doi:10.1002/psp4.12926