

## TUTORIAL

# A Tutorial on RxODE: Simulating Differential Equation Pharmacometric Models in R

W Wang<sup>1</sup>, KM Hallow<sup>2\*</sup> and DA James<sup>1</sup>

This tutorial presents the application of an R package, RxODE, that facilitates quick, efficient simulations of ordinary differential equation models completely within R. Its application is illustrated through simulation of design decision effects on an adaptive dosing regimen. The package provides an efficient, versatile way to specify dosing scenarios and to perform simulation with variability with minimal custom coding. Models can be directly translated to Rshiny applications to facilitate interactive, real-time evaluation/iteration on simulation scenarios.

CPT Pharmacometrics Syst. Pharmacol. (2016) 5, 3–10; doi:10.1002/psp4.12052; published online 19 December 2015.

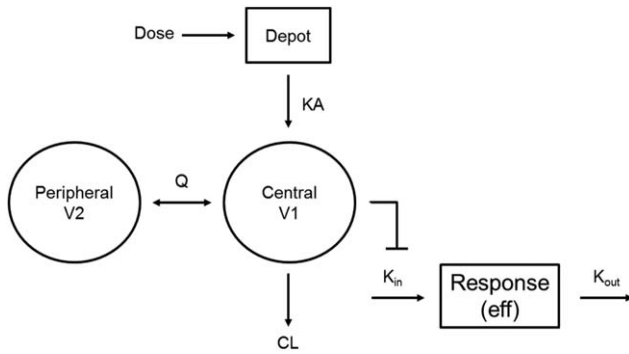
The use of simulations for drug development has been shown to be a cost-effective approach for the exploration of multiple dosing regimens and their likely pharmacodynamics effects over diverse patient populations.<sup>1–4</sup> For instance, simulations provide a means to assess the effects of various loading and maintenance dosing parameters on steady-state concentrations; effects of dosing holidays on pharmacodynamics response; variation across patients in drug exposure and/or response, etc. However, there are several factors that hinder greater utilization of pharmacometric simulation in drug development. It is difficult to identify *a priori* all possible simulation scenarios of interest, and thus oftentimes building the set of simulations that fully addresses the questions at hand is an iterative, collaborative process between the modeler and the endusers of the simulations—usually the clinical team. A first set of simulation results are typically presented through static graphics, and typically leads to further questions and other scenarios to evaluate. However, performing simulations with most currently available simulation tools is cumbersome, tedious, and time-consuming, requiring extensive custom programming and moving between one software application to perform simulations and another application to visualize simulations. This iterative process often involves multiple meetings/discussions, with significant lag time between each, and can lead to loss of momentum and lost opportunities for making quantitatively driven decisions. Thus, there is a great need for more efficient simulation processes that facilitate interactive, real-time evaluation and iteration on simulation scenarios.

The R software environment<sup>5</sup> has an excellent set of tools for analyzing and visualizing simulation results (e.g., lattice,<sup>6</sup> ggplot2<sup>7</sup> packages). In addition, with the recent release of the Shiny package,<sup>8</sup> Web-based interfaces to R programs can be easily generated. Thus, R provides an ideal environment in which to perform pharmacometric simulations in real time. However, traditionally, R has lacked extensive facilities to support modeling based on differential equations (DE) like the ones used in pharmacokinetic/pharmacodynamic (PK/PD) applications, although the R

package deSolve<sup>9</sup> now provides many general-purpose DE solvers. However, using deSolve for pharmacometric simulation is still not ideal, requiring extensive custom programming to facilitate the specification of dosing regimens and sampling schedules, especially for more complex dosing regimens. Furthermore, the convenience of specifying differential equations in the R language presents run-time limitations for deSolve when simulating large models or performing a very large number of runs (users can hard-code in C or Fortran their deSolve models to increase run-time performance, but at the expense of additional low-level, error-prone programming). Recently, efforts have increased to develop tools to address these problems and facilitate efficient simulation in R, including the PKPDsim<sup>10</sup> and Simulx<sup>11</sup> packages. In this tutorial we present a new R package, RxODE, that facilitates quick and efficient simulations of ordinary differential equation (ODE) models in R.

RxODE provides an elegant, efficient, and versatile way to specify dosing scenarios, including multiple routes of administrations within a single regimen, sampling schedules, etc. It also enables simulations with between-patient variability, and minimizes the amount of custom coding required for pharmacometric simulations. An RxODE model is automatically translated into C, compiled into machine code, and loaded into the running R program. This allows for very fast execution times, relative to deSOLVE, and the advantage in execution time increases as the number of ODEs increases. For example, RxODE is 8–10 times faster for a model with four ODEs, but 100 times faster for a model with seven ODEs (see **Supplementary Material** for runtime comparisons). Although similar run times may be achieved with deSolve by writing the model in a low-level programming language like C or Fortran and loading it dynamically, RxODE eliminates the need for this additional programming step and knowledge of a second programming language. It is designed with pharmacometric models in mind, but can be applied more generally to any ODE-based model. It also provides a function for directly generating R Shiny applications, useful for interactively probing the model; this Web-based application can then be further

<sup>1</sup>Novartis Pharmaceuticals, East Hanover, New Jersey, USA; <sup>2</sup>University of Georgia, Athens, Georgia, USA. \*Correspondence: KM Hallow ([hallowkm@uga.edu](mailto:hallowkm@uga.edu))  
Received 28 August 2015; accepted 15 November 2015; published online on 19 December 2015. doi:10.1002/psp4.12052



**Figure 1** A two-compartment pharmacokinetic model with an indirect response pharmacodynamic model.

customized by the user and used to facilitate interactive simulations. Like R, RxODE is an open source application available in all computer platforms on which R runs.

To illustrate the application of RxODE, we will present an example using simulations to evaluate the impact of various design decisions on an adaptive dosing regimen. We will first illustrate the workflow for conducting simulations in RxODE, including specifying the model structure, dosing and sampling scheme, and incorporating variability. We will show how RxODE can be used to simulate different adaptive dosing decision rules in individuals and in populations. Lastly, we will demonstrate how RxODE simulations can be linked with R Shiny to generate a user-interface that facilitates real-time interactions and scenario evaluations with clinical teams.

### CASE STUDY: EVALUATING ADAPTIVE DOSING REGIMENS

For drugs that operate in a narrow therapeutic range, it can be desirable or even necessary to use pharmacodynamic measures to adjust the dose to achieve an appropriate level of response. For example, anticoagulants such as warfarin act by reducing the ability of the blood to form clots. But too little clotting can result in excessive bleeding. Thus, pharmacodynamic measurement of blood clotting ability is used to adjust the dose within a specified range.<sup>12</sup>

Establishing a dose adjustment algorithm requires quite a few decisions. What endpoint will be used to make dose adjustment decisions? What is the target exposure range to maintain this endpoint within? Should trough, peak, or both levels be controlled? How often should measurements and dose adjustments be made? What should be the starting dose? Will monitoring and adaptive dosing continue indefinitely, or only during an initial period after treatment initiation? Simulations of these scenarios can be quite helpful in understanding the impact of these decisions.

In this tutorial we will consider a case study for adaptive dosing. For illustration purposes, we have chosen a system where the PK is described by a two-compartment model, and the PD effect is modeled with an indirect response<sup>13</sup> (**Figure 1**). In order to maintain efficacy while avoiding adverse side effects, it has been determined that inhibition of the target should be in the range of 40–60%.

### WORKFLOW FOR PERFORMING SIMULATIONS IN RxODE

First, we will demonstrate how to set up and run a simulation using RxODE. The RxODE package can be installed from Github at <https://github.com/hallowkm/RxODE> (see the **Supplementary Material** for installation instructions). RxODE is made available as open source under the GNU General Public License version 2 or later.

**Figure 2** gives an overview of the workflow for performing a simple simulation in RxODE with the model described above. The model structure is specified through a text string of equations. Both differential and algebraic equations are permitted. Differential equations are specified by “d/dt(var\_name) =”. Each equation is separated by a semicolon. All referenced, undefined quantities are assumed to be input parameters.

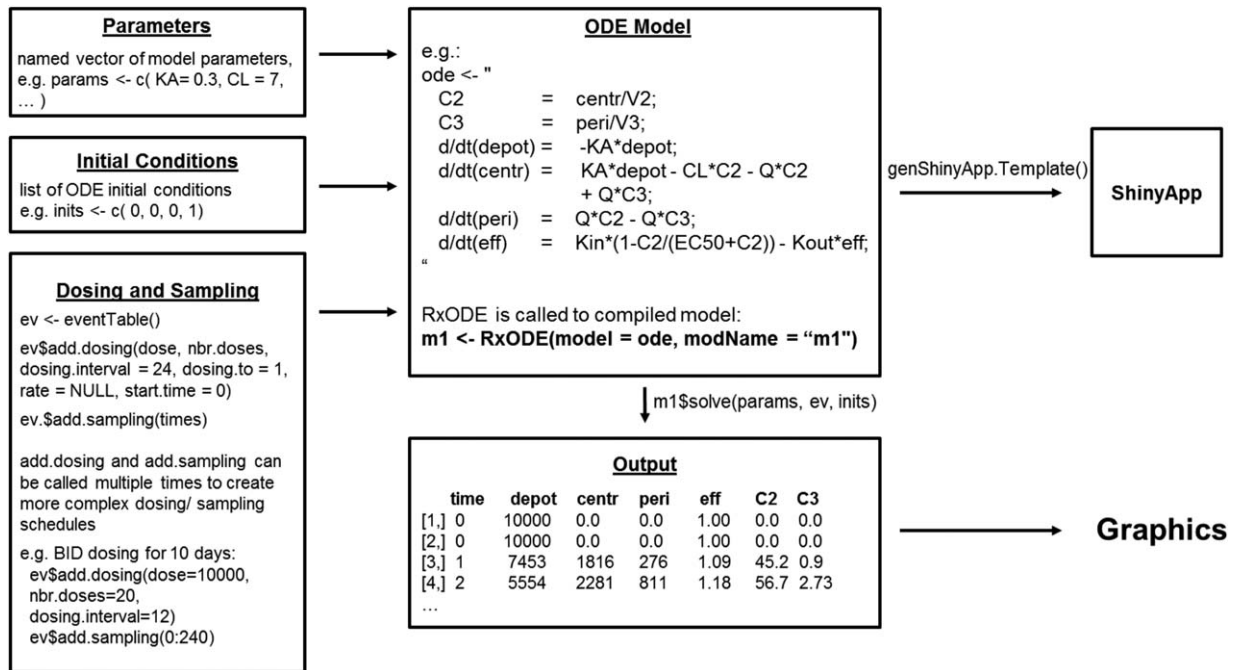
```
#Define model
ode <- "
  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(depot) = -KA*depot;
  d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
  d/dt(peri) = Q*C2 - Q*C3;
  d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
"
```

Model parameters are defined in a named vector. Names of parameters in the vector must be a superset of parameters in the ODE model, and the order of parameters within the vector is not important. Initial conditions (ICs) are defined through a vector as well. The number of ICs must equal exactly the number of ODEs in the model, and the order must be the same as the order in which the ODEs are listed in the model.

```
# Define parameters and initial conditions
params <- c(KA = 0.3, CL = 7, V2 = 40, Q = 10, V3 = 300,
  Kin = 0.2, Kout = 0.2, EC50 = 8)
inits <- c(0, 0, 0, 1)
```

The creation of an `eventTable()` object provides an extremely efficient and flexible way to specify dosing and sampling. **Table 1** shows examples for generating a variety of dosing schedules. An `eventTable` is generated using the `RxODE` function `eventTable()`. The generated `eventTable` object has functions that allow easy addition of dosing and sampling events. The `add.dosing()` function allows specification of the dose amount, number of doses, dosing interval, compartment to dose into, rate (if an infusion), and dosing start time. More complex dosing schedules can be simulated by applying the `add.dosing()` function multiple times. The `add.sampling()` function allows specification of the time points to be included in the simulation output.

Calling the `RxODE()` function, the model is translated into C code, compiled, and dynamically loaded into the running R process. A simulation can then be performed by calling the R model object's function `run()`, with the specified parameter vector, initial conditions vector, and event table as inputs.



**Figure 2** Workflow for performing a simulation with RxODE.

```

# Compile model
mod1 <- RxODE(model = ode, modName = "mod1")
# Run simulation
x <- mod1$run(params, ev, inits)

```

All state variables as well as other variables computed in the model are returned in the output matrix, at the times specified in the eventTable. Thus, the simulation results are readily available for performing calculations and generating plots in R using any of the existing R packages (lattice, ggplot, etc).

The user can also choose to specify the absolute and/or relative tolerance, as well as the type of solver to be used:

```

X <- m1$run(theta, ev, inits, stiff = F, atol = 1e-8,
rtol = 1e-6)

```

RxODE uses the LSODA and a Runge-Kutta integrators for stiff and non-stiff equations, respectively. The LSODA (Livermore Solver for Ordinary Differential Equations) Fortran package is an automatic method switching for stiff and non-stiff problems throughout the integration interval. For purely non-stiff systems, RxODE uses DOP853, an explicit Runge-Kutta method of order 8(5,3).

### SIMULATING WITH VARIABILITY

RxODE provides a straightforward way to perform simulations that incorporate parameter variability and uncertainty. A matrix of parameter values can be generated, where each row represents one set of parameter values. R sampling functions such as rnorm and mvrnorm can be utilized to build this matrix, but decisions on the level(s) of uncertainty to include are left to the discretion of the modeler, since this depends on the spe-

cific questions a given simulation is designed to address. The following code generates parameters for 100 subjects with correlated interindividual variability on CL and V2.

```

nsub <- 100# 100 subproblems
sigma <- matrix(c(0.09,0.08,0.08,0.25),2,2) # IIV covari-
ance matrix
mv <- mvrnorm(n=nsub, rep(0,2), sigma) # Sample
from covariance matrix
CL <- 7*exp(mv[,1])
V2 <- 40*exp(mv[,2])
params.all <- cbind(KA=0.3, CL=CL, V2=V2, Q=10,
V3=300, Kin=0.2, Kout=0.2, EC50=8)

```

Once this parameter matrix is generated, each subproblem can be simulated by looping through the parameter matrix, using each row as an input for the simulation, and collecting the output of each simulation in an output matrix.

```

res <- NULL
# Loop through each row of parameter values and
simulation
for (i in 1:nsub) {
  params <- params.all[i,]
  x <- mod1$solve(params, ev, inits = inits)
  #Store results for effect compartment
  res <- cbind(res, x[, "eff"])
}

```

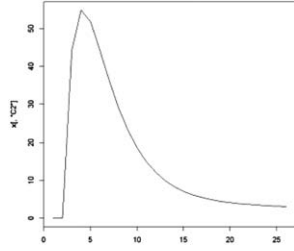
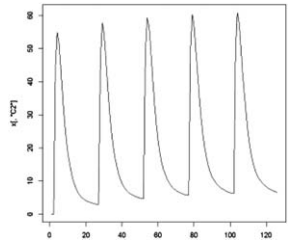
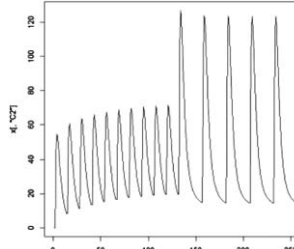
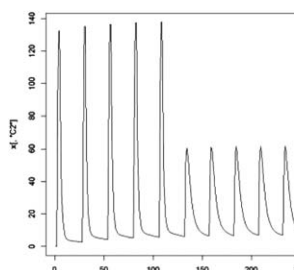
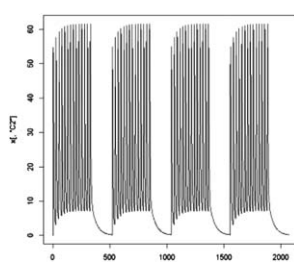
The same result can be achieved more efficiently with the following code:

```

res <- apply(theta.all, 1, function(theta) mod$run(theta,
ev, inits)[, "eff"])

```

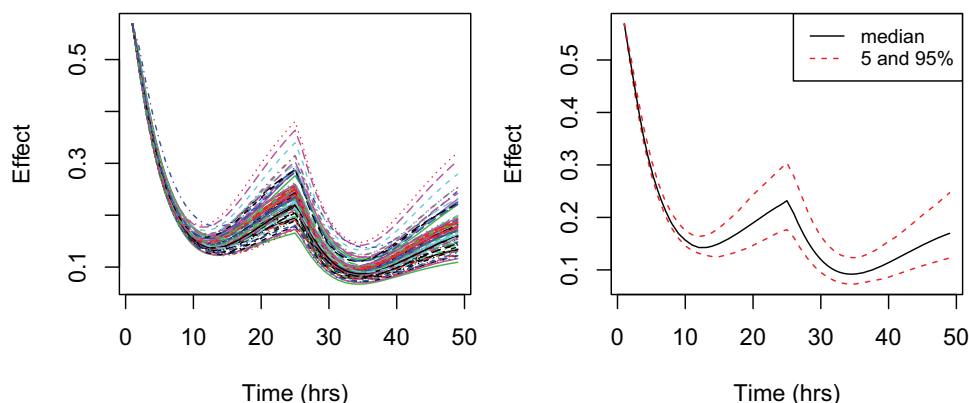
**Table 1** The `add.dosing()` function provides an efficient method to specify a variety of dosing schedules

Single dose	<code>ev\$add.dosing(dose=10000, nbr.doses=1)</code>	
Multiple doses	<code>ev\$add.dosing(dose=10000, nbr.doses=5, dosing.interval=24)</code>	
Bid for 5 days, followed by qd for 5 days	<pre>ev\$add.dosing(dose=10000, nbr.doses=10, dosing.interval=12) ev\$add.dosing(dose=20000, nbr.doses=5, dosing.interval=24, start.time=120)</pre>	
Infusion for 5 days, followed by oral for 5 days	<pre>ev\$add.dosing(   dose=10000,   dose=10000,   nbr.doses=5,   dosing.to=2,   rate=5000 ) ev\$add.dosing(dose=10000,   nbr.doses=5, start.time=120)</pre>	
2wk-on, 1wk-off	<pre>for (i in 1:ncyc)   ev\$add.dosing(dose=10000,   nbr.doses=14, start.time=(i-1)*21*24)</pre>	

Simulation results can be then be directly analyzed and visualized using any of the statistical and graphics tools available within R. **Figure 3** shows the results of the above simulation when the drug is given QD for 2 days. The full script for this simulation is available in the **Supplemental Material**.

### SIMULATING ADAPTIVE DOSING IN A TYPICAL PATIENT

To return to the problem of simulating adaptive dosing, there are many factors that must be considered in a designing an adaptive dosing scheme. Utilizing RxODE



**Figure 3** Simulation of effect for 2 days of QD dosing, correlated interindividual variability on CL and V2. (a) Individual effect profiles. (b) Median, 5th, and 95th percentile effect profile. Since simulations are conducted in R, RxODE simulation results can easily be summarized and visualized graphically.

within the R environment, simulations of the impact of various factors can be quickly performed and evaluated.

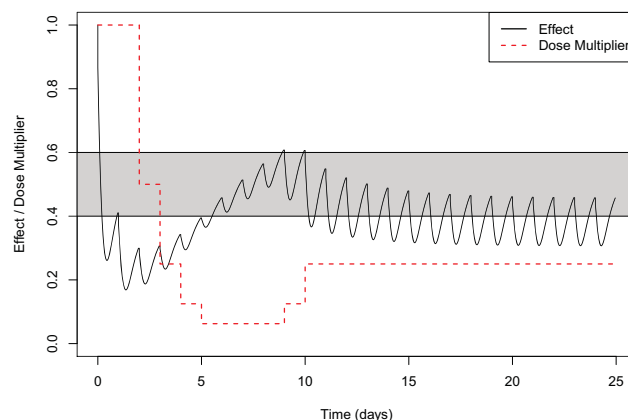
In order to simulate an adaptive dosing scenario, a decision rule must be specified. We will first simulate the following decision rule: The drug is to be dosed once daily, and trough PD effect levels will be measured 24 hours after each dose. The target range for the effect is 40–60% inhibition. If the measured PD effect is less than 40%, the dose will be doubled. If the measured PD effect is greater than 60%, the dose will be cut in half. If the PD effect is between 40 and 60%, no change will be made.

The following R code is used to perform this simulation over 25 days. In brief, parameters governing the simulation are defined, including the decision rule limits and dose adjustments, number of days, starting dose, and sampling frequency. Then treatment is simulated one day at a time, and after each day the simulated trough level at the end of that day is used to determine the dose level for the next simulated day. This is repeated for the number of days specified, and the results are stored in a matrix. Results contained in this matrix can then be plotted using any available R plotting tools.

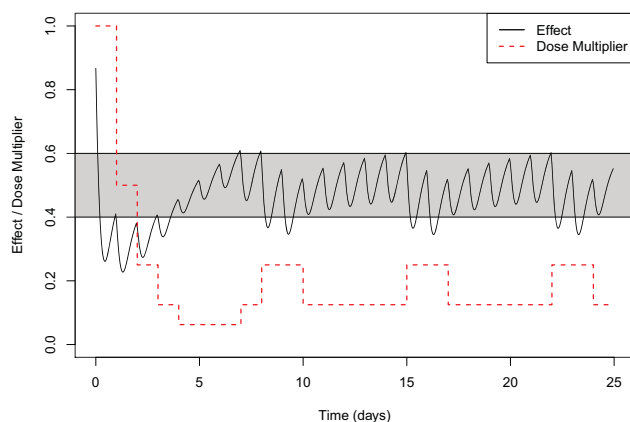
```
effect.limits = c(0, 0.4, 0.6, 9) # Decision rule limits
dose.multipliers = c(0.5, 1, 2) # Decision rule effects
ndays <- 25; unit.dose <- 10000; start.dose <- 1; sampling.frequency <- 1 # Sample every day
# Simulate each day. At the end of each day, test the effect level, and adjust the dose level according to the decision rule
for (i in seq(1, ndays, by = sampling.frequency)) {
  if (i==1) {# Initialize on first day
    inits <- c(0, 0, 0, 1)
    last.multiplier <- start.dose
    this.multiplier <- 1
  } else {# Use end of previous day as initial conditions
    for next day, compare trough effect with
    #decision rule limits and determine dose multiplier accordingly
    inits <- x[dim(x)[1], vars]
    wh <- cut(inits["eff"], effect.limits)
    this.multiplier <- dose.multipliers[wh]
  }
}
```

```
}
this.multiplier <- this.multiplier*last.multiplier # Adjust dose
last.multiplier <- this.multiplier # Store new dose
#specify dosing and sampling
ev <- eventTable()
ev$add.dosing(dose = this.multiplier*unit.dose, dosing.interval = 24, nbr.doses = sampling.frequency)
ev$add.sampling(0:(24*sampling.frequency))
x <- mod1$run(params, ev, inits) # Run simulation
# Compile outputs
time.total <- ev$get.EventTable()[,"time"]+(i-1)*(24)
doses <- rep(last.multiplier, length(time))
x <- cbind(x, time.total, doses);
res <- rbind(res, x)
}
```

**Figure 4** shows the results of this simulation. After 10 days, a steady state dose that is 25% of the starting dose



**Figure 4** Simulated adaptive dosing to maintain trough pharmacodynamic inhibition within 40–60%. The solid line shows the effect of the drug over time. The red dashed line shows the dose multiplier over time (actual dose is the dose multiplier times the starting dose). The gray region represents the target range for the trough effect. A stable dose that maintains the trough effect within the target range is reached after 10 days.



**Figure 5** Simulated adaptive dosing to maintaining both trough and peak effect within the target range of 40–60%. The solid line shows the effect of the drug over time. The red dashed line shows the dose multiplier over time (actual dose is the dose multiplier times the starting dose). The gray region represents the target range for both trough and peak effect. A pattern emerges, indicating that dosing 12.5% of the starting dose, but then giving a double dose every 5 days, would achieve this goal.

is reached, which maintains the trough levels within the desired range. However, during the first 5 days the level of inhibition is well beyond the desired target.

Alternative decision rules can easily be evaluated. For instance, **Figure 5** shows the simulation results if both trough and peak levels are controlled within the 40–60% range (i.e., biomarker measurements are taken at  $t_{\max}$  of 12 hours and at trough, and dose is adjusted daily). This full script for performing this simulation can be found in the **Appendix**, but it requires only the addition of the following line to the script above:

```
# If effect at 12 hours is less than 0.4, cut dose in half
if (x[13, "eff"] < effect.limits[2]) {
  this.multiplier <- dose.multipliers[1]
}
```

In this case, a stable dose is not reached. Instead, a more complex pattern emerges, suggesting that dosing 12.5% of the starting dose, but then giving a double dose every 5 days, could achieve the desired result.

### SIMULATING ADAPTIVE DOSING WITH VARIABILITY

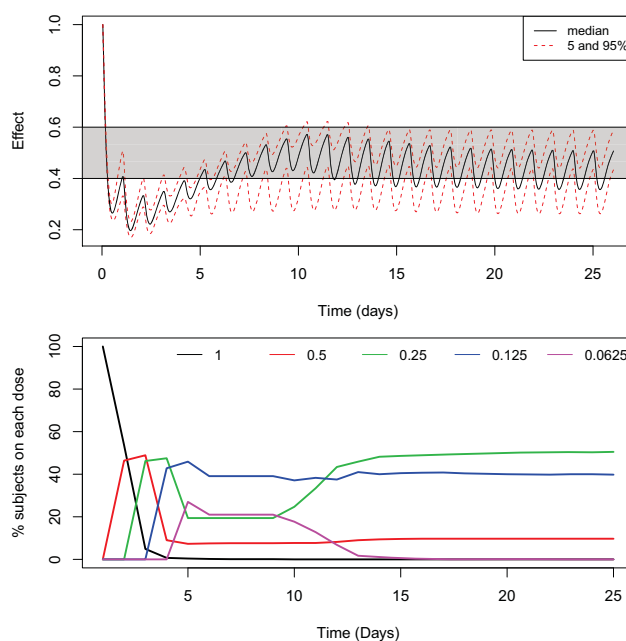
It may also be of interest to explore the impact of PK and PD variability within the population on the resulting dose trajectory and final dose. We can simulate variability, as described above, by specifying a matrix of parameter sets, rather than a fixed set of parameters, and looping through this parameter matrix, performing the adaptive simulation as described above for each set of parameters. This is easily done by adapting the simulation of a single subject to contain an outer for-loop that cycles through the parameter matrix. Code for this simulation is available in the

**Supplementary Material.** Again, we assume correlated interindividual variability on CL and V2, and simulate 100 subjects for 25 days. The resulting plots are shown in **Figure 6**. Trough levels for all subjects are controlled within the specified range by the end of the time period. 62% of subjects end at 25% of the starting dose, 35% require only 12.5%, and a few subjects (2.5%) require a higher dose (50% of starting dose). It took at least 20 days to reach a final dose in all subjects (compared to 10 days for the typical subject in the previous simulation).

### USING SIMULATIONS TO ADDRESS QUESTIONS IN REAL TIME

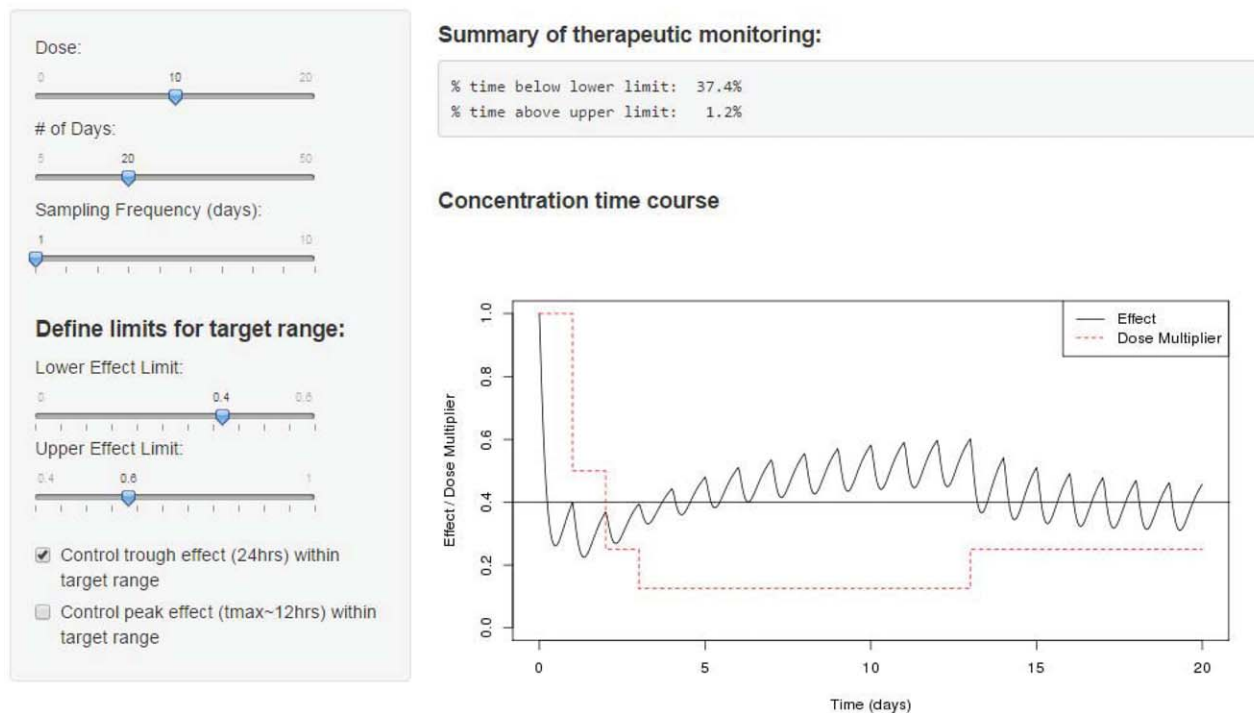
The simulations performed thus far are static, in that fixed parameter inputs are defined, simulations are performed, and plots are generated. Often, when the results of such simulations are reviewed, further questions arise—what happens if the range limits for our decision rule are relaxed or tightened? What happens if we change the starting dose? What happens if we sample every 2nd or 3rd day, rather than every day?

These new questions usually require the modeler to go off and perform new simulations, and the team must reconvene days or weeks later to discuss new results. This introduces significant lag time, especially since several iterations may be required to reach a final set of simulations. Although it may be possible to compile a massive document containing all the permutations of scenarios of interest *a priori*, sifting through this document with team members in real time can



**Figure 6** Simulation of adaptive dosing regimen in 1,000 subjects, with correlated interindividual variability on CL and V2. (a) The adaptive dosing regimen successfully controls effect levels within the desired range after 5 days. (b) The dose trajectory and final dose varies among the population, and most subjects end on either 25% or 12.5% of the starting dose.

## Therapeutic monitoring simulator



**Figure 7** RxODE provides a function for generating interactive Shiny Apps, which can then be customized. This app allows users to vary the dose, number of days, sampling frequency, and decision rules, and to view simulation results in real time.

be a daunting task, and is not advisable, as it can lead to confusion and/or loss of attention of key stakeholders. A better alternative is to encase the simulation procedure in an interactive application in which a wide range of scenarios can be evaluated in real time, as they arise.

A major advantage of performing simulations within the R environment is the ability to take advantage of R's Shiny package for developing interactive Web applications. These Web applications can be made available to the team online, and used in meetings to explore simulation scenarios in real time. This is extremely advantageous in facilitating modeler-team interactions, because it can greatly reduce the number of iterations, separate meetings, and associated lag time.

Shiny apps are easy to write, requiring no Web development skills and very limited programming skills. RxODE models can be easily linked with rShiny. The package includes a function for generating a shinyApp template from a simple model. The user can then adapt this template for their specific model.

```
genShinyApp.Template(appDir = "shinyExample",
  verbose = TRUE)
library(shiny) #Load the shiny package
runApp("shinyExample") #Run the example app
```

The function `genShinyApp.Template()` generates a folder that contains the R Shiny `ui.R` and `server.R` files for the template app, as well as an RDA file containing the saved exam-

ple model, parameters, and initial conditions. To incorporate a different R model, the user needs can save the new model to an RDA file and load this new file from the `server.ui` file.

```
# Save the model, parameters, init values, etc. in the file
rx_shiny_data.rda to be loaded by the server.R
save(mod1, params, inits, stiff = TRUE, atol = 1e8,
  rtol = 1e6, file = "rx_shiny_data.rda")
```

To tailor the app, the only two files that need to be altered by the user are the `ui.R` and `server.R` files. The `ui.R` file can be edited to add widgets for obtaining user input and to control the layout of the user interface. The `server.R` file can be edited to control how inputs are used and how outputs are displayed. There is excellent documentation on developing Shiny App interfaces available online<sup>8</sup> as well as a previous tutorial on this topic in this journal.<sup>14</sup>

**Figure 7** shows the shiny app interface for exploring different factors in an adaptive dosing regimen, including the starting dose, the number of days of simulation, the frequency at which the dose is adjusted, the lower and upper limits of the range for the decision rule, and whether trough, peak, or both levels are controlled by the decision rule. It displays the calculated % time above and below the decision rule limits, and it plots the effect and dose as a function of time. As the user moves the sliders or checks the check boxes, the outputs are adjusted interactively. The `ui.R`, `server.R`, `RxODE.run.R` files and the compiled C

model are available in the **Supplementary Material**. The interactive app can also be accessed at <http://qsp.engr.uga.edu:3838/adaptiveDosing>.

Thus, starting within a baseline scenario, many different scenarios and combinations of scenarios can be quickly explored. While the modeler will likely facilitate evaluation of scenarios, these apps require no technical expertise or experience with R, and thus the apps are accessible to nontechnical partners. Also, importantly, Shiny apps can be easily deployed online and do not require local installation or R. This is in contrast to other software such as Berkeley Madonna. While Berkeley Madonna can facilitate interactive model exploration, it also requires purchase of a software license, local installation of the software, and some user familiarity with the software and modeling.

## CONCLUSION

RxODE is an R package that provides tools for the efficient simulation of complex dosing regimens via PK/PD models described by ODEs. It provides great flexibility and speed in performing simulations with variability and uncertainty. As part of the R environment, RxODE outputs can be combined with a multitude of R facilities to create advanced static and interactive visualization displays for effective communications with clinical team members and other consumers. Furthermore, unlike many other simulation tools, simulation and preparation of graphics can be conducted completely within a single, freely available, and open-source software. No licenses are required, and it does not require linking with any external software. Although currently focused on efficient simulations, RxODE can also be used for parameter estimation through the many existing statistical estimation algorithms in R, including nonlinear mixed effects models,<sup>14</sup> stochastic approximation expectation-maximization (SAEM),<sup>16</sup> and Bayesian methods using Gibbs sampling, e.g., JAGS.<sup>17</sup> Future work includes developing functionality to aid users in linking RxODE models with these estimation algorithms in a more efficient manner.

**Acknowledgments.** We thank Jerry Nedelman, Kalundayan Subramanian, Varun Goel, and Abhijit Chakraborty.

**Conflicts of Interest.** The authors declare no conflicts of interest.

**Author Contributions.** W. W.: Developed and tested RxODE package, collaborated in development of adaptive dosing application; K.M.H.:

Aided in development of RxODE package, tested RxODE package, wrote article, collaborated in development of adaptive dosing application; D.A.J.: Developed and tested RxODE package, collaborated in development of adaptive dosing application

1. Huang, X. & Li, J. Pharmacometrics: The science of quantitative pharmacology. *Am. J. Pharm. Educ.* **71**, 75 (2007).
2. Ette, E., Godfrey, C., Ogenstad, S. & Williams, P. Analysis of simulated clinical trials. In *Simulation for Designing Clinical Trials: A Pharmacokinetic/Pharmacodynamic Modeling Perspective* (eds. Kimko, H. & Duffull, S.) (Marcel Dekker, New York, 2003).
3. Riggs, M.M., Godfrey, C.J. & Gastonguay, M.R. Clinical trial simulation: efficacy trial. In *Pharmacometrics: The Science of Quantitative Pharmacology* (eds. Ette, E.I. & Williams, P.J.) 881–900 (John Wiley & Sons, Hoboken, NJ, 2007).
4. Holford, N., Ma, S.C. & Ploeger, B.A. Clinical trial simulation: a review. *Clin. Pharmacol. Ther.* **88**, 166–182 (2010).
5. R Core Team. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. [Software] (Version 3.1.1). <<http://www.R-project.org/>> (2014).
6. Sarkar, D. *Lattice: Multivariate Data Visualization with R* (Springer, New York, 2008).
7. Wickham, H. *ggplot2: Elegant Graphics for Data Analysis* (Springer, New York, 2009).
8. RStudio and Inc. shiny: Web Application Framework for R. R package version 0.10.1. <<http://CRAN.R-project.org/package=shiny>> (2014).
9. Soetaert, K., Petzoldt, T. & Setzer, R.W. Solving differential equations in R: package deSolve. *J. Stat. Softw.* **33**, 1–25. <<http://www.jstatsoft.org/v33/i09/>> (2010).
10. Keizer, R. PKPDsim. GitHub repository. <<https://github.com/ronkeizer/PKPDsim>> (2015).
11. Inria POPIX team. Simulx: A R function of the mlxR package for computing predictions and sampling longitudinal data from Mlxtran and PharmML models. <<http://simulx.webpopix.org/>> (2015).
12. Kuruvilla, M. & Gurk-Turner, C. A review of warfarin dosing and monitoring. *Proc. (Bayl. Univ. Med. Cent.)* **14**, 305–306 (2001).
13. Sharma, A. & Jusko, W.J. Characteristics of indirect pharmacodynamic models and applications to clinical drug responses. *Br. J. Clin. Pharmacol.* **45**, 229–239 (1998).
14. Wojciechowski, J., Hopkins, A.M. & Upton, R.N. Interactive pharmacometric applications using R and the shiny package. *CPT Pharmacometrics Syst. Pharmacol.* **4**, 146–159 (2015).
15. Pinheiro, J., Bates, D., DebRoy, S., Sarkar, D. and R Core Team. *nlme: Linear and Nonlinear Mixed Effects Models*. R package version 3.1-117. <<http://CRAN.R-project.org/package=nlme>> (2014).
16. Comets, E., Lavenue, A. & Lavielle, M. SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstract 2173. <<http://www.page-meeting.org/default.asp?abstract=2173>> (2011).
17. Plummer, M. rjags: Bayesian Graphical Models using MCMC. R package version 3-15. <<http://CRAN.R-project.org/package=rjags>> (2015).

© 2015 The Authors CPT: Pharmacometrics & Systems Pharmacology published by Wiley Periodicals, Inc. on behalf of American Society for Clinical Pharmacology and Therapeutics. This is an open access article under the terms of the Creative Commons Attribution-NonCommercial License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.

Supplementary information accompanies this paper on the *CPT: Pharmacometrics & Systems Pharmacology* website (<http://www.wileyonlinelibrary.com/psp4>)