RxODE package for R: a tool for performing simulations with Ordinary Different Equation models in R, with applications for pharmacometrics

---

Authors: Wenping Wang, K. Melissa Hallow, David James

Introduction RxODE is an R package that facilitates simulation with ODE models in R. It is designed with pharmacometric models in mind, but can be applied more generally to any ODE model. Here, a typical pharmacokinetic-pharmacodynamic (PKPD) model is used to illustrate the application of RxODE. This model is illustrated in Figure 1. It is assumed that all model parameters have been estimated previously.

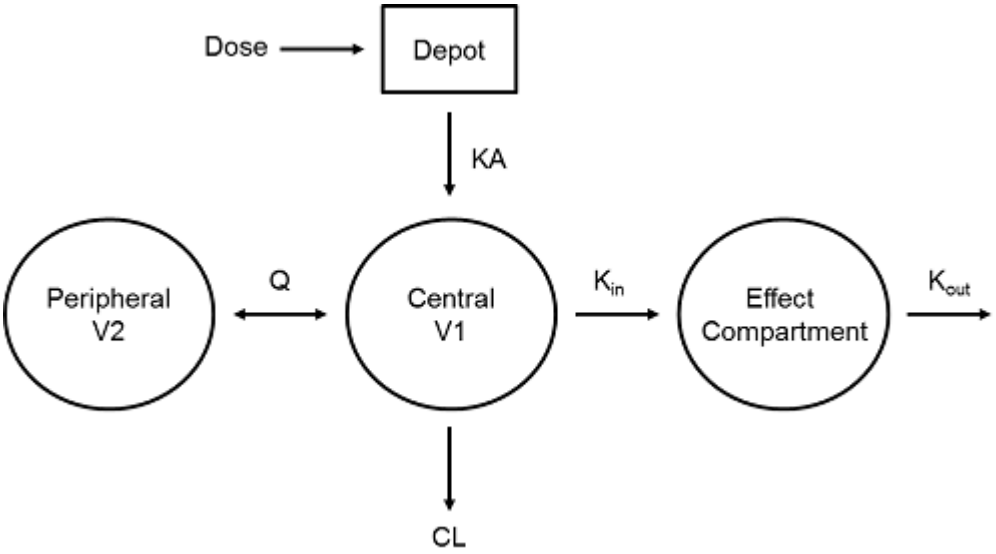


Figure 1. A two compartment pharmacokinetic model with an effect compartment

---

Description of RxODE illustrated through an example The model equations are specified through a text string in R. Both differential and algebraic equations are permitted. Differential equations are specified by "d/dt(var_name) = ...". Each equation is separated by a semicolon.

```
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;
"
```

To load RxODE package and compile the model:

```
library(RxODE2)
mod1 <- RxODE(model = ode, modName = "mod1")
```

Model parameters are defined in named vectors. 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.

```
theta <-
   c(KA=2.94E-01, CL=1.86E+01,             # central
     V2=4.02E+01, Q=1.05E+01, V3=2.97E+02, # peripheral
     Kin=1, Kout=1, EC50=200)              # effects
```

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. Elements may be names if desired:

```
inits <- c(depot=0, centr=0, peri=0, eff=1)
```

RxODE provides an simple and very flexible way to specify dosing and sampling through functions that generate an event table. First, an empty event table is generated through the "eventTable()" function: Next, the add.dosing() and add.sampling() functions of the event class are used to specify how to dose and sample. A description of inputs to add.dosing() and add.sampling() are given in Table 1. These functions can be called multiple times to specify more complex dosing or sampling regiments. Here, these functions are used to specify 10mg BID dosing for 5 days, followed by 20mg QD dosing for 5 days.

```
ev <- eventTable()
ev$add.dosing(dose=10000, nbr.doses=10, dosing.interval=12)
ev$add.dosing(dose=20000, nbr.doses=5, start.time=120)
ev$add.sampling(0:240)
```

The get.dosing() and get.sampling() functions can be used later to retrieve information from the event table.

```
ev$get.dosing()
```

```
##    time evid   amt
## 1     0  101 10000
## 2    12  101 10000
## 3    24  101 10000
## 4    36  101 10000
## 5    48  101 10000
## 6    60  101 10000
## 7    72  101 10000
## 8    84  101 10000
## 9    96  101 10000
## 10  108  101 10000
## 11  120  101 20000
## 12  144  101 20000
## 13  168  101 20000
## 14  192  101 20000
## 15  216  101 20000
```

```
head(ev$get.sampling())
```

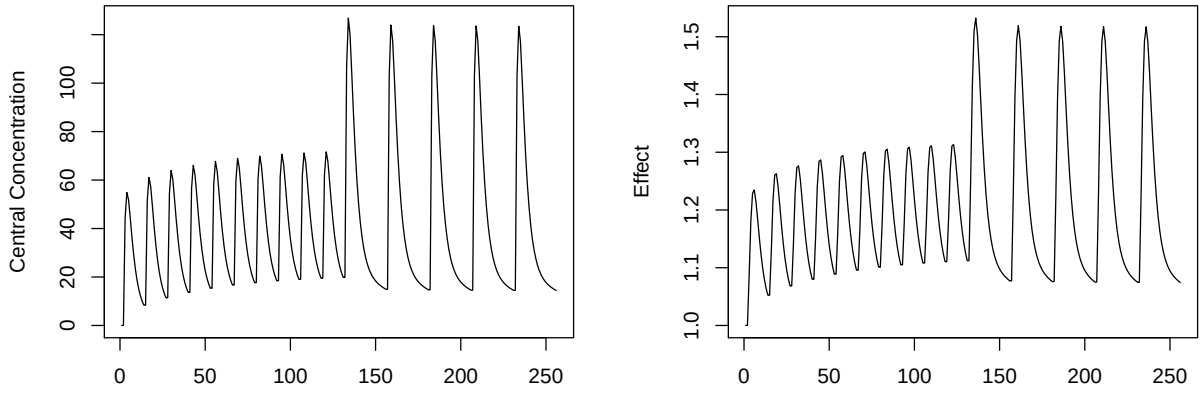
```
##    time evid amt
## 16    0    0  NA
## 17    1    0  NA
## 18    2    0  NA
## 19    3    0  NA
## 20    4    0  NA
## 21    5    0  NA
```

The simulation can now be run by calling the model object's run function. Simulation results for all variables in the model are stored in the output matrix x.

```
x <- mod1$run(theta, ev, inits)
head(x)
```

```
##      time depot centr   peri   eff    C2     C3
## [1,]    0 10000     0    0.0 1.000  0.00 0.0000
## [2,]    0 10000     0    0.0 1.000  0.00 0.0000
## [3,]    1  7453  1784  273.2 1.085 44.38 0.9198
## [4,]    2  5554  2206  793.9 1.181 54.88 2.6730
## [5,]    3  4140  2087 1323.6 1.229 51.90 4.4565
## [6,]    4  3085  1789 1776.3 1.235 44.50 5.9807
```

```
par(mfrow=c(1,2))
matplot(x[,"C2"], type="l", ylab="Central Concentration")
matplot(x[,"eff"], type="l", ylab = "Effect")
```



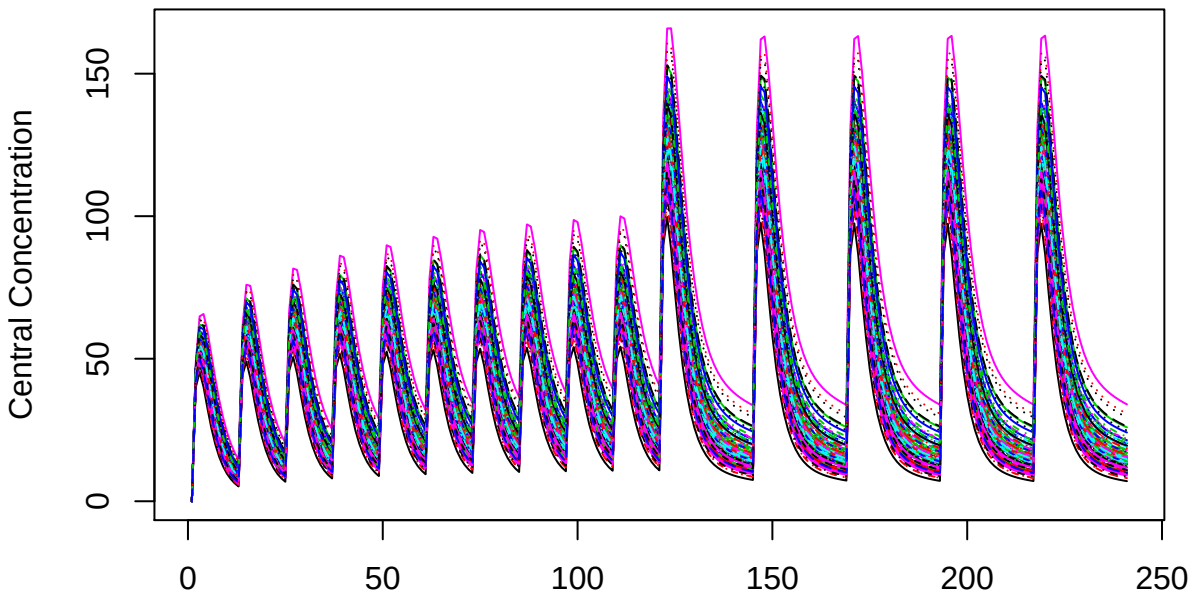
Simulation of Variability with RxODE Variability in model parameters can be simulated by creating a matrix of parameter values for use in the simulation. In the example below, 40% variability in clearance is simulated.

```
nsub=100                          #number of subproblems
CL=1.86E+01*exp(rnorm(nsub,0,.4^2))
theta.all <-
    cbind(KA=2.94E-01, CL=CL,             # central
    V2=4.02E+01, Q=1.05E+01, V3=2.97E+02, # peripheral
    Kin=1, Kout=1, EC50=200)              # effects
head(theta.all)
```

```
##         KA    CL   V2    Q  V3 Kin Kout EC50
## [1,] 0.294 27.95 40.2 10.5 297   1    1  200
## [2,] 0.294 15.97 40.2 10.5 297   1    1  200
## [3,] 0.294 23.41 40.2 10.5 297   1    1  200
## [4,] 0.294 20.04 40.2 10.5 297   1    1  200
## [5,] 0.294 19.30 40.2 10.5 297   1    1  200
## [6,] 0.294 11.04 40.2 10.5 297   1    1  200
```

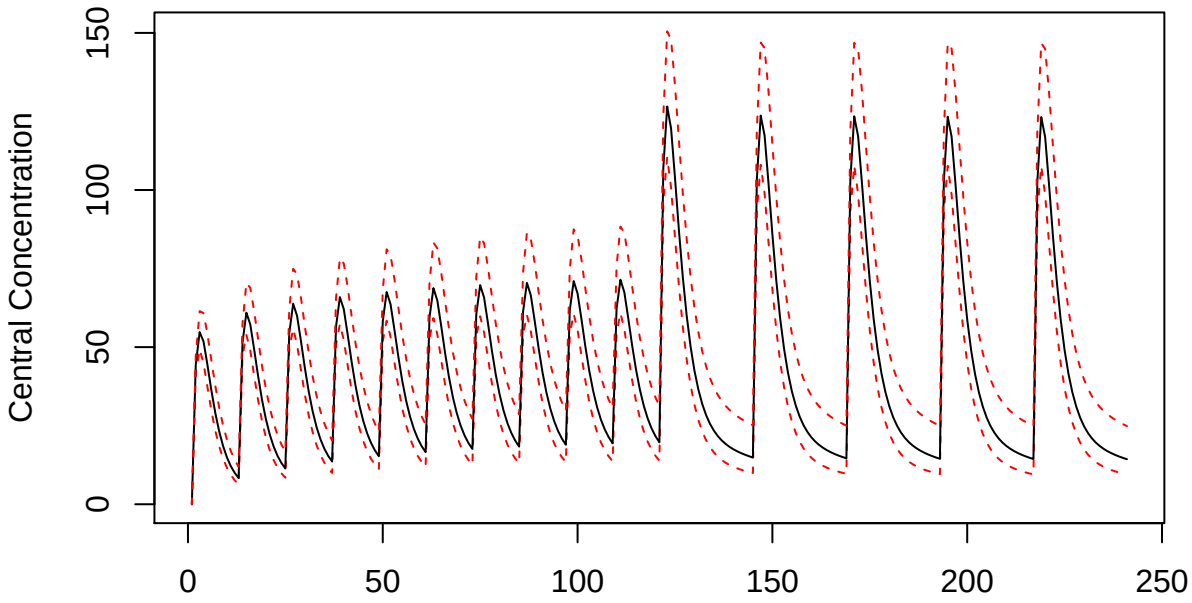
Each subproblem can be simulated by using a loop to run the simulation for each set of parameters of in the parameter matrix.

```
obs.rec = ev$get.obs.rec()
nobs = sum(obs.rec)
cp.all = matrix(NA, nobs, nsub)
for (i in 1:nsub)
{
    theta = theta.all[i,]
    x <- mod1$run(theta, ev, inits=inits)
    cp.all[, i] = x[obs.rec, "C2"]
}
matplot(cp.all, type="l", ylab="Central Concentration")
```



It is now straightforward to perform calculations and generate plots with the simulated data. Below, the 5th, 50th, and 95th percentiles of the simulated data are plotted.

```
cp.q = matrix(NA, nobs, 3)
for (i in 1:nrow(cp.all))
{
    cp.q[i, ] = quantile(cp.all[i,], prob=c(.05, .5, .95))
}
matplot(cp.q, type="l", lty=c(2,1,2), col=c(2,1,2), ylab="Central Concentration")
```



RxODE facilatates efficient generation of interactive model interfaces through Rshiny The RxODE package includes a function genShinyApp() that automatically generates an interactive shiny application for running and interacting with the model. This function creates four key files: a platform-specific compiled C model file, a script that can call and run the compiled model within the shiny app, and the ui.R and server.R files make up the shiny app. The default app includes widgets for varying the dose, dosing regimen, dose cycle, and number of cycles. The default output is a table of state variables and plot of one state variable, as shown below. The user is then free to adapt the shiny app to their needs by editing the ui.R and server.R files.

```
mod1.genShinyApp()

library("shiny")
runApp("mod1.d")
```

Click here to go to the Shiny App