RxODE: A tool for performing simulations from

Ordinary Differential Equation (ODE) models, with applications for pharmacometrics ***

Authors: Melissa Hallow and Wenping Wang

Introduction

`RxODE` is an R package that facilitates simulation with ODE models in R. It is designed with pharmacometrics models in mind, but can be applied more generally to any ODE model. Here, a typical pharmacokinetics-pharmacodynamics (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.

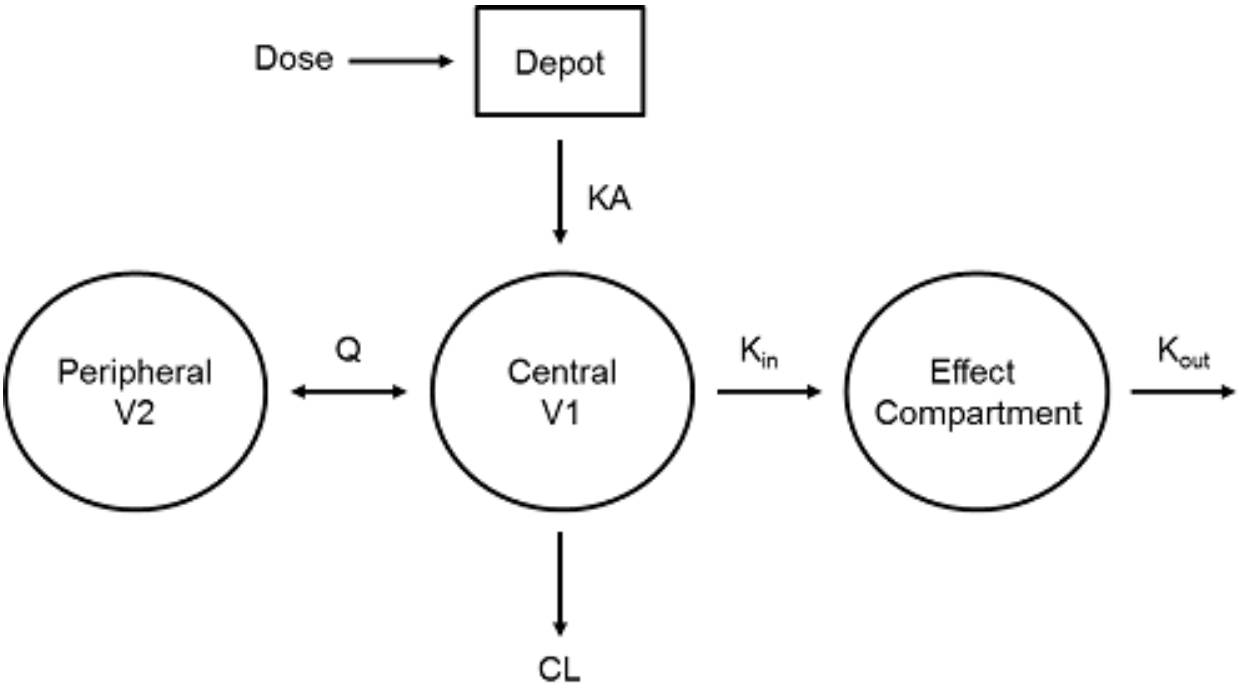


Figure 1. A two compartment pharmacokinetics model with an effect compartment

---

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(RxODE)
work <- tempfile("Rx_intro-")
mod1 <- RxODE(model = ode, modName = "mod1", wd = work)
```

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, V2=4.02E+01, # central
     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 named if desired:

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

`RxODE` provides a 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:

```
ev <- eventTable(amount.units='mg', time.units='hours')
```

Next, use the `add.dosing()` and `add.sampling()` functions of the `EventTable` object to specify the dosing (amounts, frequency and/or times, etc.) and observation times at which to sample the state of the system. 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$add.dosing(dose=10000, nbr.doses=10, dosing.interval=12)
ev$add.dosing(dose=20000, nbr.doses=5, start.time=120, dosing.interval=24)
ev$add.sampling(0:240)
```

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

```
head(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
```

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

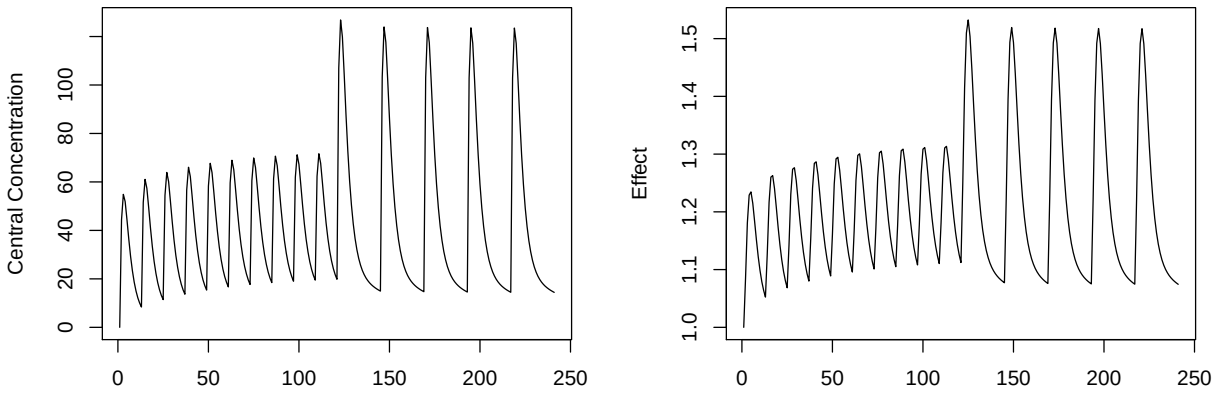
```
##    time evid amt
## 16    0    0  NA
## 17    1    0  NA
## 18    2    0  NA
## 19    3    0  NA
## 20    4    0  NA
## 21    5    0  NA
```

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$solve(theta, ev, inits)
head(x)
```

```
##      time     depot    centr      peri      eff       C2        C3
## [1,]    0 10000.000    0.000    0.0000 1.000000  0.00000 0.0000000
## [2,]    1  7452.765 1783.897  273.1895 1.084664 44.37555 0.9198298
## [3,]    2  5554.370 2206.295  793.8758 1.180825 54.88296 2.6729825
## [4,]    3  4139.542 2086.518 1323.5783 1.228914 51.90343 4.4564927
## [5,]    4  3085.103 1788.795 1776.2702 1.234610 44.49738 5.9807076
## [6,]    5  2299.255 1466.670 2131.7169 1.214742 36.48434 7.1774981
```

```
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.

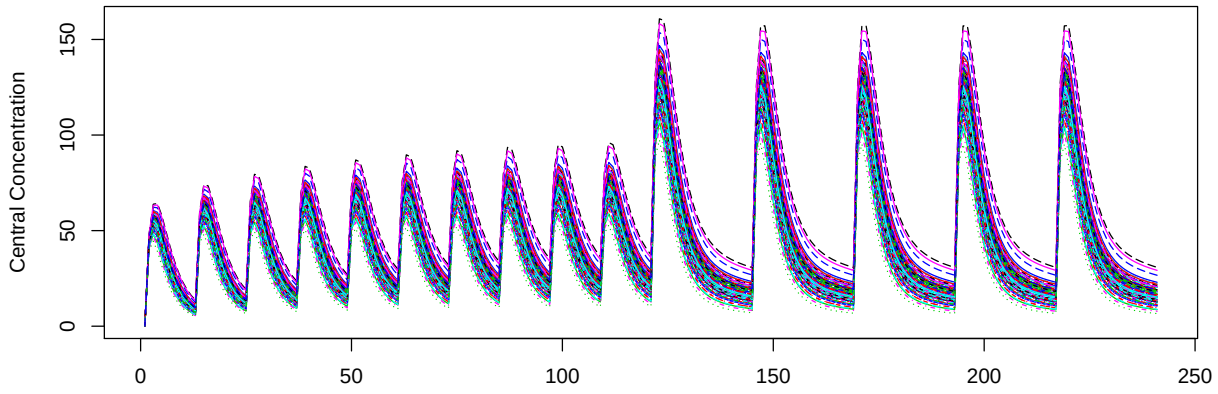
```
nsub <- 100                          #number of subproblems
CL <- 1.86E+01*exp(rnorm(nsub,0,.4^2))
theta.all <-
    cbind(KA=2.94E-01, CL=CL, V2=4.02E+01,  # central
    Q=1.05E+01, V3=2.97E+02,                # peripheral
    Kin=1, Kout=1, EC50=200)                # effects
head(theta.all)
```

```
##         KA       CL   V2    Q  V3 Kin Kout EC50
## [1,] 0.294 15.76435 40.2 10.5 297   1    1  200
## [2,] 0.294 23.50827 40.2 10.5 297   1    1  200
## [3,] 0.294 20.39916 40.2 10.5 297   1    1  200
## [4,] 0.294 17.61499 40.2 10.5 297   1    1  200
## [5,] 0.294 18.33685 40.2 10.5 297   1    1  200
## [6,] 0.294 17.19664 40.2 10.5 297   1    1  200
```

Each subproblem can be simulated by using an explicit loop (or the `apply()` function) to run the simulation for each set of parameters of in the parameter matrix.

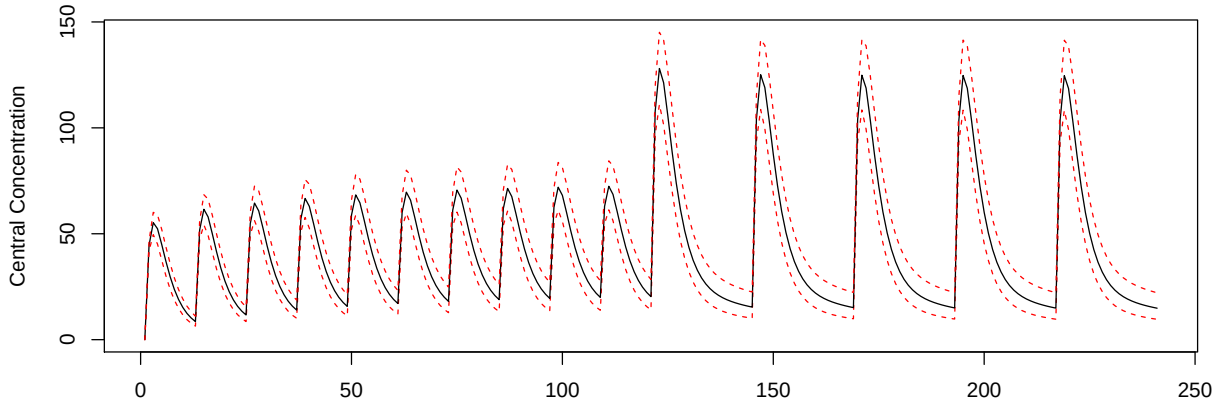
```
nobs <- ev$get.nobs()
cp.all <- matrix(NA, nobs, nsub)
for (i in 1:nsub)
{
    theta <- theta.all[i,]
    x <- mod1$solve(theta, ev, inits=inits)
    cp.all[, i] <- x[, "C2"]
}

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 <- apply(cp.all, 1, quantile, prob = c(0.05, 0.50, 0.95))
matplot(t(cp.q), type="l", lty=c(2,1,2), col=c(2,1,2), ylab="Central Concentration")
```



Facilities for generating R shiny applications

An example of creating an R shiny application to interactively explore responses of various complex dosing regimens is available at http://qsp.engr.uga.edu:3838/RxODE/RegimenSimulator. Shiny applications like this one may be programmatically created with the experimental function `genShinyApp.template()`.

The above application includes widgets for varying the dose, dosing regimen, dose cycle, and number of cycles.

```
genShinyApp.template(appDir = "shinyExample", verbose=TRUE)

library(shiny)
runApp("shinyExample")
```

Click here to go to the Shiny App