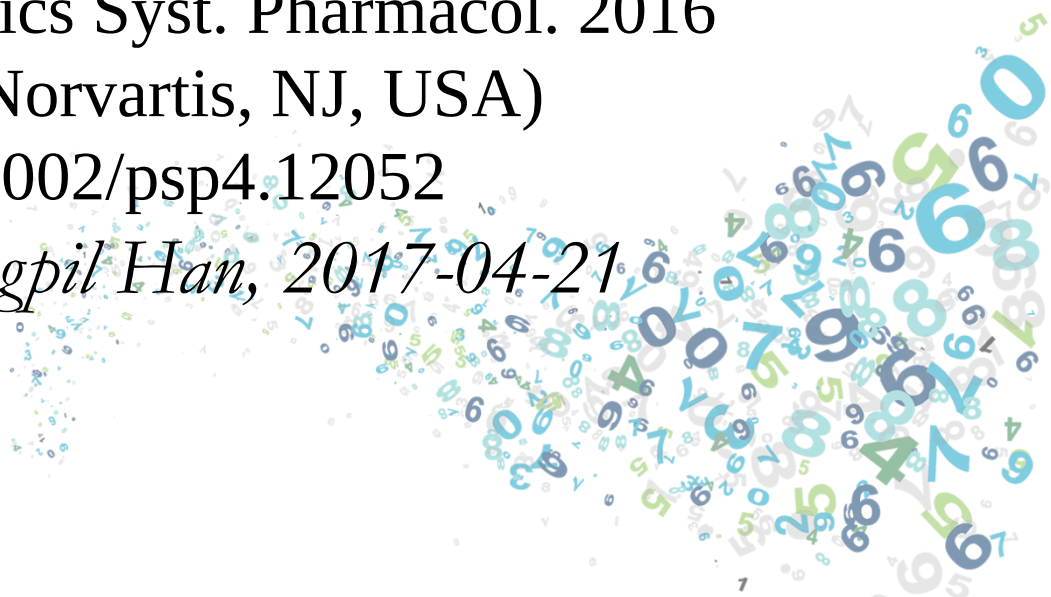
A Tutorial on RxODE: Simulating Differential Equation Pharmacometric Models in R

CPT Pharmacometrics Syst. Pharmacol. 2016
Wang et al. (Norvartis, NJ, USA)
DOI: 10.1002/psp4.12052

Asan CPT, Sungpil Han, 2017-04-21



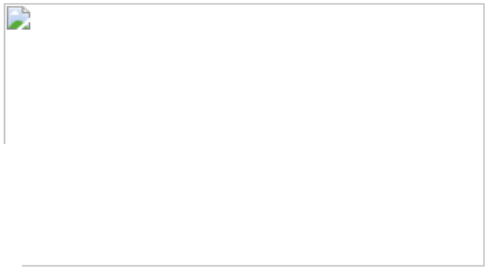
Blog : kimmingul

[R] RxODE를 이용해서 Differential Equation Pharmacometric model을 시뮬레이션 하자.

R

R에 RxODE라는 패키지가 있다.
이것을 이용하면 Differential Equation Pharmacometric model을 시뮬레이션할 수 있다고 하기에 이에 대해 알아보고자 한다.
첨부한 RxODEintro.pdf 파일에 간단한 사용법이 소개되어 있다. 이를 기준으로 아래 내용을 적어본다.

아래 링크를 따라가면 CPT:PSP 에서 2016년에 출간된 RxODE 사용법에 대한 논문이 나온다. (첨부한 pdf 파일과 동일)
http://onlinelibrary.wiley.com/doi/10.1002/psp4.12052/full



Two-compartment pharmacokinetic model with an indirect response pharmacodynamic model.

위와 같은 모델을 시뮬레이션 한다고 보면 된다.

실행함에 있어 문제가 있다.
Linux에서는 잘 작동하는데, Windows 환경 하에서 작동하지 않는다. 뭐가 문제인가 살펴보았더니, 컴파일러 문제였다.
R을 컴파일하는 GCC, Make 등이 필요하다. (linux에는 이런 것들이 기본적으로 설치되어 있으나, Windows에서는 MinGW 류를 설치해야만 한다)
이것은 Rtools 라는 것을 설치하면 쉽게 해결이 가능하다. 아래 사이트에 가서 자신의 R 버젼에 맞는 Rtools 를 다운받아 설치하자.
용량은 대략 100메가 초반이다. 설치할 때,Path에 포함하겠냐고 묻게 되는데, 여기에서 체크를 해주어야 한다(필수).
그렇지 않으면 시스템 설정에 직접 들어가서 PATH를 수정해줘야한다. 초보에게는 어려운 작업이니 꼭 설치시 check!!
https://cran.r-project.org/bin/windows/Rtools/

아래 게시물을 보면, RxODE를 windows 환경 하에서 설치하는 방법에 대해 자세히 기술되어 있으니 참고하시길.
https://github.com/hallowkm/RxODE/blob/master/RxODE/inst/Install-windows.Rmd

-- 예제 시작 --

1. 기본적인 RxODE 예제

```
# 설치하기
install.packages("RxODE")

# Differential Equation으로 모형 설정
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;
"
```

http://blog.naver.com/kimmingul/220984707121

Introduction

Simulation and drug development

- The use of simulations for drug development
  - **cost-effective approach** for the **exploration of multiple dosing regimens** and their **pharmacodynamics effects** over **diverse patient populations**.
- Several factors
  - hinder greater utilization of pharmacometric simulation in drug development
  - difficult to identify a priori all possible simulation scenarios of interest
  - 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.
  - **typically presented through static graphics, and typically leads to further questions and other scenarios to evaluate**

Current Simulation Tools

- Performing simulations with most currently available simulation tools
  - cumbersome, tedious, and time-consuming
  - extensive custom programming and moving between one software application to perform simulations and another application to visualize simulations
  - iterative process
    - **multiple meetings/discussions**, with **significant lag time between each**
    - **loss of momentum** and **lost opportunities for making quantitatively driven decisions**.
- Great need for more efficient simulation processes
  - facilitate interactive, real-time evaluation and iteration on simulation scenarios.

R software

- R software
  - excellent set of tools for analyzing and visualizing simulation results
    - lattice
    - ggplot2 packages
    - Shiny package, Web-based interfaces to R programs can be easily generated.
- Ideal environment
  - perform pharmacometric simulations in real time

Limitation of R in pharmacometrics

- lacked extensive facilities to support modeling based on differential equations (DE) like the ones used in pharmacokinetic/pharmacodynamic (PK/PD) applications
- R package deSolve
  - now provides many general-purpose DE solvers
  - 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
    - 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**).

RxODE

- Efforts have increased to develop tools to address these problems and facilitate efficient simulation in R
  - PKPDsim package
  - Simulx package
- new R package, **RxODE**
  - facilitates ***quick and efficient simulations*** of ordinary differential equation (ODE) models in R
- RxODE provides
  - **elegant, efficient, and versatile way** to specify **dosing scenarios**
  - multiple routes of administrations within a single regimen, sampling schedules

RxODE

- **simulations with between-patient variability**, and **minimizes the amount of custom coding required for pharmacometric simulations**
- RxODE vs deSolve
  - 8–10 times faster for a model with four ODEs
  - 100 times faster for a model with seven ODEs
- **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**
- Designed with pharmacometric models in mind
- **function for directly generating R Shiny applications**
  - **useful for interactively probing the model**
  - Web-based application can then be further customized by the user and used to facilitate interactive simulations

Case Study : Evaluating Adaptive Dosing Regimens

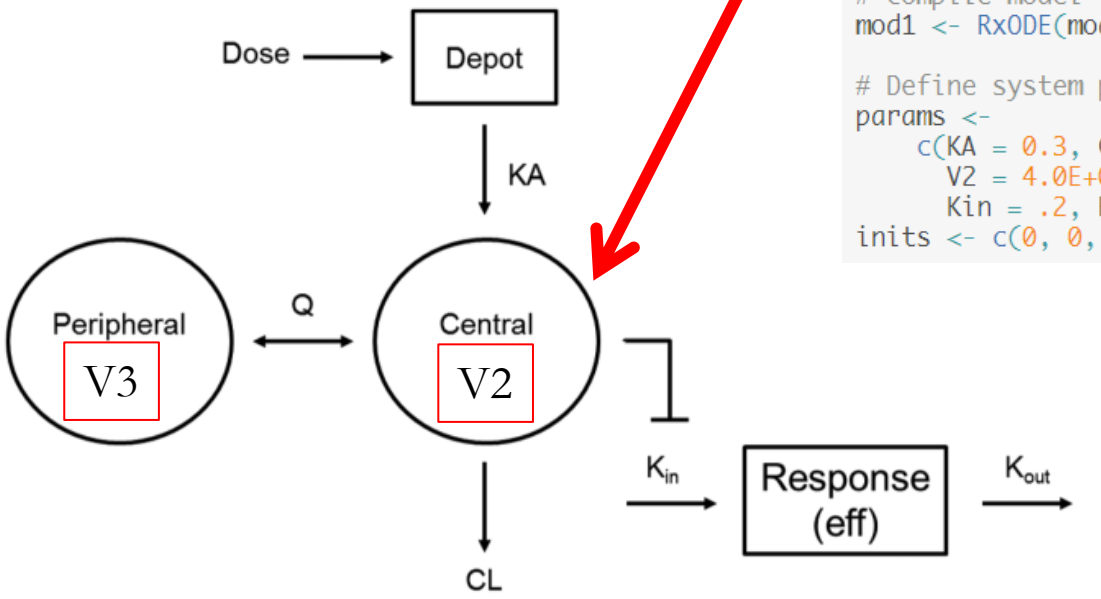
dose adjustment algorithm

- 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

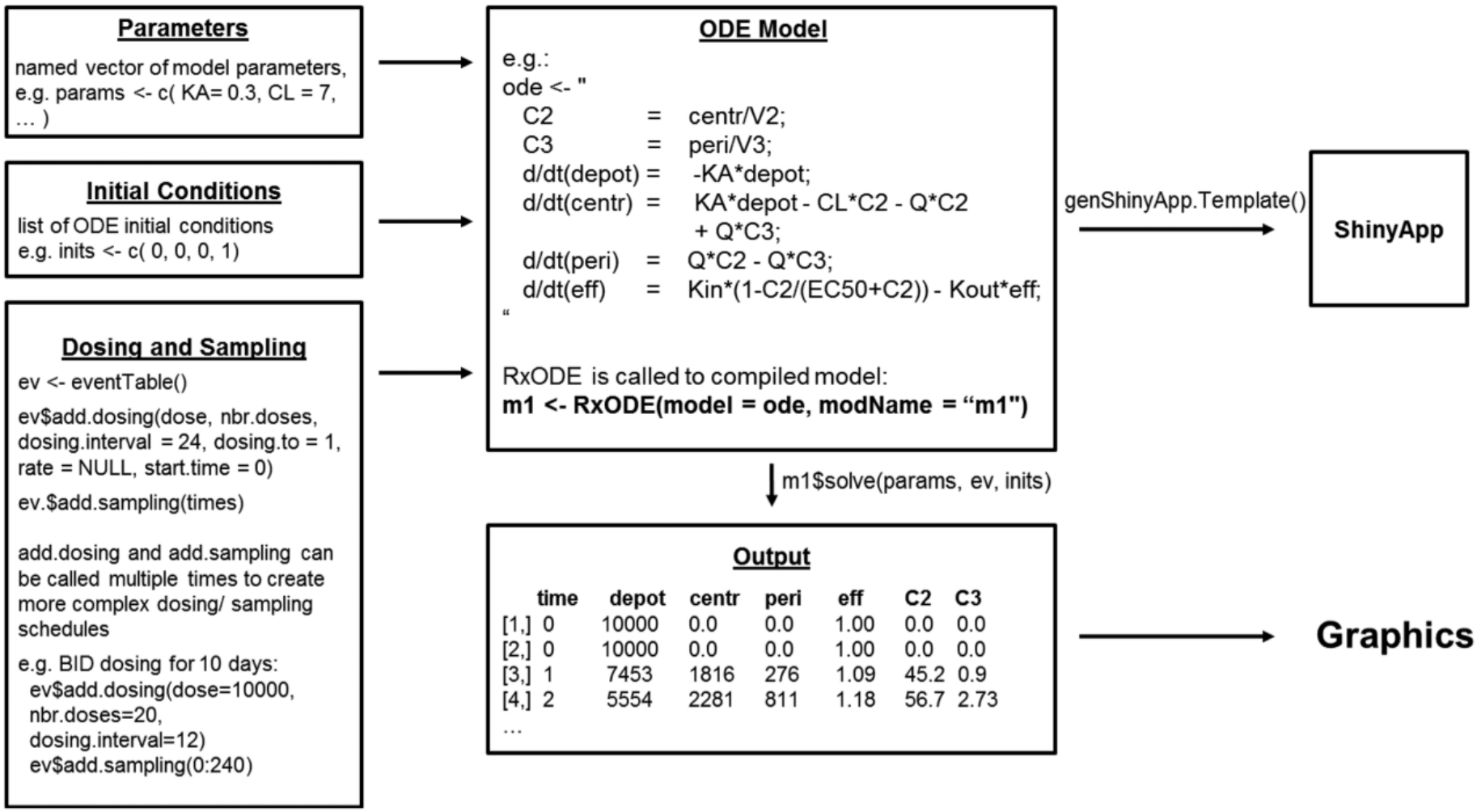
- a case study for adaptive dosing. For illustration purposes, we have chosen a system where the PK is described by a two-compartment model
  - PD effect is modeled with an **indirect response**
- In order to maintain efficacy while avoiding adverse side effects, it has been determined that **inhibition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

Fig 1. 2 CMT PK model with an indirect response PD model.



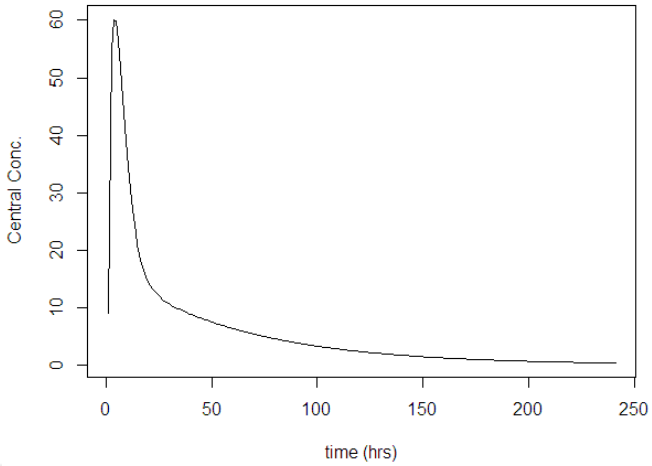

```
######################Set up ####################
# Load RxODE
library(RxODE)

# 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*(1-C2/(EC50+C2)) - Kout*eff;
"
# Compile model
mod1 <- RxODE(model = ode, modName = "mod1")

# Define system parameters
params <-
    c(KA = 0.3, CL = 7,                          # central
      V2 = 4.0E+01, Q = 1.0E+01, V3 = 3E+02, # peripheral
      Kin = .2, Kout = .2, EC50 = 8)             # effects
inits <- c(0, 0, 0, 1)
```

WORKFLOW FOR PERFORMING SIMULATIONS IN RxODE


Parameters
named vector of model parameters, e.g. params <- c(KA= 0.3, CL = 7, …)
Initial Conditions
list of ODE initial conditions e.g. inits <- c(0, 0, 0, 1)
Dosing and Sampling
ev <- eventTable()
ev$add.dosing(dose, nbr.doses, dosing.interval = 24, dosing.to = 1, rate = NULL, start.time = 0)
ev.$add.sampling(times)
add.dosing and add.sampling can be called multiple times to create more complex dosing/ sampling schedules
e.g. BID dosing for 10 days:
ev$add.dosing(dose=10000, nbr.doses=20, dosing.interval=12)
ev$add.sampling(0:240)
ODE Model
e.g.:
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*(1-C2/(EC50+C2)) - Kout*eff;
"
RxODE is called to compiled model:
m1 <- RxODE(model = ode, modName = "m1")
genShinyApp.Template()
ShinyApp
m1$solve(params, ev, inits)
Output
time depot centr peri eff C2 C3
[1,] 0 10000 0.0 0.0 1.00 0.0 0.0
[2,] 0 10000 0.0 0.0 1.00 0.0 0.0
[3,] 1 7453 1816 276 1.09 45.2 0.9
[4,] 2 5554 2281 811 1.18 56.7 2.73
…
Graphics

Only considering Thetas

```
#######################Simulate a single dose ########

# Initialize event table
ev <- eventTable()

# Specify dose
ev$add.dosing(dose = 10000, nbr.doses = 1)

# Specify sampling
ev$add.sampling(0:240)

# Simulate
x <- mod1$run(params, ev, inits)
```




```
# Plot results
matplot(x[,"C2"], type = "l", xlab = "time (hrs)", ylab = "Central Conc.")

#######################Simulate 5 days BID followed by 5 days QD########

ev <- eventTable()

# Specify 5 days BID dosing
ev$add.dosing(dose = 10000, nbr.doses = 10, dosing.interval =

# Use "start.time" parameter to specify 5 days QD starting at
ev$add.dosing(dose = 20000, nbr.doses = 5, dosing.interval = 2
ev$add.sampling(0:240)

# Simulate and plot
x <- mod1$run(params, ev, inits)
matplot(x[,"C2"], type = "l", xlab = "time (hrs)", ylab = "Cen
```

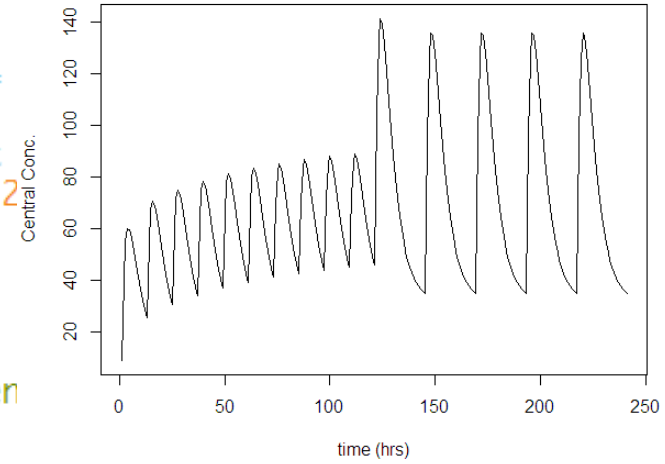
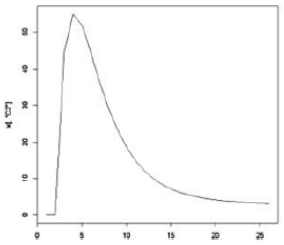
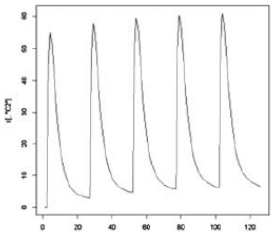
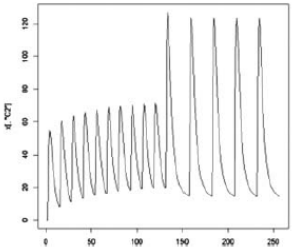
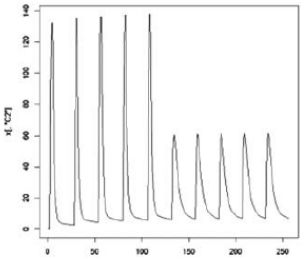
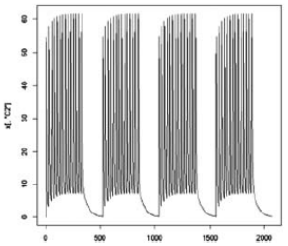
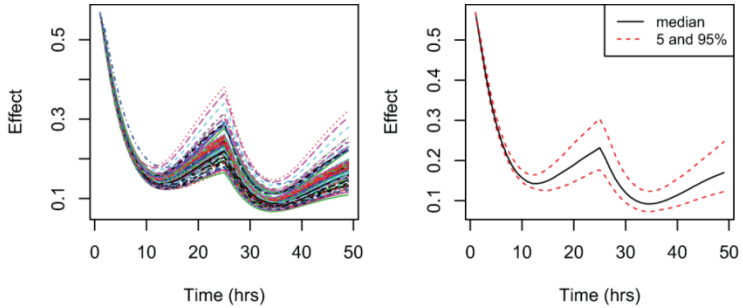


Table 1 The add.dosing() function provides an efficient method to specify a variety of dosing schedules

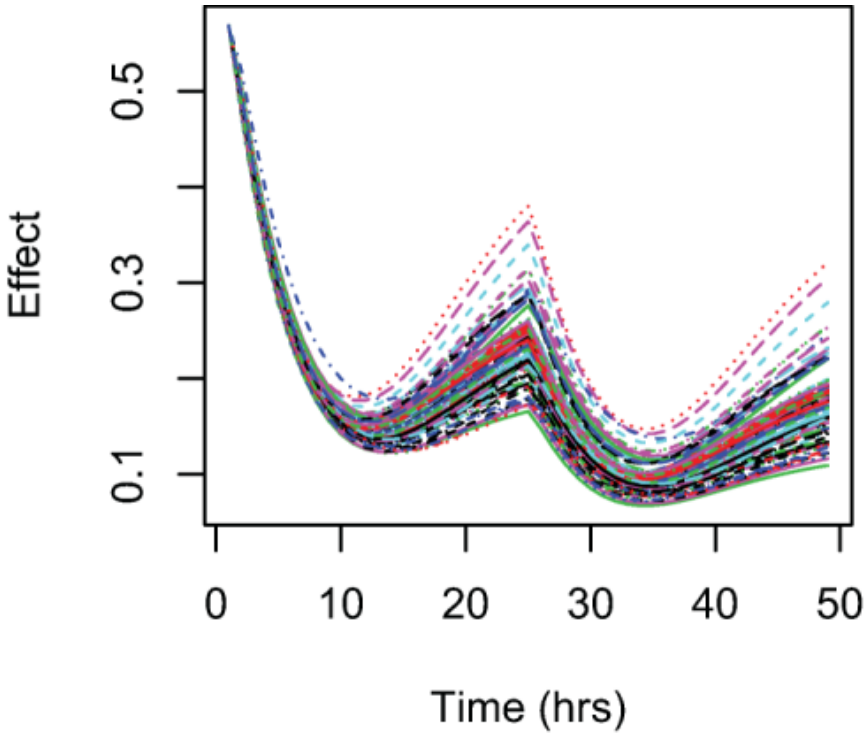
Single dose	ev$add.dosing(dose=10000, nbr.doses=1)	
Multiple doses	ev$add.dosing(dose=10000, nbr.doses=5, dosing.interval=24)	
Bid for 5 days, followed by qd for 5 days	ev$add.dosing(dose=10000, nbr.doses=10, dosing.interval=12) ev$add.dosing(dose=20000, nbr.doses=5, dosing.interval=24, start.time=120)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

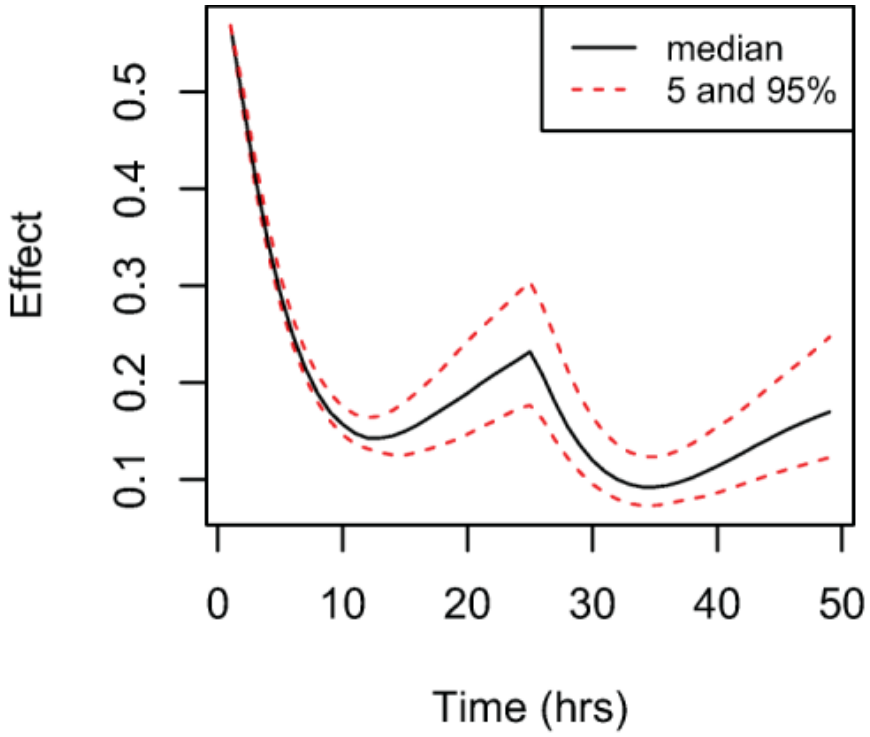
Simulating with variabililty (eta)

```r
#####################Simulate with variability#####################
# Perform simulation 100 times
nsub <- 100

# Create a parameter matrix with correlated interindividual variability on CL and V2

library(MASS)
sigma= matrix(c(0.09,0.08,0.08,0.25),2,2)
mv = mvrnorm(n=nsub, rep(0,2), sigma)
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)

# QD dosing for 2 days
ev <- eventTable()
ev$add.dosing(dose = 20000, nbr.doses = 10, dosing.interval = 24)
ev$add.sampling(0:48)

res <- NULL #Create an empty matrix for storing results
i = 1
# Loop through each row of parameter values and simulate
for (i in 1:nsub)  {
    params <- params.all[i,]
    x <- mod1$run(params, ev, inits = inits)
    res <- cbind(res,x[,"eff"])
}

#The same can be achieved more efficiently by replacing the above for-loop with:
#    res <- apply(params.all, 1, function(params) mod$run(params, ev, inits)[, "eff"])

# Plot results
par(mfrow = c(1,2), mar = c(4,4,1,1))
matplot(res, type = "l",  ylab = "Effect", xlab = "Time (hrs)")

# Calculate and plot quantiles
res.q.t <- apply(res, 1, quantile, prob = c(.05, .5, .95))
matplot(t(res.q.t), type = "l", lty = c(2,1,2), col = c(2,1,2), ylab = "Effect", xlab = "Time (hrs)")
legend("topright", c("median","5 and 95%"), lty = c(1,2), col = c("black","red"), cex = .8)
```



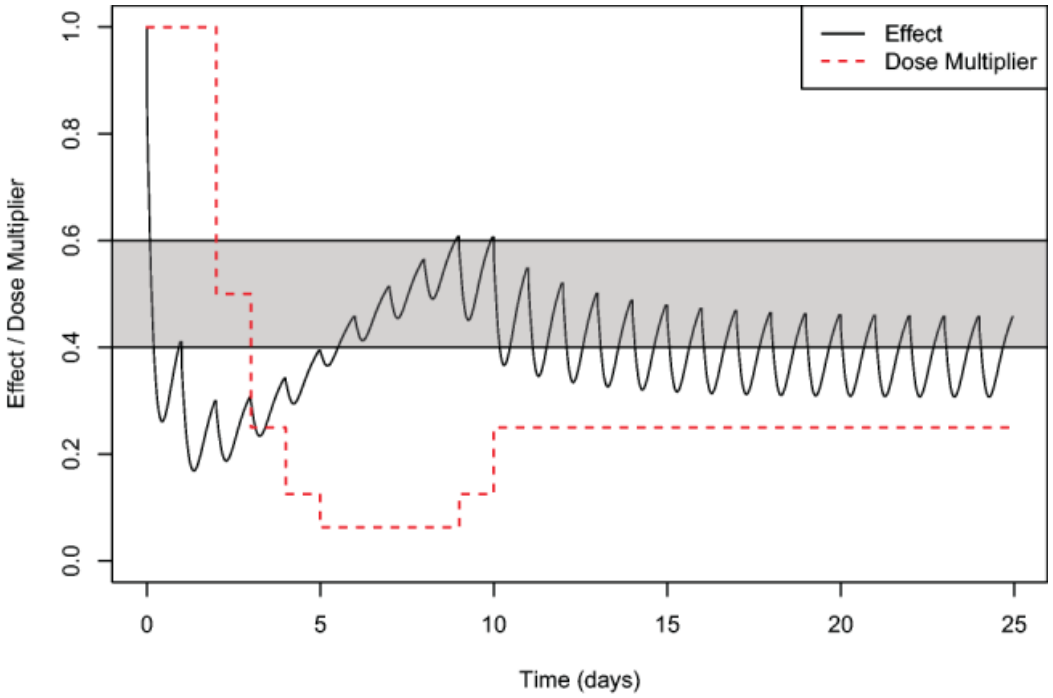
2 days of QD dosing, correlated interindividual variability on CL and V2.



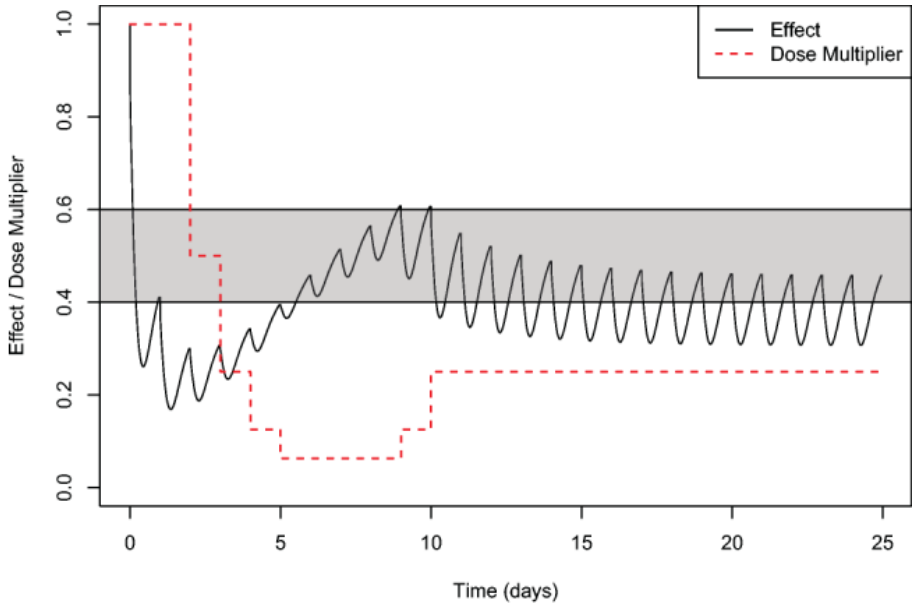


Simulated adaptive dosing

- Decision Rule:
  - If trough effect <.4, cut dose in half
  - If trough effect in [0.4, 0.6], do not change dose
  - If trough effect > 0.6, double the dose.

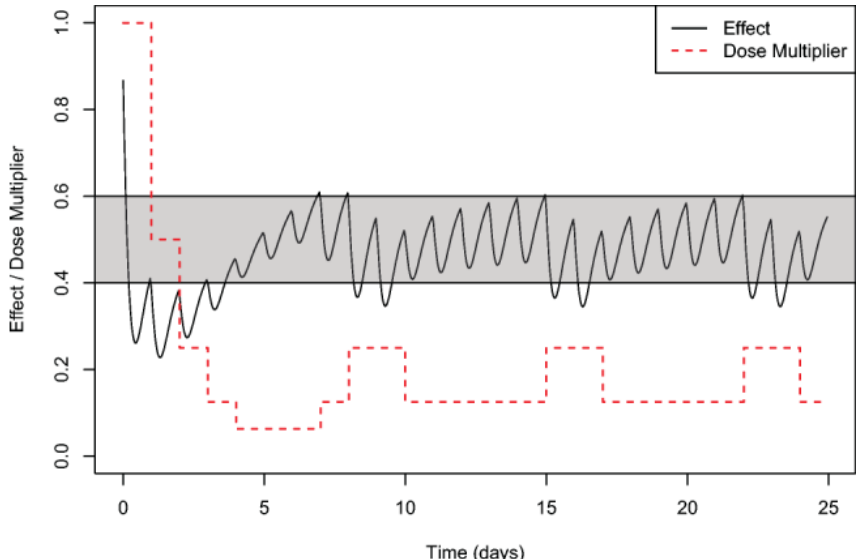


```
#Decision Rule:
#   If trough effect <.4, cut dose in half
#   If trough effect in [0.4, 0.6], do not change dose
#   If trough effect > 0.6, double the dose.
effect.limits <- c(0, .4, .6, 9)  #decision rule limits
dose.multipliers <- c(.5, 1, 2)   #decision rule effects

#simulation parameters
ndays <- 25               #number of days to simulate
unit.dose <- 10000
start.dose <- 1
sampling.frequency <- 1         #sample every day

# Define system parameters
params <-
    c(KA = 0.3, CL = 7,             # central
      V2 = 4.0E+01, Q = 1.0E+01, V3 = 3E+02, # peripheral
      Kin = .2, Kout = .2, EC50 = 8)              # effects
inits <- c(0, 0, 0, 1)

#Initialize other variables
vars <- c("depot", "centr", "peri", "eff")
res <- NULL          #results vector
time.vec <- NULL     #time vector
doses <- NULL   #doses vector

#Simulate a  period of time (set by sampling.frequency), check dose and adjust according to decision
rules, then simulate #next time period
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
        inits <- x[dim(x)[1], vars]         # Use last value of state variables as new initial
conditions
        wh <- cut(inits["eff"], effect.limits)  # Compare trough effect with decision rule limits
        this.multiplier <- dose.multipliers[wh] # Determine dose multiplier accordingly
    }
    this.multiplier <- this.multiplier*last.multiplier  # Adjust dose
    last.multiplier <- this.multiplier          # Store new dose
    ev <- eventTable()              # Generate event table
    # Specify dosing
    ev$add.dosing(dose = this.multiplier*unit.dose, dosing.interval = 24, nbr.doses = sampling
.frequency)
    ev$add.sampling(0:(24*sampling.frequency))          # Specify sampling
    x <- mod1$run(params, ev, inits)                    # Run simulation
    time.total <- ev$get.EventTable()[,"time"]+(i-1)*(24)       # Calculate time vector
    doses <- rep(last.multiplier, length(time))
    x <- cbind(x,time.total, doses)                # Compile and store results
    res <- rbind(res, x)
}
```



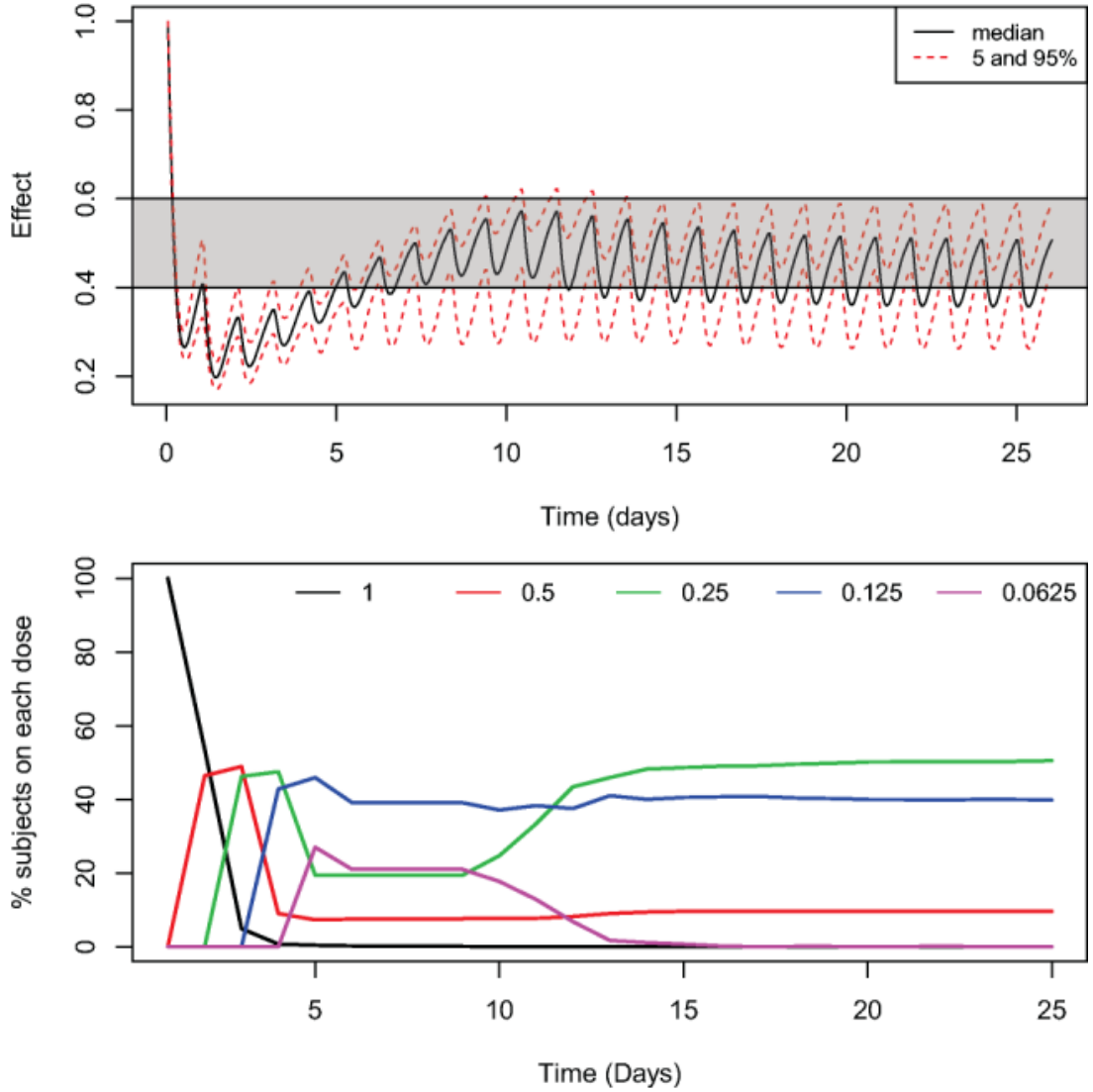
Simulated adaptive dosing

- Decision Rule:
  - If peak effect <0.4, cut dose in half
  - If effect at tmax < 0.4, cut the dose in half
  - If trough effect and effect at tmax in [0.4, 0.6], do not change dose
  - If trough effect > 0.6, double the dose.
  - If effect at tmax > 0.6, double the dose.

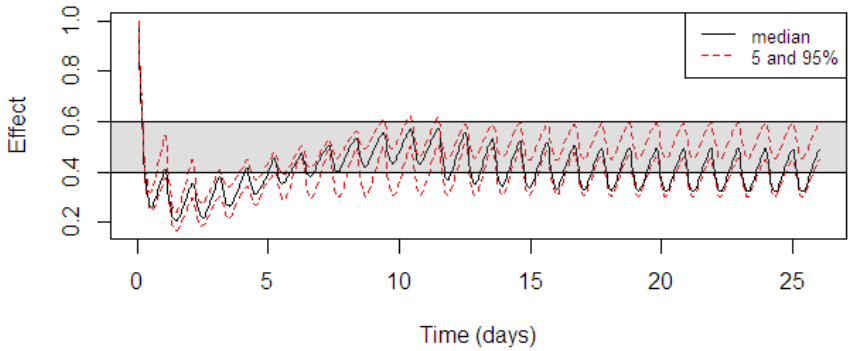


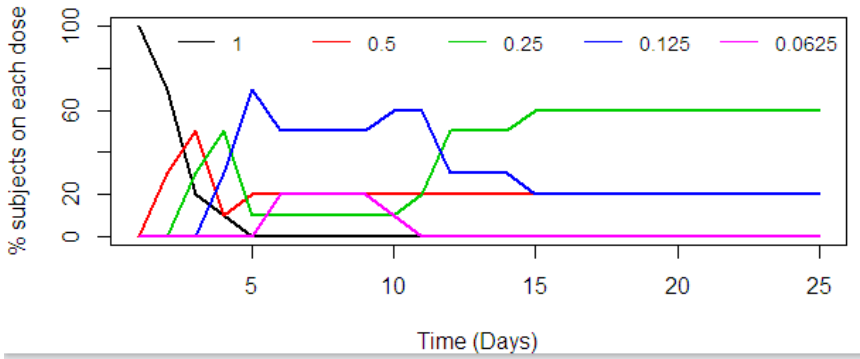
Simulating Adaptive Dosing with Variabililty (eta)

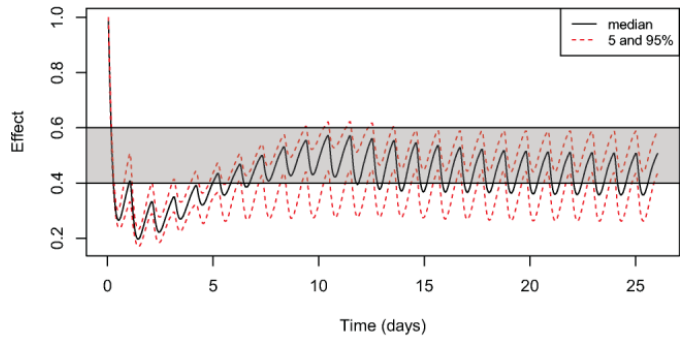
Simulating Adaptive Dosing with Variabililty

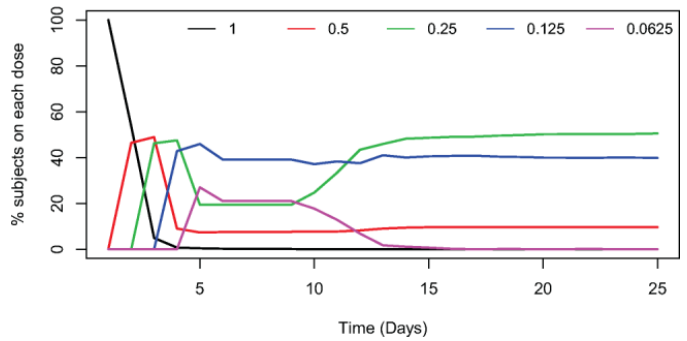


```r
# Perform simulation 100 times
nsub <- 10;

# Create a parameter matrix with correlated interindividual variability on CL and V2
library(MASS)
sigma= matrix(c(0.09,0.08,0.08,0.25),2,2)
mv = mvrnorm(n=nsub, rep(0,2), sigma)
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)


# Decision Rule:
# If trough effect <.4, cut dose in half
# If trough effect in [0.4, 0.6], do not change dose
# If trough effect > 0.6, double the dose.
effect.limits <- c(0, 0.4, 0.6, 9)   # decision rule limits
dose.multipliers <- c(0.5, 1, 2)   # decision rule effects

# Simulation parameters
ndays <- 25      #number of days to simulate
unit.dose <- 10000
start.dose <- 1

# Initialize other variables
vars <- c("depot", "centr", "peri", "eff")
res <- NULL
time.vec <- NULL
doses <- NULL

for (j in 1:nsub) {     # Loop through each row of parameter values and simulate
    this.run <- NULL
    DOSE <- NULL
    for (i in 1:ndays) {    # Simulate each day
        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
            inits <- x[dim(x)[1], vars]             # Use last value of state variables as new initial
conditions
            wh <- cut(inits["eff"], effect.limits)  # Compare trough effect with decision rule limits
            this.multiplier <- dose.multipliers[wh] # Determine dose multiplier accordingly
        }
        this.multiplier <- this.multiplier*last.multiplier      # Adjust dose
        last.multiplier <- this.multiplier              # Store new dose
        ev <- eventTable()                  # Generate event table
        # Specify dosing
        ev$add.dosing(dose = this.multiplier*unit.dose, dosing.interval = 24, nbr.doses = sampling
.frequency)
        ev$add.sampling(0:(24*sampling.frequency))      # Specify sampling
        params <- params.all[j,]                    # Specify parameters for the current subject
        x <- mod1$run(params, ev, inits)                # Run simulation
        time.total=ev$get.EventTable()[,"time"]+(i-1)*(24)  #calculate time vector

        #compile results for this subject, all days
        DOSE <- rbind(DOSE, last.multiplier)
        this.run <- rbind(this.run, cbind(x, time.total))
    }
    # Compile results for all subjects
    res <- cbind(res, this.run[, "eff"])
    time.vec <- this.run[, "time.total"]
    doses <- cbind(doses, DOSE)
}
```









ShinyApps

- http://qsp.engr.uga.edu:3838/adaptiveDosing/

Therapeutic monitoring simulator

Dose:

0 10 20

# of Days:

5 20 50

Sampling Frequency (days):

1 10

Define limits for target range:

Lower Effect Limit:

0 0.4 0.6

Upper Effect Limit:

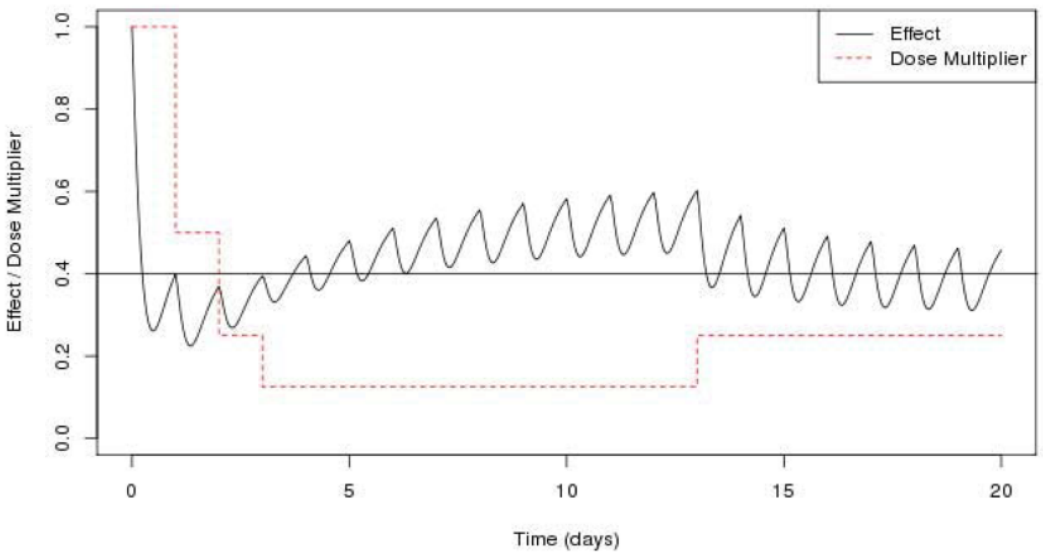
0.4 0.6 1

- [x] Control trough effect (24hrs) within target range
- [ ] Control peak effect (tmax~12hrs) within target range

Summary of therapeutic monitoring:

```
% time below lower limit:  37.4%
% time above upper limit:   1.2%
```

Concentration time course



Conclusion

- RxODE
  - R package that provides tools for the efficient simulation of complex dosing regimens via PK/PD models described by ODEs.
- **Advanced static and interactive visualization**
  - **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.**
- Parameter estimation through the many existing statistical estimation algorithms in R
  - nonlinear mixed effects models
  - stochastic approximation expectation-maximization (SAEM)
  - Bayesian methods using Gibbs sampling, e.g., JAGS
- Future work includes developing functionality to aid users in linking RxODE models with these estimation algorithms in a more efficient manner.