Case study 2: a one-compartment PK model involving both covariates and BQL

Introduction

- R is one of the most widely used softwares among pharmacometricians to perform data manipulation/visualization and statistical analysis.
- RsNLME provides a R interface to the Phoenix NLME engine to enable users to
  - Define PK/PD models via R objects (package **RsNlme**).
  - Use the “Initial Estimates" shiny app to visually determine a set of reasonable initial values for fixed effects (package **RsNlme**).
  - Perform estimation and simulation in a R environment with the capability of parallelizing the runs using Multicore, MPI and Grids (SGE/Torque/LSF) in-house or hosted on AWS (package **Certara.NLME8**).
  - Access the xpose graphics library for PK/PD models by creating compatible database from NLME results (package **Xpose.Nlme**).

Objectives

Demonstration of RsNLME through a one-compartment PK model involving both covariates and BQL.

- Define the model through **RsNlme**.
- Map model variables to input dataset columns.
- Fit the model, and then use the **xpose.Nlme** package to create commonly used diagnostic plots.
- VPC analysis for the fitted model.

Note: R script and input dataset for this example can be found in C:\Program Files\R\R-n.n.n\library\RsNlme\

Define the model through RsNlme

Structural Model

```
# model name
ModelName = "OneCpt_IVBolus_ContCovariatesOnClV_BQL_Laplacian"

# define the basic PK model (a one-compartment model with IV bolus)
model = pkmodel(numCompartments = 1, modelName = ModelName)
```

Covariate Model

$$V = tvV \left(\frac{\mathrm{BW}}{30}\right)^{\mathrm{dVdBW}} \exp(nV)$$

$$Cl = tvCl \left(\frac{\mathrm{BW}}{30}\right)^{\mathrm{dCldBW}} \frac{\mathrm{PMA}^{\mathrm{Gam}}}{\mathrm{PMA}^{\mathrm{Gam}} + \mathrm{PMA}_{50}^{\mathrm{Gam}}} \exp(nCl)$$

```
# define covariates BW and PMA
BW = NlmeCovariateParameter(name = "BW", type = Continuous
                        , continuousType = CovarNumber, centerValue = "30")
PMA = NlmeCovariateParameter(name = "PMA")

#automatically incorporate covariate BW into the basic model
model = addCovariates(model, covariates = c(BW, PMA)
                        , effects = c("V" = "BW", "Cl" = "BW"))

# maually incorporate covariate PMA into the basic model
structuralParam(model, "Cl") = c(style = Custom
, code = "stparm(Cl=tvCl*(BW/30)^dCldBW*(PMA^Gam/(PMA^Gam+PMA50^Gam))*exp(
 nCl))", extraCode = c("fixef(PMA50=c(, 5, ))", "fixef(Gam=c(, 1, ))"))
```

Residual Error Model

```
# set residual error model (default: additive model with standard
# deviation = 1 and isBQL = FALSE, where the isBQL option is used to
# specify whether the input dataset involves BQL observations or not)
residualEffect(model, "C") = c(errorType = Multiplicative
                                , SD = "0.1", isBQL = TRUE)
```

Initial Values for Theta and Omega

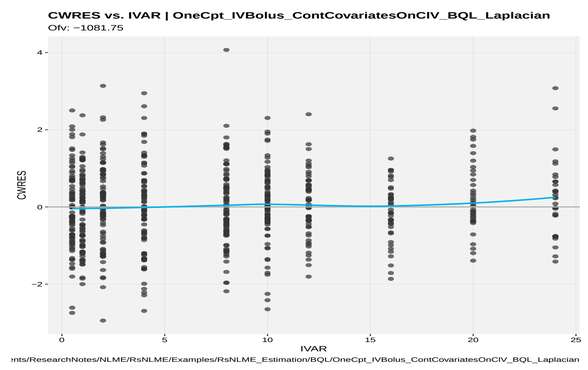
```
# set initial values for fixed effects (default value for the one related
# to a covariate is 0; otherwise, it is 1)
initFixedEffects(model) = c(tvV = 20, tvCl = 20, dVdBW = 1, dCldBW = 1)

# set initial values for random effects (the default value is 1)
initRandomEffects(model) = c(Diagonal, isFrozen=FALSE, "nV,nCl", "0.1,0.2")
```

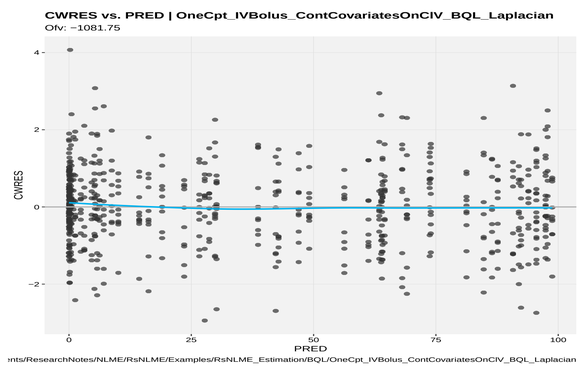
Map model variables to input dataset columns

```
# load the input dataset
dt_InputDataSet = fread("OneCpt_IVBolus_ContCovariatesOnClV_BQL.csv")

# initialize model mapping and automatically mapping some of the model
# variables to the data columns
initColMapping(model) = dt_InputDataSet

# manually set up the mapping for the rest of variables
modelColumnMapping(model) = c(A1 = "Dose")
```

Fit the model and create diagnostic plots

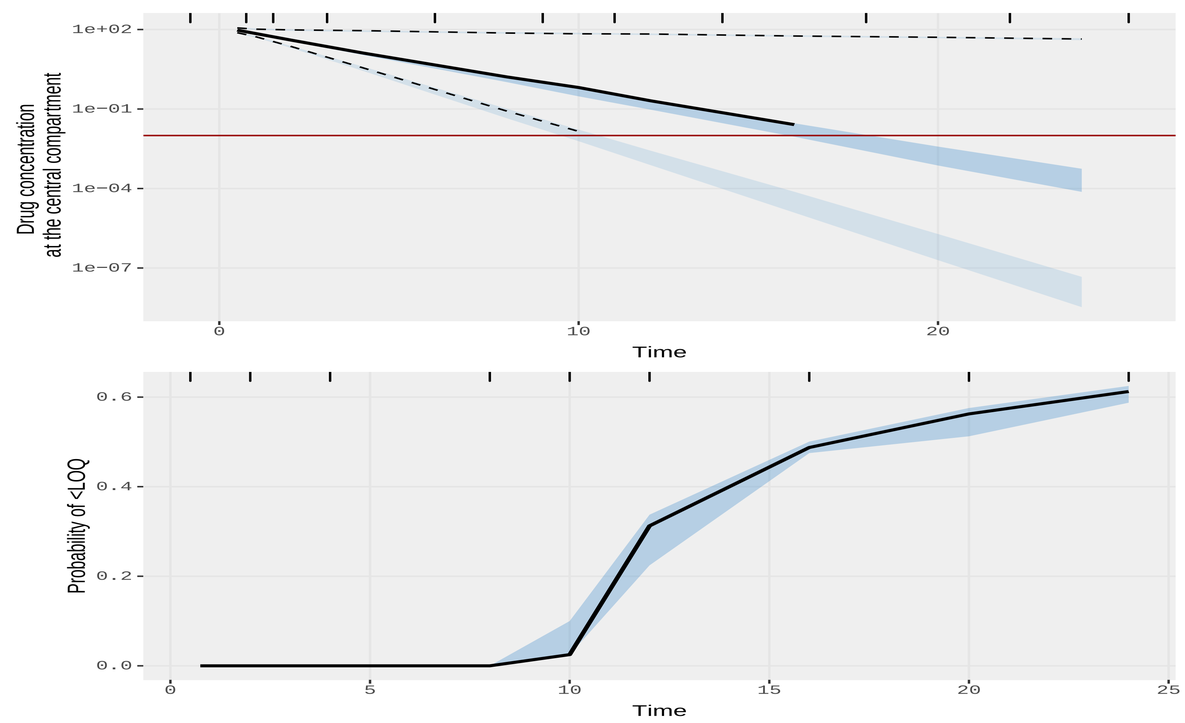
```
# host setup: run locally with MPI enabled
host = NlmeParallelHost(sharedDirectory = Sys.getenv("NLME_ROOT_DIRECTORY")
                        , parallelMethod = NlmeParallelMethod("LOCAL_MPI")
                        , hostName = "MPI", numCores = 4)
# engine setup
engineParams = NlmeEngineExtraParams(PARAMS_METHOD = METHOD_LAPLACIAN
                , PARAMS_NUM_ITERATIONS = 1000, PARAMS_SAND="TRUE")

# fit the model
job = fitmodel(host, engineParams, model)

# Imports results of an NLME run into xpose database
xp = xposeNlme(dir = model@modelInfo@workingDir, modelName = ModelName)

# Create the plot for the CWRES against the independent variable
res_vs_idv(xp, res = "CWRES", type = "ps")

# Create the plot for the CWRES against population predications
res_vs_pred(xp, res = "CWRES", type = "ps")
```





VPC analysis for the fitted model

```
# Accept the estimates for fixed effects, random effects and sigma
modelVPC = acceptAllEffects(model)

# Set the output file name to "predout.csv" (default name: "out.txt")
modelVPC@dataset@outputFilename = "predout.csv"

# VPC setup
VPCSetup = NlmeVpcParams(numReplicates = 100, seed = 1)

# Run VPC
job = vpcmodel(host, VPCSetup, modelVPC)

# Load observed data and simulated data
dt_ObsData = getObsData(dt_InputDataSet)
dt_SimData = getSimData(input=dt_InputDataSet, simFile="predout.csv")

# Use the vpc package to create a VPC plot for un-censored data that shows
# the censor limit (LLOQ) as a horizontal line
vpc(sim=dt_SimData, obs=dt_ObsData, lloq=lloq_value, log_y=TRUE, log_y_min=1
 e-9, xlab="Time", ylab="Drug concentration \n at the central compartment")

# Use the vpc package to create a VPC plot for the probability of
# left-censored data
vpc_cens(sim=dt_SimData,  obs=dt_ObsData, lloq=lloq_value, xlab="Time")
```



Conclusions

- RsNLME provides R command line access to the Phoenix NLME engine allowing pharmacometricians with little or no knowledge of Phoenix NLME to format and visualize data, build and analyze models, and post-process results.
- RsNLME also provides greater flexibility for advanced Phoenix NLME users to work seamlessly with other R packages within the R environment.