Exploring DDMoRe interoperability with the Simeoni model

This example script is intended to illustrate how to use the ddmore R package to perform a M&S workflow using the DDMoRe Standalone Execution Environment (SEE).

The following steps are implemented in this workflow:

- Initialisation of the R console
- Exploratory graphical analysis
- Parameter estimation with Monolix using SAEM and model evaluation in Xpose
- Parameter estimation in NONMEM using FOCE and model evaluation in Xpose
- Parameter estimation in WinBUGS using MCMC
- Comparison of parameter estimates
- Updating parameter estimates in the MDL Parameter Object using MLE values from NONMEM
- Performing a Visual Predictive Check (VPC) in PsN
- Using MLE values from NONMEM to simulate new observed values using the simulx function in the mlxR package.
- Updating parameter estimates with NONMEM and simulx output
- Evaluation of study design characteristics using PFI M
- Evaluation of study design characteristics using PepED

Background

Tumour growth inhibition model

The efficacy of most drugs approved for oncology have first been tested in xenograft models. This is an in-vivo preclinical experiment with drug and control arms of 6-10 mice. Tumour cells are inoculated at Day 0, and drug administered when tumour size has reached a certain size. Tumour size is then measured at regular intervals until approximately Day 40.

The model describes the PKPD relationship of drug concentration to tumour size across time. The model is then used to quantitatively assess and compare the potency of different drugs, predict tumour growth for regimens not tested and to evaluate and define optimal designs for future trials.

The Simeoni tumour growth inhibition model Cancer Res, 2004 is widely used for this purpose. The model describes the initial exponential growth, followed by linear phase of tumour cells, the influence of drug concentration to damage cells and the process of cell death.

Figure 1: Simeoni model diagram
The data presented here are simulated data based on the original publication. The cell line is A2780. The active treatment group is paclitaxel where 30mg/kg was administered every 4 days for 3 administrations starting on day 8. PK of paclitaxel was modelled in a previous step and PK parameters fixed for this model. The data are summary statistics (mean) across the 6 mice in each arm. A model allowing for inter-individual variability was examined, but inter-individual variability was low and for expediency here the naive pooled approach is used to model the effect of drug across time.

**Initialisation**

Clear workspace and set working directory

```r
rm(list=ls(all=F))
getwd()
```

```r
## [1] "C:/SEE/MDL_IDE/workspace/Demo_Simeoni/models"
Sys.getenv("MDLIDE_WORKSPACE_HOME")
## [1] "C:\SEE\MDL_IDE\workspace"
```

```r
mydir <- file.path("~/Demo_Simeoni/models")
setwd(mydir)
```

Set name of .mdl file and dataset

```r
datafile <- "Simulated_simeoni2004_data.csv"
mdlfile <- "Simeoni_Demo.mdl"
model <- "Simeoni_Demo"
```

Read the different MDL objects needed for upcoming tasks An MDL file may contain more than one object of any type (though typically only one model!) so these functions return a list of all objects of that type found in the target MDL file. To pick out the first, use the double square bracket notation in R to retrieve the first item of the list.

```r
myModelObj <- getModelObjects(mdlfile)[[1]]
myDataObj <- getDataObjects(mdlfile)[[1]]
myParameterObj <- getParameterObjects(mdlfile)[[1]]
myPriorObj <- getPriorObjects(mdlfile)[[1]]
myDesignObj <- getDesignObjects(mdlfile)[[1]]
```

You can also refer to objects by their name.

```r
myTaskPropertiesObj_NM <- getTaskPropertiesObjects(mdlfile)[["simeoni2004_NONMEM_task"]]
myTaskPropertiesObj_MLX <- getTaskPropertiesObjects(mdlfile)[["simeoni2004_Monolix_task"]]
myTaskPropertiesObj_BUGS <- getTaskPropertiesObjects(mdlfile)[["simeoni2004_BUGS_task"]]
myTaskPropertiesObj_OptDes <- getTaskPropertiesObjects(mdlfile)[["simeoni2004_Evaltask"]]
```

**Exploratory Data Analysis**

The `getDataObjects` function reads the MDL defining the location and content of the data file. Use the `ddmore` function `readDataObj` to read the MDL and create an R object based on the MDL Data Object.

```r
myData <- readDataObj(myDataObj)
```

Let’s look at the first 6 lines of the data set
head(myData)

```
## ID TIME DV AMT MDV CMT
## 1 1 0 0.00000 0 1 3
## 2 1 8 0.47809 0 0 3
## 3 1 10 0.95340 0 0 3
## 4 1 13 2.14540 0 0 3
## 5 1 15 3.41620 0 0 3
## 6 1 18 4.35640 0 0 3
``` 

Extract only observation records
myEDADatay < myData[myData$MDV==0,]

Plot the data using ggplot2 library
plot1 <- ggplot(aes(y=DV, x=TIME, group=ID), data=myEDADatay) +
  geom_line() +
  geom_point() +
  labs(y=" Tumour volume (mm3)" , x="Time (days)"")

print(plot1)

Model Development

Next we will illustrate interoperability across three estimation software tools: NONMEM, Monolix and BUGS.
To do this, we require a Data Object, Parameter Object and Model Object. We also require the relevant Task Properties Object for each target software tool. We assemble each of these into a Modelling Object Group (MOG) which then is written to an MDL file and converted to the target software input via the DDMoRe model exchange standard PharmML.

ESTIMATE model parameters using Monolix

Assemble the Modelling Object Group (MOG) for MLX
The createMogObj function creates a Modelling Object Group comprising of Data, Parameter, Model and Task (for estimation). (Table 1, first row):
MLX.MOG <- createMogObj(dataObj = myDataObj,
  parObj = myParameterObj,
  mdlObj = myModelObj,
  taskObj = myTaskPropertiesObj_MLX)

We then write the MOG back out to an .mdl file using the writeMogObj function.
mdlfile.MLX <- paste0(model,"_MLX.mdl")
writeMogObj(MLX.MOG, mdlfile.MLX)

The ddmore estimate function translates the contents of the .mdl file to MLXTRAN and then estimates parameters using Monolix. After estimation, the output from Monolix is converted to a Standard Output object which is saved in a .SO.xml file.
Translated files and Monolix output will be returned in the ./Monolix subfolder. The Standard Output object (.SO.xml) is read and parsed into an R object called “MLX” of (S4) class “StandardOutputObject”.

3
Figure 2: Exploratory Analysis
MLX <- `estimate`(mdlfile.MLX, target="MONOLIX", subfolder="Monolix")

```
# -- Thu Oct 20 14:20:46 2016
# New
# Submitted
# Job 939bff07-38b7-4450-8cdd-5da50af73da4 progress:
# Running [ ...... ]
# Importing Results
# Copying the result data back to the local machine for job ID 939bff07-38b7-4450-8cdd-5da50af73da4...
# From C:\Users\smith_mk\AppData\Local\Temp\3\Rtmp0kZiHR\DDMORE.job4a50616175fd to C:/SEE/MDL_IDE/workspace/Demo_Simeoni/models/Monolix
# Done.
#
# The following main elements were parsed successfully:
# ToolSettings
# RawResults
# Estimation::PopulationEstimates::MLE
# Estimation::PrecisionPopulationEstimates::MLE
# Estimation::IndividualEstimates::Estimates
# Estimation::IndividualEstimates::RandomEffects
# Estimation::Residuals::ResidualTable
# Estimation::Predictions
# Estimation::OFMeasures::IndividualContribToLL
# Estimation::OFMeasures::InformationCriteria
# Estimation::OFMeasures::LogLikelihood
#
# Completed
# -- Thu Oct 20 14:22:49 2016
```

```
slotNames(MLX)
```

```
# [1] "ToolSettings" "RawResults" "TaskInformation"
# [4] "Estimation" "ModelDiagnostic" "Simulation"
# [7] "OptimalDesign" ".pathToSourceXML"
```

**Retrieve estimated parameters**

The `ddmore` function `getPopulationParameters` extracts the Population Parameter values from an R object of (S4) class “StandardOutputObject” and returns the MLE estimates. See documentation for `getPopulationParameters` to see other arguments and settings for this function.

The Standard Output Object (SO) is an XML file containing output from modelling and simulation software tools and defines a standard for the output. R reads this output and parses the XML into an R object which can then be interrogated using functions in the `ddmore` R package. The benefit of this is that the output is consistent regardless of which tool has been used. This means that downstream activities can be consistent and need not change if we swap from one software target tool to another.

```
getPopulationParameters(MLX)$MLE
```

```
# Warning in getMLEPopulationParameters(SOObject, what = what): Tried to fetch the parameter interval values, however section Estimation::PrecisionPopulationEstimates::MLE::AsymptoticCI was not found in the SO Object.
# Omitting interval values for MLE section in returned output.
# Parameter MLE SE RSE
# 1 ADD 0.00000 0.00000 0.00
# 2 CV 0.10355 0.01776 17.15
```
Perform basic model diagnostics

Use `ddmore` function as `as.xpdb` to create an Xpose database object from an R object of (S4) class “Standard-OutputObject”. We can then call Xpose functions referencing this mlx.xpdb object as the input.

The Xpose package is typically only used with NONMEM estimation for model diagnostics, but thanks to the DDMoRe Standard Output we can convert to the relevant inputs for Xpose and use this package with any modelling tool. This has clear benefits for interoperability, consistency and reproducibility of workflow.

```r
# Removed dose rows in rawData slot of SO to enable merge with Predictions data.
print(basic.gof(mlx.xpdb))

print(ind.plots(mlx.xpdb, layout=c(1,2)))

ESTIMATE model parameters in NONMEM

To illustrate interoperability we can also estimate the model parameters with NONMEM. To do so we only need to change the Task Properties Object in the MOG to use the relevant estimation settings for NONMEM. All other MDL objects remain the same i.e. the Data Object, (initial) Parameters for estimation and Model Object are all unchanged.

```r
NM.MOG <- createMogObj(dataObj = myDataObj,
                        parObj = myParameterObj,
                        mdlObj = myModelObj,
                        taskObj = myTaskPropertiesObj_NM)
```
Figure 3: Basic Goodness of Fit following Monolix Estimation
Figure 4: Individual Plots following Monolix Estimation
The `ddmore estimate` function translates the contents of the .mdl file to NMTRAN and then estimates parameters using NONMEM. After estimation, the output from NONMEM is converted to a Standard Output object which is saved in a .SO.xml file.

Translated files and NONMEM output will be returned in the ./NONMEM subfolder. The Standard Output object (.SO.xml) is read and parsed into an R object called “NM” of (S4) class “StandardOutputObject”.

```r
NM <- estimate(mdlfile.NM, target="NONMEM", subfolder="NONMEM")
```

Once the model has run and the Standard Output (SO) object has returned it is possible to read that object in to R at any time.

```r
# NM <- LoadSOObject("NONMEM/Simeoni_PAGE_Estimation_NM.SO.xml")
```

Results from NONMEM should be comparable to previous results

```r
parameters_nm <- getPopulationParameters(NM, what="estimates")$MLE
parameters_nm
```

```
## CV LAMBDA0_POP LAMBDA1_POP K1_POP K2_POP W0_POP
## 0.1003870 0.2987270 0.7741350 0.7866220 0.7147320 0.0421836
## K10_POP K12_POP K21_POP V1_POP ADD
## 20.8320000 0.1440000 2.0110000 0.8100000 0.0000000
```
Xpose diagnostics using NONMEM output

```r
nm.xpdb <- as.xpdb(NM, datafile)
```

```r
##
## Removed dose rows in rawData+Predictions slot of SO to enable merge with Residuals data.
##
## Residuals data does not currently contain dose rows in output from Nonmem executions.
##
## Warning in as.data(SOObject, inputDataPath): No
## Estimation::IndividualEstimates::RandomEffects::EffectMean found in the
## SO; the resulting data frame will not contain these
```

Basic diagnostics for NONMEM fit.

```r
print(basic.gof(nm.xpdb))
```

```r
nm.xpdb@Data$ID <- as.numeric(as.character(nm.xpdb@Data$ID))
```

```r
print(ind.plots(nm.xpdb, layout=c(1, 2)))
```

**ESTIMATE model parameters in WINBUGS**

Assembling the new MOG for Winbugs. Note that we reuse the data and model. Two main changes are made: selecting the appropriate Task Properties Object for BUGS and exchanging the Prior Object instead of the Parameter Object. In Table 1 this is the second item. Separating the prior specification from the model means that the Model Object is consistent regardless of estimation method. The Prior Object must provide distributions or fixed values for every parameter in the model. The prior distributions then form the top level of the model random effect hierarchy.

```r
BUGS.MOG <- createMogObj(dataObj = myDataObj,
priorObj = myPriorObj,
mdlObj = myModelObj,
taskObj = myTaskPropertiesObj_BUGS)
```

We can then write the MOG back out to an .mdl file.

```r
mdlfile.BUGS <- paste0(model, ".BUGS.mdl")
writeMogObj(BUGS.MOG, mdlfile.BUGS)
#mdlfile.BUGS <- "Simeoni_PAGE_Estimation_BUGS.mdl"
```

Winbugs converter and function call has not been integrated into the SEE yet, so we cannot run:

```r
# estimate(mdlfile.BUGS, target="BUGS")
```

in the meantime, an R script has been prepared to call the converter, generate Winbugs input. This involves conversion of the NONMEM format dataset to BUGS (R list) format and preparation of the model file, data file, initial values for the MCMC and creation of the script file for running BUGS in batch mode.

```r
BUGS <- runWinBUGS(mdlfile.BUGS, subfolder="BUGS")
```

```r
## -- Thu Oct 20 14:23:35 2016
## New
## Warning in NMTRAN2BUGSdataconverter(model = model.pharmml): rate column is
## missing. rate will be set to 0 by default.
## Warning in NMTRAN2BUGSdataconverter(model = model.pharmml): ii column is
```
Figure 5: Basic Goodness of Fit following NONMEM Estimation
Figure 6: Individual Plots following NONMEM Estimation
## missing. Default 0 values will be set.
## Warning in NMTRAN2BUGSdataconverter(model = model.pharmml): cmt column is
## missing. cmt will be set to 1 by default.
## Warning in NMTRAN2BUGSdataconverter(model = model.pharmml): addl column is
## missing. addl will be set to 0 by default.
## Warning in NMTRAN2BUGSdataconverter(model = model.pharmml): ss column is
## missing. ss will be set to 0 by default.

## Submitted
## Job 23296422-35a7-4efe-8614-622884893872 progress:
## Running [ ............ ]
## Importing Results
## Copying the result data back to the local machine for job ID 23296422-35a7-4efe-8614-622884893872...
## From C:\Users\smith_mk\AppData\Local\Temp\3\RtmpOkZiHR\DDMORE.job4a50516c1001 to C:/SEE/MDL_IDE/workspace/Demo_Simeoni/models/BUGS
## Done.

## The following main elements were parsed successfully:
## RawResults
## Estimation::PopulationEstimates::Bayesian
## Estimation::PrecisionPopulationEstimates::Bayesian
## Estimation::IndividualEstimates::Estimates
## Estimation::PrecisionIndividualEstimates::StandardDeviation
## Estimation::PrecisionIndividualEstimates::EstimatesDistribution
## Estimation::PrecisionIndividualEstimates::PercentilesCI

## The following MESSAGEs were raised during the job execution:
## winbugs_message: success

## Completed
## -- Thu Oct 20 14:27:40 2016

Retrieve parameter estimates

```
parameters_BUGS<- getPopulationParameters(BUGS, what="estimates")$Bayesian
estPars <- intersect(rownames(parameters_BUGS), names(parameters_nm))
```

coda MCMC trace and density plots

```
#to read only one chain
library(coda)
BUGSOutputPath <- file.path(getwd(),"BUGS")
coda_out <- read.coda(output.file=file.path(BUGSOutputPath,"output1.txt"), index.file=file.path(BUGSOutputPath,"outputIndex.txt"), quiet=T)
coda_pars <- coda_out[,estPars]
```

Using the coda package we can look at many different MCMC diagnostics to assess MCMC convergence. Many of these only work for >1 chain and so are not shown here.

```
#to read more chains (3 in this case) use read.coda.interactive()
coda_out <- read.coda.interactive()
# Enter in the order:
#outputIndex.txt
#output1.txt
#output2.txt
#output2.txt
```

13
Summary statistics of parameters from the MCMC chain.

```
summary(coda_pars)
```

```
##
## Iterations = 4001:14000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
## plus standard error of the mean:
##
##          Mean SD Naive SE Time-series SE
## CV       0.06321 0.002261 2.261e-05 2.261e-05
## K1_POP   0.79577 0.094613 9.461e-04 1.123e-02
## K2_POP   0.71435 0.039297 3.930e-04 1.014e-02
## LAMBDA0_POP 0.29844 0.013611 1.361e-04 4.094e-03
## LAMBDA1_POP 0.78097 0.033654 3.365e-04 2.175e-03
## W0_POP   0.04313 0.006010 6.010e-05 1.672e-03
##
## 2. Quantiles for each variable:
##
##           2.5%   25%        50%      75%    97.5%
## CV       0.05893 0.06167  0.063140 0.06471 0.06779
## K1_POP   0.63390 0.72960  0.786000 0.85300 1.00100
## K2_POP   0.64150 0.68750  0.716100 0.73990 0.79950
## LAMBDA0_POP 0.27200 0.28900  0.299700 0.30700 0.32550
## LAMBDA1_POP 0.71980 0.75840  0.778600 0.80200 0.85040
## W0_POP   0.03260 0.03908  0.042040 0.04724 0.05587
```

Trace plots to assess MCMC convergence and density plots to show the posterior density of parameters.

```
plot(coda_pars, ask=F)
```
Compare estimated parameters in the 3 tools

Since all three methods have returned consistent output formats, it is easy to compare the parameter estimates across the three estimation methods.

In this instance, since we are modelling the mean of the observations at each time point, with no inter-individual random effect, the SAEM method is not really an appropriate estimation algorithm. This explains some of the discrepancy between Monolix (MLX in the table below) and the other tools for certain parameters.
estPars <- intersect(rownames(parameters_BUGS), names(parameters_nm))

cbind(BUGS=parameters_BUGS[estPars,"Median"],
      NM=parameters_nm[estPars],
      MLX=parameters_mlx[estPars])

## BUGS NM MLX
## CV 0.06314 0.1003870 0.10355
## K1_POP 0.78600 0.7866220 0.67638
## K2_POP 0.71610 0.7147320 0.78889
## LAMBDA0_POP 0.29970 0.2987270 0.32162
## LAMBDA1_POP 0.77860 0.7741350 0.76213
## W0_POP 0.04204 0.0421836 0.03310

VPC of model

A further model diagnostic that is commonly used for assessing the model fit is a Visual Predictive Check or VPC. PAGE 17 (2008) Abstr 1434 To perform a VPC we take the estimated parameter values and simulate new observations from the model using these values. The simulated values are then summarised and compared against the observed values. If the model captures both the central tendency and the spread of data (inter-individual and intra-individual) then this helps in assessing the overall picture of model qualification.

First, we update the (initial) values in the MDL Parameter Object Extract the structural and variability parameters from the SO object following NONMEM estimation.

structuralPar <- getPopulationParameters(NM, what="estimates",block='structural')$MLE

variabilityPar <- getPopulationParameters(NM, what="estimates",block='variability')$MLE

Update the parameter object using the ddmore updateParObj function. This function updates an R object of (S4) class “parObj”. The user chooses which block to update, what items within that block, and what to replace those items with. NOTE: that updateParObj can only update attributes which ALREADY EXIST in the MDL Parameter Object for that item. This ensures that valid MDL is preserved.

myParObjUpdated <- updateParObj(myParameterObj,block="STRUCTURAL",
                                 item=names(structuralPar),
                                 with=list(value=structuralPar))

myParObjUpdated <- updateParObj(myParObjUpdated,block="VARIABILITY",
                                 item=names(variabilityPar),
                                 with=list(value=variabilityPar))

Check that the appropriate initial values have been updated to the MLE values from the previous fit.

# print(myParObjUpdated@STRUCTURAL)
# print(myParObjUpdated@VARIABILITY)

Assembling the new MOG. Note that we reuse the data, model and tasks from the previous NONMEM run. This is item 3 from Table 1.

VPC.MOG <- createMogObj(dataObj = myDataObj,
                          parObj = myParObjUpdated,
                          mdlObj = myModelObj,
                          taskObj = myTaskPropertiesObj_NM)

We can then write the MOG back out to an .mdl file.

mdlfile.VPC <- paste0(model,"_VPC.mdl")
writeMogObj(VPC.MOG,mdlfile.VPC)
Similarly as above, `ddmore` R package `VPC.PsN` function can be used to run a VPC using PsN as target tool. PsN then simulates observations using a converted NMTRAN version of the model.

```r
vpcFiles <- VPC.PsN(mdlfile.VPC, samples=200, seed=12345,
                      vpcOptions ="-n_simulation=10 -auto_bin=8, -min_points_in_bin=2",
                      subfolder="VPC", plot=FALSE)
```

```
## -- Thu Oct 20 14:27:47 2016
## New
## Submitted
## Job 5a7d92a1-ec77-4bac-a3d5a100fe8e progress:
## Running [ ... ]
## Importing Results
## Copying the result data back to the local machine for job ID 5a7d92a1-ec77-4bac-a3d5a100fe8e... 
## From C:\Users\smith_mk\AppData\Local\Temp\3\RtmpOkZiHR\DDMORE.job4a5012395402 to C:/SEE/MDL_IDE/workspace/Demo_Simeoni/models/VPC 
## Done.
##
## The following main elements were parsed successfully:
##  RawResults
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##  SimulationSimulationBlock
##
## The following MESSAGEs were raised during the job execution:
##  nmoutput2so_version: This SOBlock was created with nmoutput2so version 4.5.27
##
## Completed
## -- Thu Oct 20 14:28:50 2016
```

To replay the visualisation using information from the VPC SO file:

```r
plot(xpose.VPC(vpc.info= file.path("./VPC","vpc_results.csv"),
               vpctab= file.path("./VPC","vpctab"),
               main="VPC TGI model"))
```

Note that because we are modelling naive-pooled data, the VPC cannot distinguish individual and treatment...
Figure 7: VPC using NONMEM estimates
level data. As such, the VPC is not particularly helpful in this case for model qualification.

Simulation using simulx

The (mlxR package)[http://simulx.webpopix.org/mlxr/] has been developed to visualize and explore models that are encoded in MLXTRAN or PharmML. Here we can use the function simulx within the mlxR package to simulate or predict new outcomes and evaluate the effect of changes in dosage regimen, dose times, observation times, or other changes to the model inputs. This corresponds to item 4 of Table 1.

The ddmor function as.PharmML translates an MDL file (extension .mdl) to its PharmML representation. The output file (extension .xml) is saved in the working directory.

In this instance we do not need to write out the updated MOG to a file, since we can work directly with the mlxR R package which takes PharmML as input. We can specify the design elements (inputs to simulx) directly within R and there is no Task Properties Object, again because we are working directly within R.

In future we may provide a ddmor R package function as.simulx which would take an MDL Design Object within a MOG and produce all of the required inputs for simulx. This would allow consistency of definition of inputs regardless of whether the tool is R-based or external to R.

```r
myPharmML <- as.PharmML(mdlfile.NM)
```

Use parameter values from the FOCE estimation performed with NONMEM using the NM object representation of the SO.

```r
p <- getPopulationParameters(NM, what="estimates")$MLE
print(p)
## CV LAMBDA0_POP LAMBDA1_POP K1_POP K2_POP W0_POP
## 0.1003870 0.2987270 0.7741350 0.7866220 0.7147320 0.0421836
## K10_POP K12_POP K21_POP V1_POP ADD
## 20.8320000 0.1440000 2.0110000 0.8100000 0.0000000
```

#p <- c(V1_POP=0.8, K10_POP=0.9, K21_POP=0.08, K12_POP=0.006,
# PSI_POP=20, LAMBDA0_POP=0.3, LAMBDA1_POP=0.7, K1_POP=1, K2_POP=0.5, W0_POP=0.02,
# CV=0.1)

First we need to set up the appropriate inputs for the simulx function:

adm : dose administration times

```r
adm <- list( time = c(8, 12, 16),
             amount = 30,
             target = 'Q1')
```

f : what to sample (“observe”) and at what times?

```r
f <- list( name='WTOT', time=seq(0,45,by=0.1))
```

g : define group sizes and assign treatments.

```r
g0 <- list(size=1, level='longitudinal')
g1 <- list(size=1, level='longitudinal', treatment=adm)
```

Call simulx. simulx can take PharmML as an input - so the model does NOT need to be translated to MLXTRAN.

```r
res <- simulx( model = myPharmML,
               parameter = p,
               ...)
```
Plot simulx output

```r
ggplot() +
  geom_line(data=res$WTOT, aes(x=time, y=WTOT, colour=id)) +
  scale_colour_discrete(name="Experimental condition",
                        breaks=c("1", "2"),
                        labels=c("Control", "Treatment"))
```

Figure 8: Plot of simulated data using NONMEM estimates
Optimal Design

To evaluate optimal sampling times for a future study, for a given model and set of parameter values, we must specify a Design Object, provide Parameter values in a Parameter Object (in this case using the estimated parameters from NONMEM) and specify task properties appropriate for the optimal design task and the target software tool in the Task Properties Object. This corresponds to item 5 of Fig.2.

Here we reuse the updated Parameter Object myParObjUpdated that was previously created for use with the VPC.

Note that due to current limitations in PFIM, evaluation and optimisation of multiple doses (as in the Simeoni estimation model) is not straightforward to encode. For this demonstration, we will examine one single dose at time 8 (days) so that the tumour has time to grow before treatment, so that we can see the effect of treatment.

Although the multiple dose design is not supported by PFIM, the model stays the same and we can update the parameter values used for evaluating the study design using the results of estimation using NONMEM and use simulation results to provide the prediction of the total tumour weight (WTOT) at Day 8. Update W0 in the parameter object with simulated W0 at day 8 based on the simulx simulations above.

```r
simdata <- res$WTOT
W0_8d <- simdata[simdata$time>7.9 & simdata$time<8.02 & simdata$id==2,"WTOT"]

myParObjUpdated_Design <- myParObjUpdated
myParObjUpdated_Design <- updateParObj(myParObjUpdated_Design, block="STRUCTURAL", item="W0_POP", with=list(value=W0_8d))
```

Assembling the new MOG.

```r
OptDes.MOG <- createMogObj(designObj = myDesignObj, parObj = myParObjUpdated_Design, mdlObj = myModelObj, taskObj = myTaskPropertiesObj_OptDes)
```

We can then write the MOG back out to an .mdl file.

```r
mdlfile.OptDes <- "Simeoni_Demo_Evaluation_OptDes.mdl"
writeMogObj(OptDes.MOG ,mdlfile.OptDes)
```

Design evaluation using PFIM

Again, as for BUGS, the PFIM converter and connector have not yet fully been integrated into the SEE environment. Here we call a small script which calls the converter which creates the required PFIM input R scripts.

The Design Object settings and Task Properties settings dictate whether evaluation or optimisation of the trial design is performed. #source("./scripts/callPFIMconverter_MKS_v2.R") The runPFIM function takes an MDL file, converts to PharmML, runs the converter and then runs a created batch script to run PFIM.

```r
as.PharmML(mdlfile.PFIM)
```

```r
## Error in file.exists(f): object 'mdlfile.PFIM' not found
runPFIM(mdlfile.PFIM, jarLocation=file.path(DDMORE.checkConfiguration(),"PFIM","converter"))
```

```r
## Error in ifelse(!is.null(mdlfile) & is.null(pharmmlfile), as.PharmML(mdlfile), : object 'mdlfile.PFIM'
```
## 1. "PFIM 4.0"
## 2. "Project: Generated from MDL. MOG ID: outputMog"
## 3. "Date: Wed Oct 19 13:49:36 2016"
## 4. ""
## 5. ""
## 6. ""
## 7. ""
## 8. ""
## 9. "************************** INPUT SUMMARY **************************"
## 10. ""
## 11. "Differential Equations form of the model:
## 12. "function (t, y, p) {
## 13. "  LAMBDA0 <- p[1]
## 14. "  LAMBDA1 <- p[2]
## 15. "  K1 <- p[3]
## 16. "  K2 <- p[4]
## 17. "  W0 <- p[5]
## 18. "  K10 <- p[6]
## 19. "  K12 <- p[7]
## 20. "  K21 <- p[8]
## 21. "  V1 <- p[9]
## 22. "  PSI <- 20
## 23. "  C <- ((y[1])/(V1))
## 24. "  WTOT <- y[3] + y[4] + y[5] + y[6]
## 25. "  yd1 <- ((K21) * (y[2])) - ((K10 + K12) * (y[1]))
## 26. "  yd2 <- ((K12) * (y[1])) - ((K21) * (y[2]))
## 27. "  yd3 <- (((LAMBDA0) * (y[3]))) / (((1 + (((((WTOT) * (LAMBDA0)))/(LAMBDA1)))^((PSI)))^((1/(PSI))))
## 28. "  yd4 <- (((K2) * (C))) * (y[3])
## 29. "  yd5 <- ((K1) * (y[4])) - ((K1) * (y[5]))
## 30. "  yd6 <- ((K1) * (y[5])) - ((K1) * (y[6]))
## 31. "  return(list(c(yd1, yd2, yd3, yd4, yd5, yd6, WTOT)))
## 32. "}
## 33. ""
## 34. "Design:
## 35. "Sample times for response: A 1 c(0, 4, 36, 40)
## 36. "2 c(0, 20, 55, 60)
## 37. "Initial Conditions at time 0:
## 38. "W0 0 0 0 0 W0 0 0 0 0 WTOT"
## 39. "Variance error model response A: ( 0 + 0.100387 *f)^2"
## Error tolerance for solving differential equations system: RtolEQ = 1e-08, AtolEQ = 1e-08, Hmax = Inf

### Computation of the Individual Fisher information matrix

FIM saved in FIM.txt

### Fisher Information Matrix

|   | [,1]      | [,2]      | [,3]      | [,4]      | [,5]      | [,6] |
|---|-----------|-----------|-----------|-----------|-----------|------|
| 1 | 3634.01035| 1.070042e+03 | -4.299685e+01 | -1092.60364 | 430.6468  | 0.000|
| 2 | 1070.04244| 6.218607e+02 | -2.229345e-04 | -271.33133 | 157.3159  | 0.000|
| 3 | -42.99685 | -2.229345e-04| 4.204643e+02 | 66.43736   | -442.5852 | 0.000|
| 4 | -1092.60364| -2.713313e+02| 6.643736e+01 | 157.3159  | -173.0523 | 0.000|
| 5 | 430.64677 | 1.573159e+02 | -4.425852e+02| -173.05226| 1449.4261 | 0.000|
| 6 | 0.00000  | 0.000000e+00| 0.000000e+00| 0.00000   | 0.00000   | 1587.688|

### Expected Standard Errors

|------------------------ Fixed Effects Parameters ------------------------- |
|------------------------ |------------------|------------------|
| Beta                  | StdError         | RSE              |
| LAMBDA0               | 0.2987270        | 0.14591461       | 48.845471 % |
| LAMBDA1               | 0.7741350        | 0.08728178       | 11.274750 %|
| K1                    | 0.7866220        | 0.08001167       | 10.171552 %|
| K2                    | 0.7147320        | 0.41557182       | 58.143726 %|
| W0                    | 0.4615215        | 0.03276079       | 7.098433 % |

|------------------------ Standard deviation of residual error ------------------------ |
|------------------------ |------------------|------------------|
| Sigma                 | StdError         | RSE              |
| sig.slopeA            | 0.100387         | 0.02509675       | 25 %        |

### Determinant

4.025483e+15

### Criterion

398.8439

### Eigenvalues of the Fisher Information Matrix

| FixedEffects | VarianceComponents |
|--------------|--------------------|
| min          | 242.63930          | NA |
| max          | 4380.96227         | NA |
| max/min      | 18.05545           | NA |
Design evaluation using PopED

As with the estimation task, we can easily switch from one optimal design tool to another using the same inputs. Here we take the MDL file which was written out for use with PFIM and convert it to PharmML so that it can be used with the PopED optimal design package. As with simulx package, PopED can take PharmML as the input directly. In future, if PharmML is adopted as a model input or import format (as with SBML) then the only conversion that may be required is from the user-centric MDL to the software interchange standard of PharmML.

The PopED function as.poped takes a PharmML file and creates a poped.db database object ready for use with the package. We can then use PopED functions directly (natively) in R as we did with mlxR / simulx.

```r
library(PopED)

pharmMLFile <- as.PharmML(mdlfile.OptDes)
as.poped(pharmMLFile)
```

```r
## Warning: PopED Warning: No PopED operation algorithm found in design step
## at line 619
## Warning: PopED Warning: No PopED-specific settings could be retrieved from
## modelling steps at line 531
```

create plot of model without variability

```r
plot_model_prediction(poped.db)
```

get predictions from model

```r
model_prediction(poped.db)
```

```r
## Time PRED Group Model DOSE_1_AMT DOSE_1_TIME
## 1 0 0.4615215 1 1 120 0
## 2 20 1.1257519 1 1 120 0
## 3 55 27.5144022 1 1 120 0
## 4 60 31.3850772 1 1 120 0
## 5 0 0.4615215 2 1 0 0
## 6 36 25.9778569 2 1 0 0
## 7 4 1.5245292 2 1 0 0
## 8 40 29.0743969 2 1 0 0
```

evaluate initial design
Figure 9: Plot model predictions from PopED
FIM <- \texttt{evaluate.fim(poped.db)}

FIM

```
## [,1]         [,2]      [,3]        [,4]     [,5]        [,6]
## [1,]  1587.687  575.897   93.532   -147.686   189.682
## [2,]   575.897 42383.32   601.294   8545.931 -12965.8  5389.813
## [3,]   93.532  601.294   557.815    81.839 -127.44  98.457
## [4,]  -147.687 -12965.8 -127.440  -2723.73  4132.15 -1430.37
## [5,]   189.682  5389.8  98.457   942.787 -1430.37  1929.52
```

det(FIM)

```
# [1] 4.691807e+13
```

\texttt{get_rse(FIM,poped.db)}

```
## bpop[2] bpop[3] bpop[4] bpop[5] bpop[6] bpop[7]
##  25.250697  10.185718 1377.224210  998.829462   7.045583
```

**MDL Objects - R representation**

\texttt{getModelObjects} and similar functions return an R object representation of MDL. This can then be edited and manipulated using R and the results written back to MDL. Here’s what the Model Object looks like in R.

When working with models in R (using R packages) the package developer can opt to work directly with the R representation of the MDL as seen below or can use the DDMoRe software interchange standard PharmML. If using the former, the user must write their own conversion rules for handling certain constructs such as dosing inputs or interpreting the MDL compartment constructs. However some development has already been done to handle these in a consistent manner via PharmML converter technology.

\texttt{myModelObj}

```
# An object of class "mdlObj"
# Slot "IDV":
# [1] "T"
#
# Slot "COVARIATES":
# list()
#
# Slot "VARIABILITY_LEVELS":
# [[1]]
# [[1]]$DV
# [[1]]$DV$level
# [1] "1"
#
# [[1]]$DV$type
# [1] "observation"
#
# Slot "STRUCTURAL_PARAMETERS":
# [1] "LAMBDA0_POP" "LAMBDA1_POP" "K1_POP" "K2_POP" "W0_POP"
# [6] "K10_POP" "K12_POP" "K21_POP" "V1_POP"
```
## Slot "VARIABILITY_PARAMETERS":
## [1] "ADD" "CV"

## Slot "RANDOM_VARIABLE_DEFINITION":
## [[1]]
## [[1]]$subtype
## [1] "RandVarDefn"
## [[1]]$blkAttrs
## [[1]]$blkAttrs$level
## [1] "DV"
##
## [[1]]$name
## [1] "eps_RES_W"
##
## [[1]]$distn
## [1] "Normal(mean=0, var=1)"
##
## Slot "INDIVIDUAL_VARIABLES":
## [[1]]
## [[1]]$subtype
## [1] "EquationDef"
##
## [[1]]$name
## [1] "LAMBDA0"
##
## [[1]]$expr
## [1] "LAMBDA0_POP"
##
## [[2]]
## [[2]]$subtype
## [1] "EquationDef"
##
## [[2]]$name
## [1] "LAMBDA1"
##
## [[2]]$expr
## [1] "LAMBDA1_POP"
##
## [[3]]
## [[3]]$subtype
## [1] "EquationDef"
##
## [[3]]$name
## [1] "K1"
##
## [[3]]$expr
## [1] "K1_POP"
## EquationDef

### K2

```text
K2
```

### W0

```text
W0
```

### K10

```text
K10
```

### K12

```text
K12
```

### K21

```text
K21
```
## Slot "MODEL_PREDICTION":

```r
[[1]]
[[1]]$DEQ
[[1]]$DEQ[[1]]
[[1]]$DEQ[[1]]$.subtype
[[1]] "EquationDef"

[[1]]$DEQ[[1]]$name
[[1]] "PSI"

[[1]]$DEQ[[1]]$expr
[[1]] "20"

[[1]]$DEQ[[2]]
[[1]]$DEQ[[2]]$.subtype
[[1]] "EquationDef"

[[1]]$DEQ[[2]]$name
[[1]] "C"

[[1]]$DEQ[[2]]$expr
[[1]] "Q1/V1"

[[1]]$DEQ[[3]]
[[1]]$DEQ[[3]]$.subtype
[[1]] "EquationDef"

[[1]]$DEQ[[3]]$name
[[1]] "WTOT"

[[1]]$DEQ[[3]]$expr
[[1]] "X1+X2+X3+X4"

[[1]]$DEQ[[4]]
[[1]]$DEQ[[4]]$Q1
[[1]]$DEQ[[4]]$Q1$deriv
[[1]] "K21*Q2-(K10+K12)*Q1"
```
## $\text{DEQ}[4]$ $\text{Q}_1$ \text{init}
## [1] "0"
##
##
## 
## $\text{DEQ}[5]$
## 
## $\text{DEQ}[5] \text{Q}_2$
## 
## $\text{DEQ}[5] \text{Q}_2$ deriv
## [1] "K_{12} \text{Q}_1-K_{21} \text{Q}_2"
##
## $\text{DEQ}[5] \text{Q}_2$ init
## [1] "0"
##
##
##
## $\text{DEQ}[6]$
## 
## $\text{DEQ}[6] \text{X}_1$
## 
## $\text{DEQ}[6] \text{X}_1$ deriv
## [1] "(\text{LAMBDA}_0 \text{X}_1/((1+(\text{WTOT} \text{LAMBDA}_0/\text{LAMBDA}_1)^\text{PSI})^\text{PSI}))-K_2 \text{C} \text{X}_1"
##
## $\text{DEQ}[6] \text{X}_1$ init
## [1] "W_0"
##
##
##
## $\text{DEQ}[7]$
## 
## $\text{DEQ}[7] \text{X}_2$
## 
## $\text{DEQ}[7] \text{X}_2$ deriv
## [1] "K_2 \text{C} \text{X}_1-K_1 \text{X}_2"
##
## $\text{DEQ}[7] \text{X}_2$ init
## [1] "0"
##
##
##
## $\text{DEQ}[8]$
## 
## $\text{DEQ}[8] \text{X}_3$
## 
## $\text{DEQ}[8] \text{X}_3$ deriv
## [1] "K_1 \text{X}_2-K_1 \text{X}_3"
##
## $\text{DEQ}[8] \text{X}_3$ init
## [1] "0"
##
##
##
## $\text{DEQ}[9]$
## 
## $\text{DEQ}[9] \text{X}_4$
## 
## $\text{DEQ}[9] \text{X}_4$ deriv
## [1] "K_1 \text{X}_3-K_1 \text{X}_4"
##
## $\text{DEQ}[9] \text{X}_4$ init
## [1] "0"
## Slot "OBSERVATION":
[[1]]
[[1]]$Y
[[1]]$type
[[1]] "combinedError1"

[[1]]$additive
[[1]] "ADD"

[[1]]$proportional
[[1]] "CV"

[[1]]$eps
[[1]] "eps_RES_W"

[[1]]$prediction
[[1]] "WTOT"

## Slot "GROUP_VARIABLES":
list()