microsamplingDesign

Finding optimal microsampling designs for non-compartmental pharmacokinetic analysis

Adriaan Blommaert; Open Analytics

2018-05-03

Contents

```
knitr::opts_chunk$set(fig.width=12 )
```

1 Introduction

Microsampling, a novel blood sampling technique allows multiple blood samples to be taken per animal, reducing the number of animals required for pharmacokinetic-pharmacodynamic studies (Chapman et al. (2014)). Using sparce designs can in addition, avoid unnecessary sampling of these animals, provided an appropriate choice of sample times per animals is made. The microsamplingDesign package implements a general simulation methodology to find optimal sparse microsampling schemes aimed at non-compartmental pharmacokinetic analysis (algorithm III in Barnett et al. (2017)). This methodology consist of (1) specifying a pharmacokinetic model including variability among animals; (2) generating possible sampling times; (3) evaluating performance of each time point choice on simulated data; (4) generating possible schemes given a time point choice and additional constraints and finally (5) evaluating scheme perfomance on simulated data. The default settings differ from (Barnett et al. (2017))) in the default pharmacokinetic model used and the parameterization of variability among animals (see next section). A shiny web application is included, which guides users from model parametrization to optimal microsampling scheme.

2 Model details

A two compartmental oral dosing pharamcokinetic model (Gabrielsson and Weiner (2001)) is assumed:

$$\frac{dD_g}{dt} = -k_a.D_g$$

$$V_c\frac{dC}{dt} = F.k_a.D_g - Cl.C - Cl_d.C + Cl_d.C_t$$

$$V_t\frac{dC_t}{dt} = Cl_d.C - Cl_d.C_t$$

A dose of a substance (D_g) is administered to the gut, than graduadely absorbed into a central compartment leading to a increased concentration in the plasma (C). Where it can either be excreted or exchanged with a second peripheral compartment, the peripheral tissues, where the compound has a distinct concentration (C_t) in time, depending on the rate of exchange with the central compartment. We do not assume any excretion from the peripheral compartment.

Substance absorption and clearance are by default assumed to be capacity dependent (Michaelis-Menten kinetics):

$$k_a = \frac{V_{a,max}}{\kappa_{a,m} + D_g}$$

$$Cl = \frac{V_{e,max}}{\kappa_{e,m} + C}$$

We also leave the option open for one or both of these parameters to be constant.

For details see (Gabrielsson and Weiner (2001)).

2.1 Parametrization

- k_a is the absorption rate per unit of dose.
- V_c is the volume of the central compartment (plasma)
- V_t is the volume of the peripheral compartment (tissue)
- F bioavailability, the fraction of the dose that reaches the systemic circulation intact (dimensionless)
- Cl is the elimination rate from the central compartment (assumed the only spot where elimination occurs); in volume per time, related to the elimination rate in dose: ($k_e = \frac{Cl}{V_c}$)
- Cl_d is the distribution parameter between central and peripheral compartment; expressed in volume per time unit. It related to rates: $Cl_d = \frac{k_{ct}}{V_c} = \frac{k_{tc}}{V_t}$; with k_{ct} the rate from central to tissue (dose per time unit) , and k_{tc} the rate from tissue to central compartment.
- $V_{a,max}$ is the maximum absorption rate (absolute rate is rate per dose x dose)
- $\kappa_{a,m}$ is the Michaelis-Menten constant for absorption
- $V_{e,max}$ is the maximum clearance rate (absolute rate is rate per concentration x concentration)
- $\kappa_{e,m}$ is the Michaelis-Menten constant for clearance.

2.2 Log-normal parameters

Individual animals are assumed to have the same underlying model, with different parameters simulated from an underlying log-normal distribution parametrized in terms of the mean and the coefficient of variation.

we assume a random variable X to be log-normally distributed with parameters μ and σ:

$$X = \exp\big(\mu + \sigma Z\big)$$

with Z a standard normal variable.

Now, we want to extract μ and σ from and coefficient of variation ($CV = sd(X)/E(X)$) of the original scale.

we can use the relation for the mean:

$$E(X) = \exp\big(\mu + \frac{\sigma^2}{2}\big)$$

and the relation for the coefficient of variation:

$$CV(X) = \sqrt{\exp\sigma^2 - 1}$$

Therefore:

$$\sigma = \sqrt{ln(CV^2 + 1)}$$

and

$$\mu = \ln\big(E(X)\big) - \frac{\sigma^2}{2}$$

For the multivariate log-normal distribution, we use a the same approach per variable and can simulate a random vector:

$$\boldsymbol{X} = \exp\big(\boldsymbol{\mu} + Z\boldsymbol{\sigma}^{\boldsymbol{T}}\big)$$

with $Z \sim \mathcal{N}(\mathbf{0}, \Sigma)$ and Σ a specified correlation matrix. More information see (Halliwell (2015))

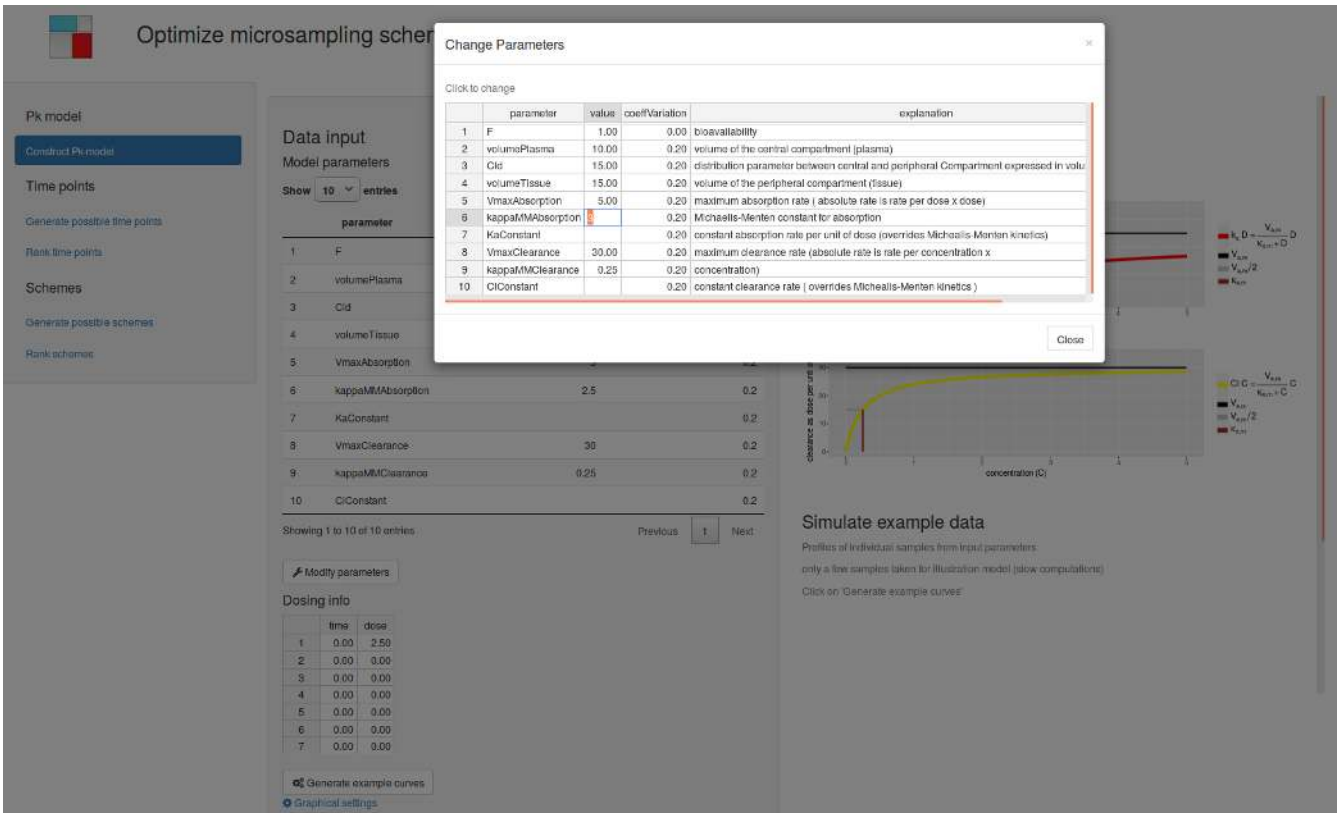


Figure 1: Construct a PK model

3 microsamplingDesign shiny application

Before diving into the R code of the microsamplingDesign package, we give a more intuitive introduction to the methodology using the included shiny application. In a local R session we can start the application:

```
library( microsamplingDesign )
runMicrosamplingDesignApp( installDependencies = TRUE )
```

The first time you want to run the application, use *installDependencies = TRUE* to automatically install the additional R-package required for this shiny application in addition to the microsamplingDesign package dependencies.

3.1 Construct a pharmacokinetic model

Start the application by constructing a pharacokinetic model.

Example parameters are shown on start up. To modify these parameters click on **Modify parameters** and a spreadsheet is displayed allowing modifying parameter values and their coefficient of variation (see Figure 1).

Next include dosing information by filling out one or several lines, click on **Generate example curves** to check simulated time-concentration curves (see Figure 2).

One can adapt the scale of the graphs by clicking on **Graphical settings**.

Note that the pharmacokinetic model in the application does not contain any measurement error.

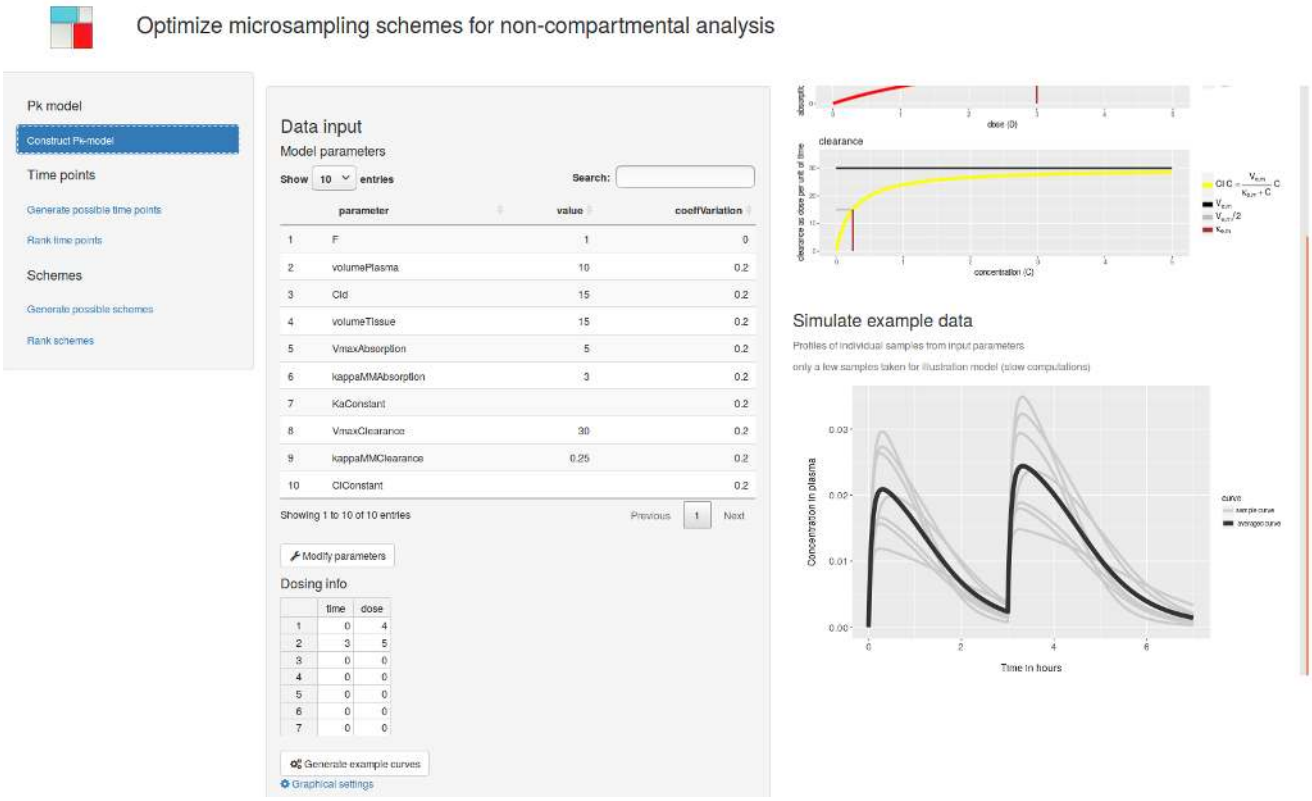


Figure 2: Check model by generating example curves

3.2 Generate possible time points

Time point options are generated from a time constraints table specifying the number of time points per time zone and minimum sampling interval in each row. Note that the endTime is not included in the zone itself but is the startTime of the next zone.

Finally click on the button **Generate time points**, to recieve all possible combinations in table form (see Figure 3).

3.3 Rank time points

Time points options are ranked by measuring the difference between approximating the average time-concentration curve based on a limited number of time points on sample data and the actual average curve at the maximal number of time points you want to consider. This is a measure of bias caused by choosing a certain time point option rather then sampling at the maximum number of time points.

In the application ranking time points takes 2 steps:

3.3.1 Generate sample data

Specify the approximate number of animals you would like to use in you scheme and the number of simulated datasets to generate. Then press **Generate data to rank time points**. A selection of simulated data will be displayed (see Figure 4).

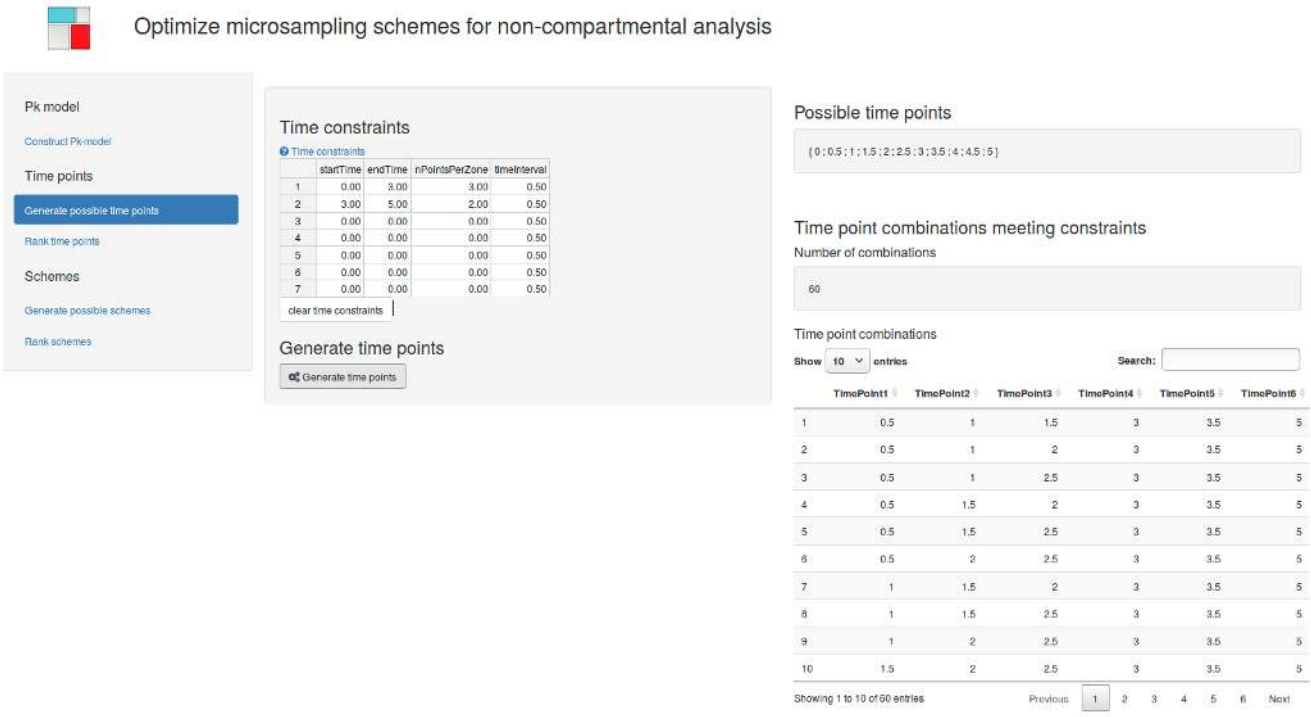


Figure 3: Generate time points

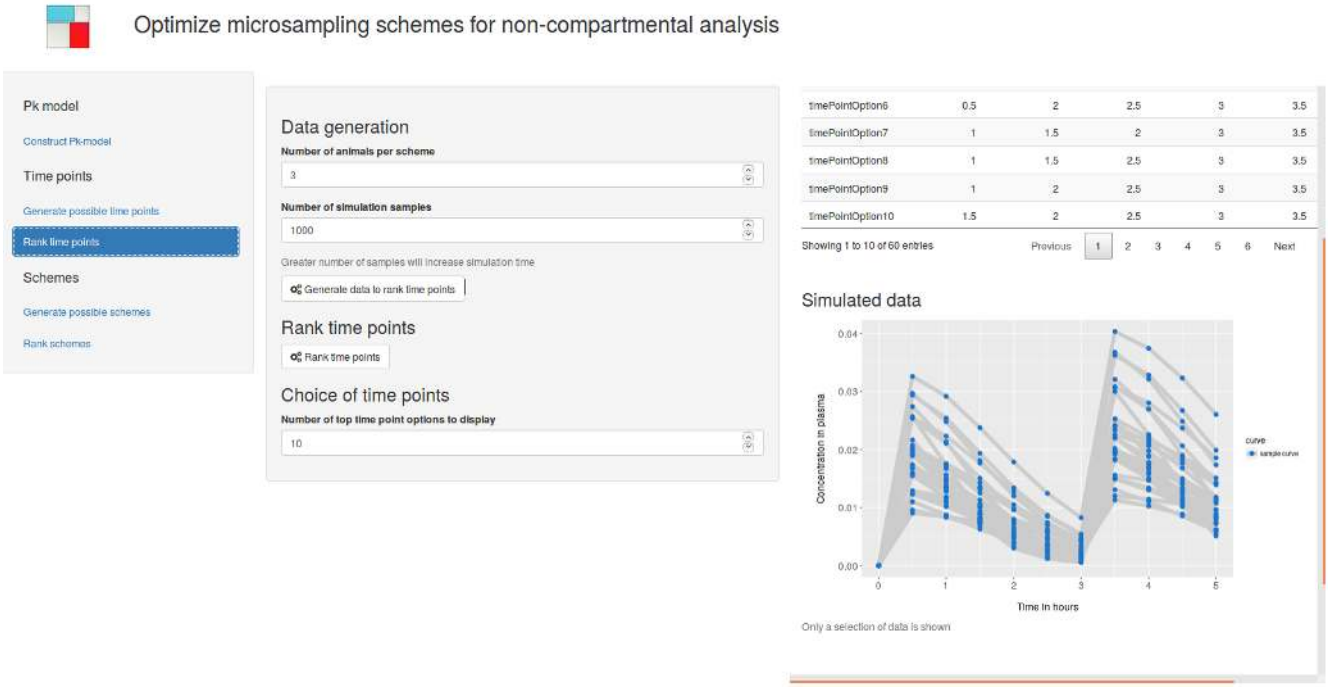


Figure 4: Generate data to rank time points

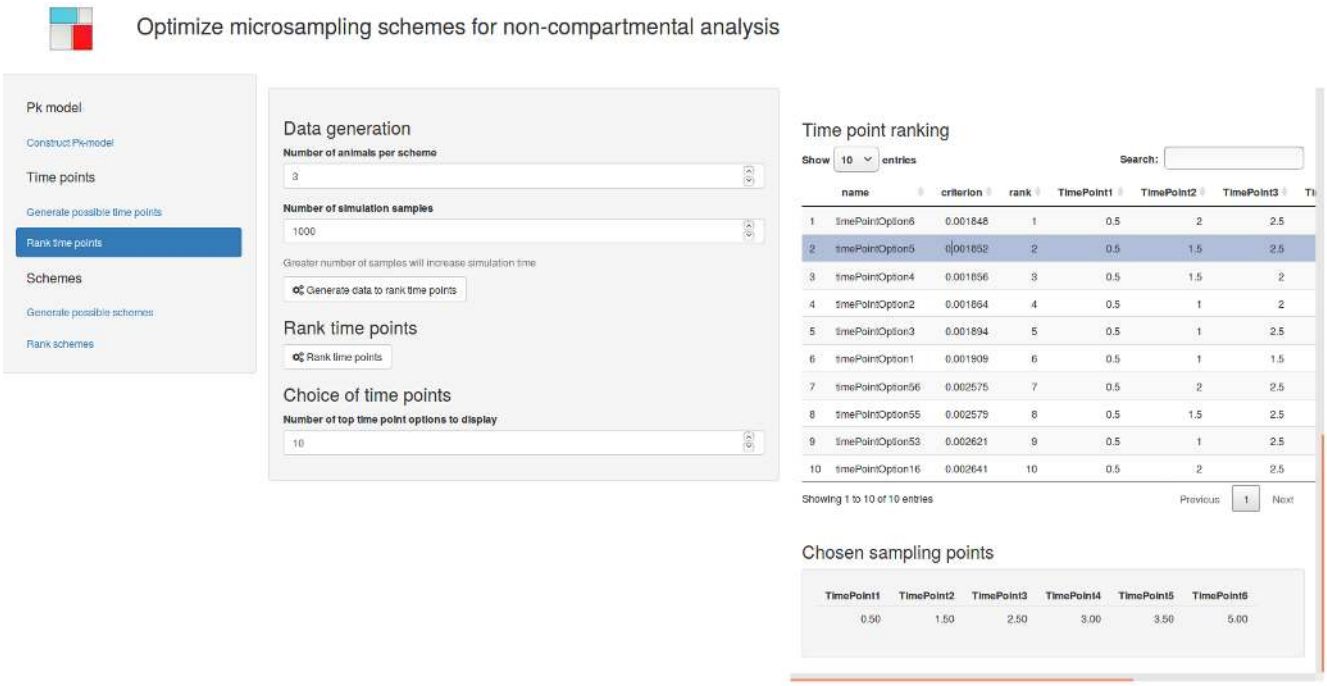


Figure 5: Rank time points and select one

3.3.2 Rank time points

After checking the generatated data, click on **Rank time points** to estimate the bias of each time point option. Calculations might take a few minutes, depending on the the number of simulation samples and time point options. When calculations are finished, time point options are tabulated from small to large deviation from the best accuracy. You can select a time point option by clicking on a row in the time point ranking table (see Figure 5).

3.4 Generate possible schemes

Given the time points, we will construct schemes specifying which subjects are sampled at which time points.

To generate these schemes, fill out the scheme's dimensions and the maximum number of repetitions of individual schemes. You can already assess the possible number of schemes by clicking on **Check number of schemes before constraints** wich is much faster then generating the schemes first. Reconsider scheme dimensions when the number of schemes is too large. The possible number of schemes can also be cut down by imposing *scheme constraints*. Finally click on **Generate schemes** to receive all schemes meeting constraints. This might take a few minutes (see Figure 6).

3.5 Rank schemes

Schemes are ranked by their precision of estimating the area under the curve (AUC) and maximum concentration (Cmax) on simulated data.

Again we work in 2 steps:

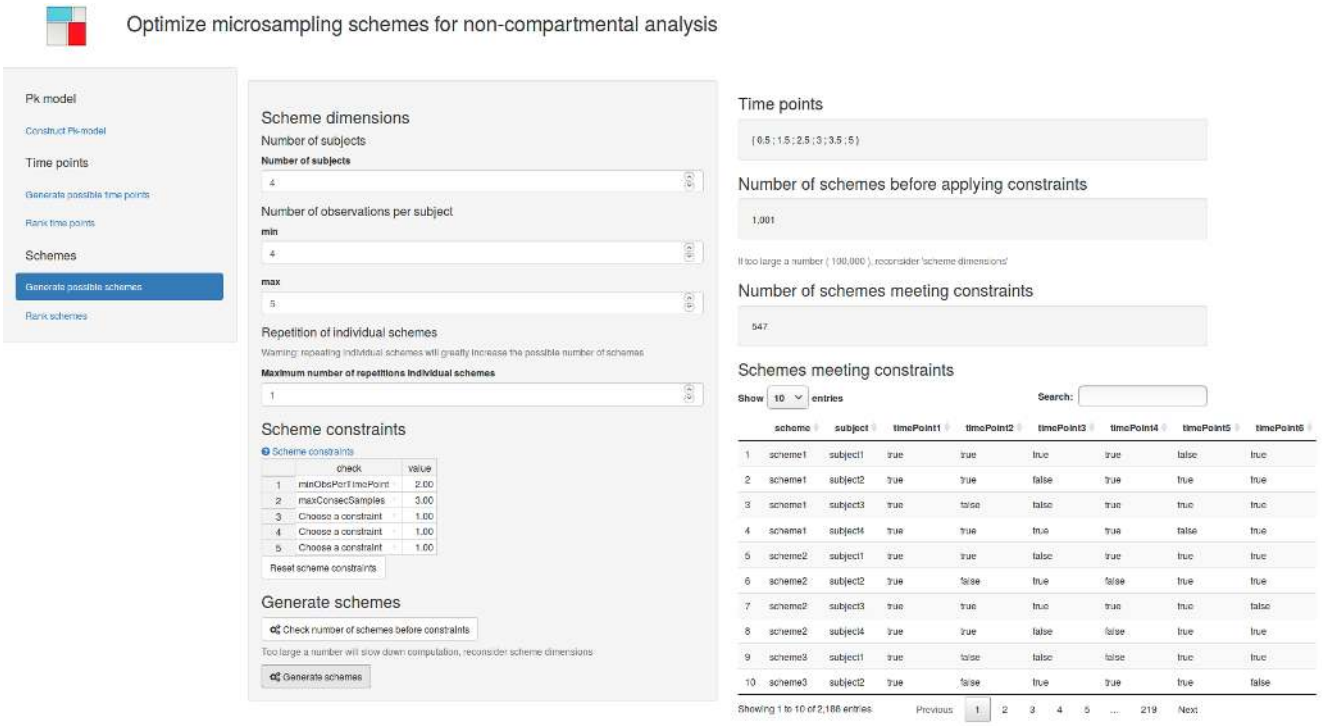


Figure 6: Generate schemes

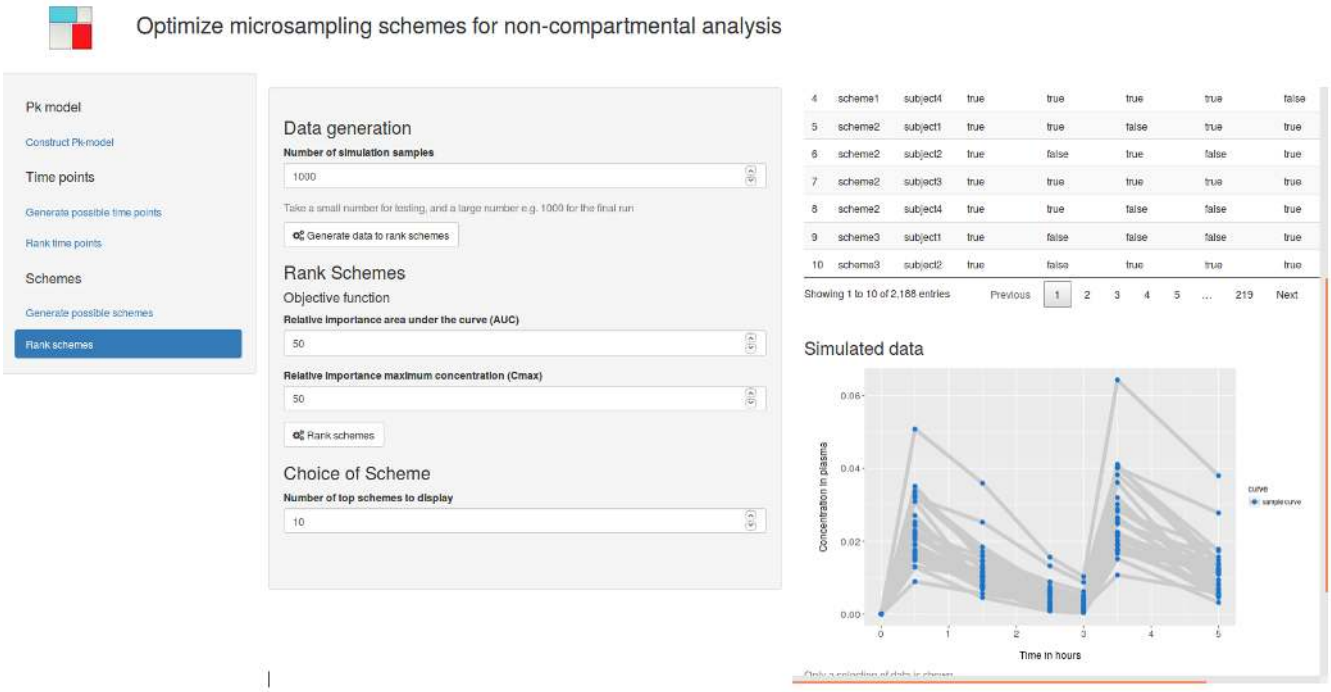


Figure 7: Generate data to rank schemes

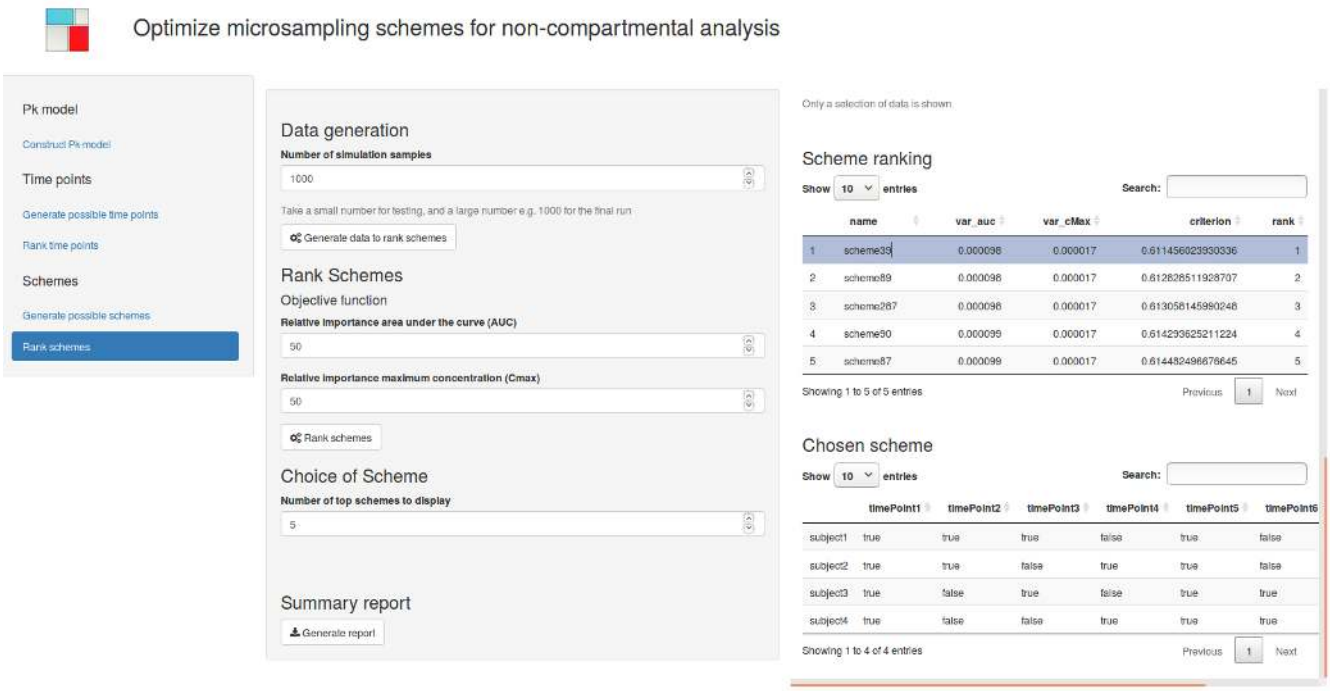


Figure 8: Rank schemes

3.5.1 Generate sample data

Generate data by specifying the number of simulation samples and click **Generate data to rank schemes** (see Figure 7).

3.5.2 Rank schemes

After data generation, specify the objective function by attaching a relative importance to different non-compartmental statistics and click on **Rank schemes** (see Figure 8). This might take some time.

Finally select a scheme by clicking on the **Scheme ranking** table.

When a final scheme is chosen, first click on **Generate report** and next on **Download report** to recieve a word document summarizing the main results.

4 Finding optimal designs using code

4.1 Settings

```
settings                <-  list()
settings$nSamples       <-  100 # increase for real life example
set.seed(124)
```

4.2 Construct a pharmacokinetic model

```
library( microsamplingDesign )
pkModel                <- getExamplePkModel()
```

some useful functions:

```
modelParameters      <-  getParameters( pkModel )
knitr::kable( modelParameters[ , c(1:2) ] )
```

parameter	value
F	1.00
volumePlasma	10.00
Cld	15.00
volumeTissue	15.00
VmaxAbsorption	5.00
kappaMMAbsorption	2.50
KaConstant	NA
VmaxClearance	30.00
kappaMMClearance	0.25
ClConstant	NA

To generate your own pharmacokinitic model see:

```
?construct2CompModel
```

4.3 Generate time points

Possible time points are generated from a full set of time points:

```
fullTimePointsEx     <-  seq( 0 , 16 , 0.5 )
print( fullTimePointsEx )
#>  [1]  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5
#> [15]  7.0  7.5  8.0  8.5  9.0  9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5
#> [29] 14.0 14.5 15.0 15.5 16.0
```

With the choice of options constraints by *timeZones*:

```
#timeZonesEx         <-  getExampleTimeZones()
timeZonesEx          <-  data.frame( startTime = c( 0 , 2 , 3 ) ,
  endTime = c( 2 , 3 , 16 ) ,
  nPointsPerZone = c( 2 , 1 , 2 )  )
knitr::kable( timeZonesEx )
```

startTime	endTime	nPointsPerZone
0	2	2
2	3	1
3	16	2

timeZones concept is defined such that : time zero is never included, last timePoint is always included.

Correct names should be used!

Now we can generate all time point options from a vector of possible time points under constraints defined in *timeZones*:

```
setOfTimePoints          <-  getAllTimeOptions( timeZones = timeZonesEx ,
    fullTimePoints = fullTimePointsEx )
# ?SetOfTimePoints   # class definition
#str( setOfTimePoints ) # to see all slots in the example
slotNames( setOfTimePoints  )
#> [1] ".Data"             "fullTimePoints"    "nFullTimePoints"
#> [4] "nTimePointsSelect" "nTimePointOptions" "ranking"

knitr::kable( head( getData( setOfTimePoints) ) )
```

	TimePoint1	TimePoint2	TimePoint3	TimePoint4	TimePoint5	TimePoint6
timePointOption1	0.5	1.0	2.0	3	3.5	16
timePointOption2	0.5	1.5	2.0	3	3.5	16
timePointOption3	1.0	1.5	2.0	3	3.5	16
timePointOption4	0.5	1.0	2.5	3	3.5	16
timePointOption5	0.5	1.5	2.5	3	3.5	16
timePointOption6	1.0	1.5	2.5	3	3.5	16

```
knitr::kable( tail( getData( setOfTimePoints) ) )
```

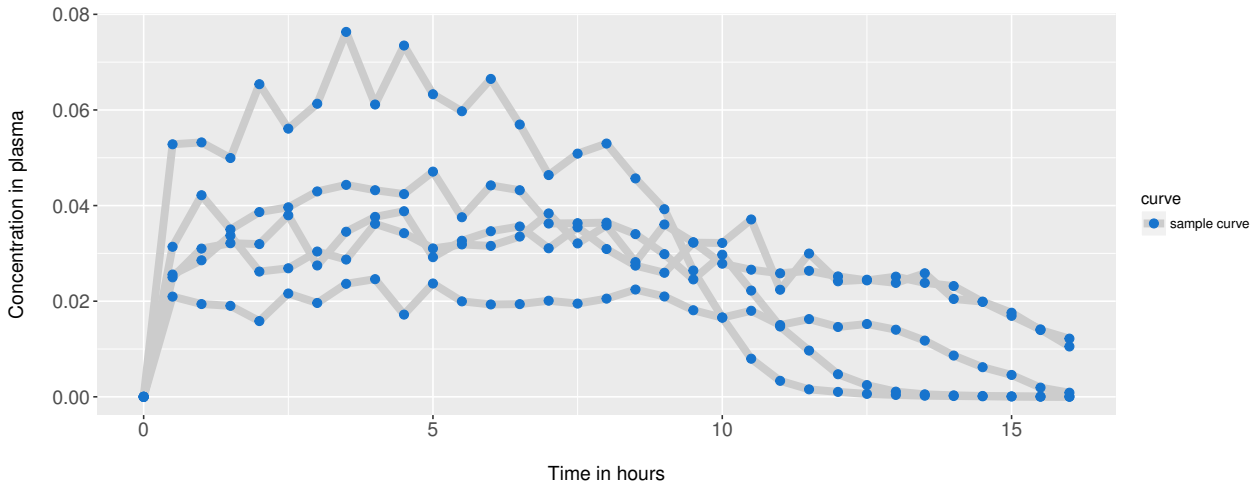
	TimePoint1	TimePoint2	TimePoint3	TimePoint4	TimePoint5	TimePoint6
timePointOption1945	0.5	1.0	2.0	15	15.5	16
timePointOption1946	0.5	1.5	2.0	15	15.5	16
timePointOption1947	1.0	1.5	2.0	15	15.5	16
timePointOption1948	0.5	1.0	2.5	15	15.5	16
timePointOption1949	0.5	1.5	2.5	15	15.5	16
timePointOption1950	1.0	1.5	2.5	15	15.5	16

note 0 never chosen , time 16 always included

4.4 Rank time points

To rank the timePoint options inside a *SetOfTimePoints* object , we first need to simulate *PkData*.

```
model               <-  getExamplePkModel()
fullTimePoints      <-  getTimePoints( setOfTimePoints )
pkDataForTimePoints <-  getPkData( pkModel = model , timePoints = fullTimePoints ,
  nSubjectsPerScheme = 5 , nSamples = settings$nSamples  )
plotObject( pkDataForTimePoints , nCurves = 5 )
```



This is just small number of samples, in reality one would use a larger number such as 1000.

We can than use the rank function to find the optimal time points:

```
rankedTimePoints      <-  rankObject( setOfTimePoints , pkData = pkDataForTimePoints ,
        nGrid =  150 , nSamplesAvCurve = settings$nSamples )
rankingTimePoints     <-  getRanking( rankedTimePoints )
knitr::kable( head( rankingTimePoints ) )
```

name	criterion	rank
timePointOption1307	0.0046569	1
timePointOption1306	0.0047028	2
timePointOption1391	0.0047126	3
timePointOption1313	0.0047186	4
timePointOption1301	0.0047273	5
timePointOption1217	0.0047366	6

```
#knitr::kable( tail( rankingTimePoints ) )
indTimeChoice         <-  getTopNRanking( rankingTimePoints , 1  )
bestTimeChoice        <-  setOfTimePoints[ indTimeChoice ,  ]
bestTimeChoice
#> TimePoint1 TimePoint2 TimePoint3 TimePoint4 TimePoint5 TimePoint6
#>        0.5        1.5        2.5        8.0       14.5       16.0
```

4.5 Generate possible schemes

```
timePointsChoice      <-  bestTimeChoice
```

To generate schemes we can define additional constraints:

```
constraintsExample    <-  getConstraintsExample()[c( 2 , 4 ) , ]
knitr::kable( constraintsExample )
```

	check	level	value
2	maxConsecSamples	subject	3
4	minObsPerTimePoint	scheme	2

Constraints are defined on 2 levels: *subject* or *scheme*.

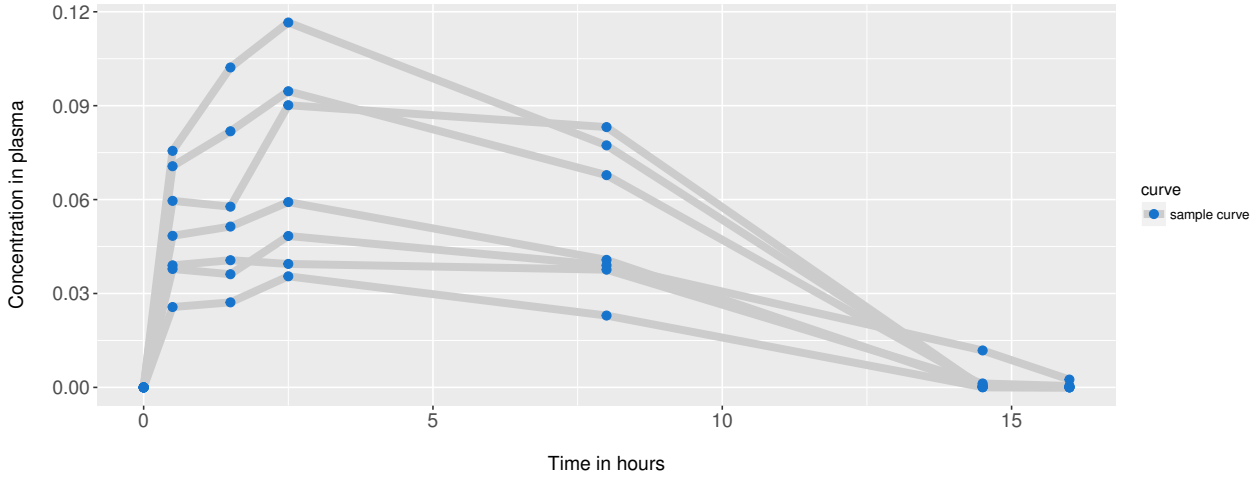
```
setOfSchemes          <-  getSetOfSchemes( minNSubjects = 4 , maxNSubjects = 5 ,
    minObsPerSubject = 4 , maxObsPerSubject = 5  ,
    timePoints =  timePointsChoice , constraints =  constraintsExample ,
    maxRepetitionIndSchemes = 1 , maxNumberOfSchemesBeforeChecks = 10^8  )
slotNames( setOfSchemes )
#> [1] ".Data"             "timePoints"        "nSchemes"
#> [4] "nSubjects"         "designConstraints" "ranking"
```

The number of combinations can get get very large especially with maxRepetitionIndSchemes > 1.

4.6 Rank schemes

To rank schemes, we need matching Pkdata (number of animals and timePoints):

```
timePointsEx    <-  getTimePoints( setOfSchemes )
pkData          <-  getPkData( pkModel, timePoints = timePointsEx ,
  nSubjectsPerScheme = 5 , nSamples = settings$nSamples )
plotObject( pkData , nCurves = 7 , addZeroIsZero = TRUE )
```



To rank schemes, we have to define an objective function, based on the a scheme based statistic (AUC , ...) a weight representing its relative importance.

```
exampleObjective      <-   data.frame(
  criterion = c( "auc" , "cMax" , "tMax" )  ,
  weight = c( 9 , 1, 1 ) )
knitr::kable( exampleObjective )
```

criterion	weight
auc	9
cMax	1
tMax	1

But be carefull cMax and tMax might be very variable when multiple doses are administered.

```
setOfSchemesRanked    <-  rankObject(setOfSchemes , pkData = pkData ,
  objective = exampleObjective , varianceMeasure = "var" , scaleWith  = "max" )
```

```
#> start Ranking Schemes on cluster with  1 cores
schemeRanking         <-  getRanking( setOfSchemesRanked )
knitr::kable( head(  schemeRanking ) )
```

name	var_auc	var_cMax	var_tMax	criterion	rank
scheme1925	0.0069886	0.0001045	2.846364	0.4955628	1
scheme1742	0.0069951	0.0001045	2.846364	0.4959535	2
scheme1047	0.0067687	0.0000959	5.400884	0.5066209	3
scheme1167	0.0068139	0.0001060	4.785758	0.5068546	4
scheme1484	0.0067777	0.0000959	5.400884	0.5071611	5
scheme1746	0.0071562	0.0001291	2.135454	0.5081408	6

```
knitr::kable( tail( schemeRanking ) )
```

	name	var_auc	var_cMax	var_tMax	criterion	rank
2272	scheme162	0.0129867	0.0001642	6.673611	0.9235083	2272
2273	scheme453	0.0126542	0.0002178	6.558081	0.9245740	2273
2274	scheme31	0.0131007	0.0001692	7.325732	0.9395866	2274
2275	scheme163	0.0131782	0.0001692	7.325732	0.9442519	2275
2276	scheme55	0.0133907	0.0001642	6.673611	0.9478366	2276
2277	scheme56	0.0135868	0.0001692	7.325732	0.9688575	2277

```
indTopSchemes         <-  getTopNRanking( schemeRanking , nSelect = 1 )
indBottomSchemes      <-  getTopNRanking( schemeRanking , nSelect = 1 , top = FALSE )
bestScheme            <-  setOfSchemesRanked[ , , indTopSchemes ]
knitr::kable( bestScheme )
```

	timePoint1	timePoint2	timePoint3	timePoint4	timePoint5	timePoint6
subject1	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
subject2	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE
subject3	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
subject4	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
subject5	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE

```
worstScheme           <-  setOfSchemesRanked[ , , indBottomSchemes ]
knitr::kable( worstScheme )
```

	timePoint1	timePoint2	timePoint3	timePoint4	timePoint5	timePoint6
subject1	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE
subject2	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
subject3	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
subject4	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE
subject5	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

5 Advanced options

5.1 Parallelization

Parallelization by forking is supported on linux machines and can be used to seed up simulating pkData, generating or ranking timepoints or schemes. You need to specify the number of cores inside these functions (*nCores*):

```
setOfSchemesRanked            <-  rankObject(setOfSchemes , pkData = pkData ,
objective = exampleObjective ,      varianceMeasure = "var" , scaleWith  = "max" ,
 nCores = 2 )
```

5.2 Working with ranges

Using ranges of parameters is also supported, see

```
?rankObjectWithRange
```

for details.

6 Memo of main functions

6.1 Data generation

- *getExamplePkModel*: Get an example of a PkModel
- *construct2CompModel* Construct your own 2 compartmental model
- *getPkData* to generate data from your a PkModel
- *plotObject* visualize model or data

6.2 Generate and rank time points

- *getAllTimeOptions*
- *getPkData*
- *rankObject*

6.3 Generate and rank schemes

- *getSetOfSchemes*
- *getPkData*
- *rankObject*

References

Barnett, Helen, Helena Geys, Tom Jacobs, and Thomas Jaki. 2017. "Optimal Designs for Non-Compartmental Analysis of Pharmacokinetic Studies."

Chapman, Kathryn, Simon Chivers, Dan Gliddon, David Mitchell, Sally Robinson, Tim Sangster, Susan Sparrow, Neil Spooner, and Amanda Wilson. 2014. "Overcoming the Barriers to the Uptake of Nonclinical Microsampling in Regulatory Safety Studies." *Drug Discovery Today* 19 (5). Elsevier: 528–32.

Gabrielsson, Johan, and Daniel Weiner. 2001. *Pharmacokinetic and Pharmacodynamic Data Analysis: Concepts and Applications*. Vol. 1. CRC Press.

Halliwell, Leigh J. 2015. "The Lognormal Random Multivariate." In *Casualty Actuarial Society E-Forum, Spring 2015*.