



A MINI-TUTORIAL ON CAMPSIS: THE R-BASED PK/PD SIMULATION SUITE

Andreas Lindauer, Nicolas Luyckx, Erno van Schaick, Emilie Hénin, Stéphanie Blaizot, Sajid Ali Shah, Pierre-

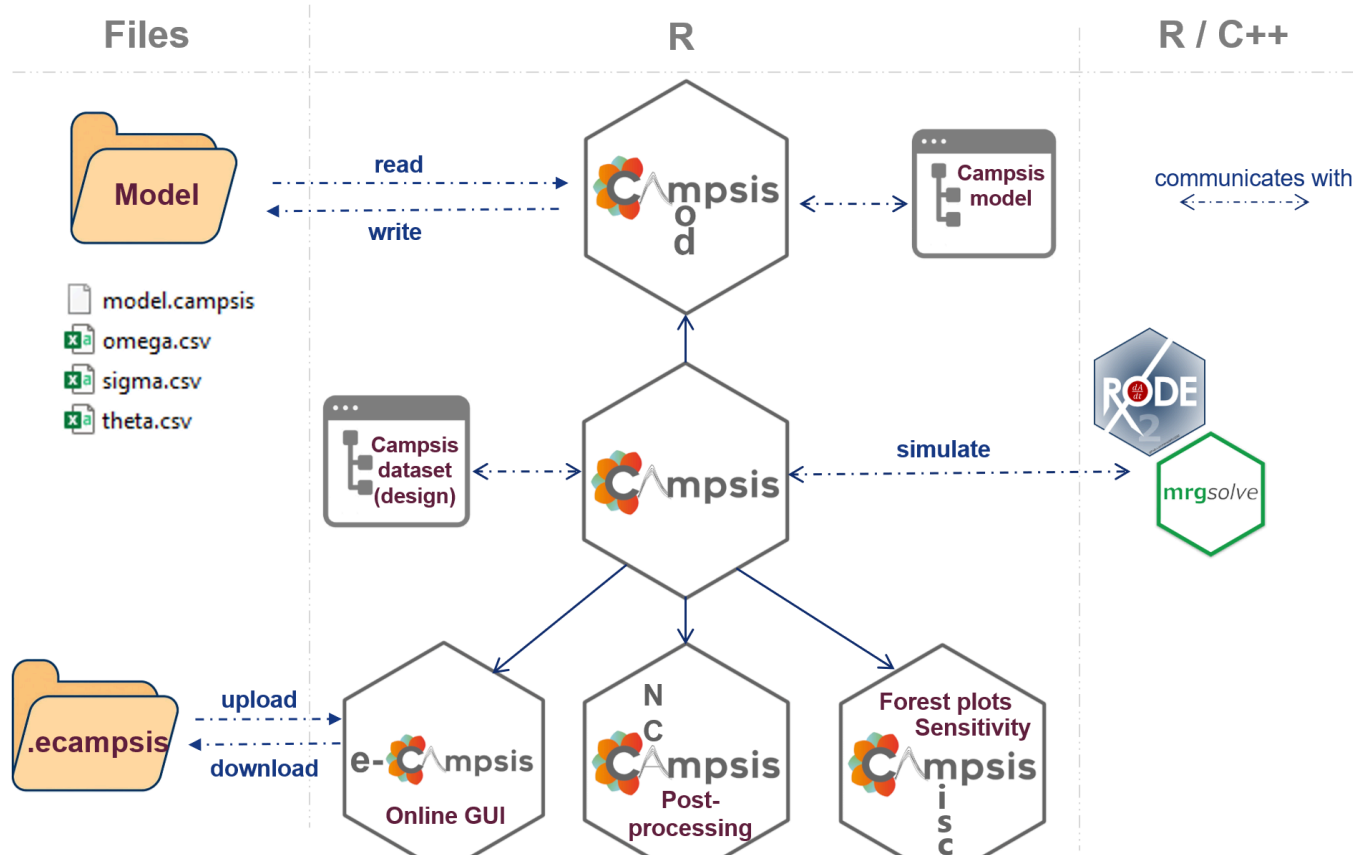
Alexandre Noguine, Christian Laveille

Calvagone SAS, France



INTRODUCTION

Campsis (www.campsis.org) is a suite of R packages designed to facilitate the setup and execution of PK/PD simulations in a user-friendly way. The two core packages, `campsismod` [1] and `campsis` [2], are open-source and freely available on CRAN and GitHub since 2022. `Campsismod` is dedicated to the manipulation of the pharmacometric model, while `campsis` executes the simulation using `rxode2` or `mrqsolve`. `e-Campsis` [3] is an online application that assists users in setting up and running simulations, visualizing and post-processing results, and exploring scenarios in an intuitive and interactive way. The complete simulation code can be downloaded and reproduced or further adapted offline in R. This mini-tutorial provides an overview of the `campsis` building blocks and their functionality.



The Campsis family

campsismod

Purpose:

- Model definition
 - Library of PK & PD models
 - Model manipulation (scripted)
 - Edit parameters
 - Read/write model files
 - Model export
- Define the simulation dataset:
 - Arms
 - Dosing schedule
 - Observations
 - Covariates and covariate distributions
 - Set the framework:
 - Inter-occasion variability
 - Time-varying covariates
 - Random number generation for simulation of categorical endpoints
 - Bootstrapping of covariates or post-hoc individual parameters
 - Scenarios (e.g. explore different parameter values)
 - Interruption events to handle response-triggered dose adaptation
 - Execute the simulation:
 - Uses `rxode2` and `mrqsolve` simulation engines in the background
 - Parallel execution of big simulations

Load model from library

Load 2-compartment PK model from built-in model library and show content.

```
model <- model_suite$pk$`2cpt_fo`
```

```
[MAIN]
TVBIO=THETA_BIO
TVKA=THETA_KA
TVVC=THETA_VC
TVVP=THETA_VP
TVQ=THETA_Q
TVCL=THETA_CL

BIO=TVBIO
KA=TVKA * exp(ETA_KA)
VC=TVVC * exp(ETA_VC)
VP=TVVP * exp(ETA_VP)
Q=TVQ * exp(ETA_Q)
CL=TVCL * exp(ETA_CL)

[ODE]
d/dt(A_ABS)=-KA*A_ABS
d/dt(A_CENTRAL)=KA*A_ABS + Q/VP*A_PERIPHERAL
               - Q/VC*A_CENTRAL - CL/VC*A_CENTRAL
d/dt(A_PERIPHERAL)=Q/VC*A_CENTRAL - Q/VP*A_PERIPHERAL

[F]
A_ABS=BIO

[ERROR]
CONC=A_CENTRAL/VC
if (CONC <= 0.001) CONC=0.001
CONC_ERR=CONC*(1 + EPS_PROP_RUV)
```

Adapt model code

Using `add()`, one can easily add an equation to the model. Here for example we want to derive also the elimination rate constant `k10` from the clearance `CL` and the volume of distribution `VC`.

```
model <- model %>% add(Equation("k10", "CL/VC"))
```

The first argument of the `Equation()` function is the left-hand side of the equation, the second argument is the right-hand side. By default the equation is added at the end of the `[MAIN]` part of the model code, just before the `[ODE]` system. However, one can easily specify the position of the equation by using the `position` argument of the `add()` function:

```
model <- model %>% add(Equation("CONC_ngmL",
                                "CONC*1000",
                                pos=Position(ErrorRecord())))
```

With `add()` you can not only add equations but also parameters, ODE state variables, lag-times, and more:

```
model <- model %>% add(LagTime(compartment=1, "LAG"))
```

`replace()` allows you to substitute a part of the model code with another part. For example, you can replace the equation for `CL` by one that includes a body weight effect:

```
model <- model %>% replace(Equation("CL",
                                    "TVCL * pow((BW/70),0.75) * exp(ETA_CL)"))
```

Note that e-Campsis use the C++ syntax for expressing a power function.

Parameter values can be replaced like this:

```
model <- model %>% replace(Theta("KA", value=2))
model <- model %>% replace(Omega("CL", value=20,
                                  type="cv%"))
```

You can remove equations or other parts of the model code with `delete()`:

```
model <- model %>% delete(Equation("k10"))
```

Combine two models

A very convenient feature of `campsismod` is the ability to combine two models, for example adding a PD model to a PK model, or - as shown below - to combine to PK models to simulate a head-to-head comparison of two drugs or formulations.

First load the models into two different objects:

```
# Model of test formulation
model_tst <- read.campsis(file="test_model/Model")

# Model of reference formulation
model_ref <- read.campsis(file="ref_model/Model")
```

Combine the models, adding a suffix to the parameter names of the reference model to avoid ambiguity:

```
model_comb <- model_tst %>%
  add(addSuffix(model_ref, suffix = "ref"))
```

campsis

Purpose:

- Define the simulation dataset:
 - Arms
 - Dosing schedule
 - Observations
 - Covariates and covariate distributions
- Set the framework:
 - Inter-occasion variability
 - Time-varying covariates
 - Random number generation for simulation of categorical endpoints
 - Bootstrapping of covariates or post-hoc individual parameters
 - Scenarios (e.g. explore different parameter values)
 - Interruption events to handle response-triggered dose adaptation
- Execute the simulation:
 - Uses `rxode2` and `mrqsolve` simulation engines in the background
 - Parallel execution of big simulations

Define the simulation dataset

The following lines illustrate how to define a very simple single-arm study with 25 subjects, a bolus dose of 1000 mg given into compartment 1 at time 0, followed by 6 additional doses every 24 hours. Observations are made every 4 hours for 7 days.

```
ds <- Dataset(subjects=25) %>%
  add(Bolus(time=0, amount=1000,
            compartment =1, ii=24, addl=6)) %>%
  add(Observations(times=seq(0, days(7), by=4)))
```

Note here the convenient use of the `days()` function to specify the duration of the observation period. Other shortcut functions to convert time units are: `seconds()`, `minutes()`, `weeks()`, `months()`, and `years()`. These functions will return the time in hours, assuming that this is the base time unit of your model. Should you have a different base time unit, you can specify this with the `timeUnitExport` argument of the `DatasetConfig()` function:

```
add(DatasetConfig(timeUnitExport="day"))
```

If you want to simulate a multi-arm study, you can define the arms separately and then combine them:

```
arm1 <- Arm(subjects=3, label="1000 mg/h") %>%
  add(Infusion(time=0, 1000,duration=1)) %>%
  add(Observations(times=seq(0, 24, by=0.1)))

arm2 <- Arm(subjects=5, label="1500 mg/h") %>%
  add(Infusion(time=0, 1000,duration=1)) %>%
  add(Observations(times=seq(0, 24, by=0.1)))

ds.arms <- Dataset() %>% add(c(arm1, arm2))
```

Covariates

Covariates can be defined as follows (adding to the dataset object `ds` created above):

a) Same value for all subjects:

```
ds <- ds %>%
  add(Covariate("BW", 70))
```

b) Different values for each subject:

You provide a vector with the exact value for each subject. The length of the vector must match the number of subjects in the dataset.

```
ds <- ds %>%
  add(Covariate("BW", c(70, 80, ..., 90)))
```

c) Randomly sampled from a distribution:

You can specify random distributions with the following shortcut functions:

```
ds <- ds %>%
  add(Covariate("BW",
                UniformDistribution(min=50, max=90))) %>%
  add(Covariate("AGE",
                NormalDistribution(mean=50, sd=10))) %>%
  add(Covariate("HGT",
                LogNormalDistribution(meanlog=log(170),
                                      sdlog=0.2))) %>%
  add(Covariate("SMOKE",
                BinomialDistribution(trials=1,
                                     prob=0.5))) %>%
  add(Covariate("CYP2D6",
                DiscreteDistribution(c(1,2,3),
                                     prob=c(0.5, 0.3, 0.2)))
```

In fact you can use any of the available R functions like `rpois`, `rexp`, `rbeta` etc. to define your own distribution. For example, to define a Poisson distribution with a rate of 0.1:

```
ds <- ds %>%
  add(Covariate("SEIZURE",
                FunctionDistribution('rpois',
                                    list(lambda=0.1))))
```

d) Sample from an external file:

It is also possible the sample covariates with replacement (i.e. bootstrap) from an external file. The NHANES dataset (including SEX, AGE, BW, BMI, HT) is already included in the package and can be used as follows, for example to sample only boys under 18 years old:

```
ds <- ds %>%
  add(Bootstrap(data=nhanes %>% filter(AGE<18,SEX==1),
                replacement=TRUE, random=TRUE, export_id=TRUE))
```

Time-varying covariates

Step-changes of a covariate at specific time points can be defined by providing a dataframe with the columns `TIME` and `VALUE`. For example, to simulate a weekly decrease in body weight from 80 to 65 kg over the course of 3 weeks you can do the following:

```
ds <- ds %>%
  add(TimeVaryingCovariate("BW",
                           data.frame(TIME=days(c(0,7,14,21)),
                                       VALUE=c(80,75,70,65))))
```

However, keep in mind that step-changes in covariates are they way they appear in the analysis dataset given a study design, in reality however, there is likely an underlying mechanism that changes a covariate in a continuous way. You may rather define a function that describes the time course of the covariate in your model:

```
BWt = BW0 - k * TIME
CL = THETA_CL * pow(BWt/70,0.75) * exp(ETA_CL)
```

Inter-occasion variability

Assuming that you model code already contains a random variable that represents IOV, e.g. like this:

```
KA = THETA_KA * exp(ETA_KA + IOV_KA)
```

You can define the IOV in the dataset as follows:

```
ds <- ds %>%
  add(IOV("IOV_KA", EtaDistribution(model, "IOVCL")))
```

This will sample the IOV from a normal distribution with mean 0 and standard deviation equal to the estimated value of the omega-IOVCL parameter in the model.

Dose adaptation

A priori dose adaptation based on a covariate can be very easily defined as follows:

```
dataset <- Dataset(25) %>%
  add(Bolus(time=0, amount=0.5)) %>% # 0.5mg / kg
  add(Covariate("BW",
                NormalDistribution(mean=70, sd=10))) %>%
  add(DoseAdaptation("AMT*BW"))
```

More advanced dose-adaptation based on e.g. a target concentration or a PD response can be defined using the so-called "interruption events". For this we use the `Event()` function to define the time points at which we would like to check if a dose adaptation is necessary, as well as providing the actual function that makes the evaluation:

```
events <- Events() %>%
  add(Event(name="Dose adaptation", times=seq(0, 144,
                                             by=24), fun=function(inits) {
    inits$BIO <- ifelse(inits$CONC > 5, inits$BIO*0.75,
                       inits$BIO)
    return(inits)
  })))
```

To adapt the dose in a PK model it is easiest to actually change the bioavailability. The example above checks every 24 hours if the concentration exceeds 5 mg/L and if so, reduces the bioavailability by 25%. You need to define in the dataset which variable is the one to be changed and in `simulate()` the event object needs to be provided

```
ds <- ds %>%
  add(EventCovariate("BIO", 1))

results <- model %>%
  simulate(dataset, events=events, seed=1,
          outvars="BIO")
```

Of course, make sure that your model is set up to use the `BIO` variable as the bioavailability.

Scenarios

One of the most powerful features of the `campsis` package is the ability to define scenarios. Scenarios are a set of conditions that can be applied to the dataset or the model to simulate different situations. For example, you can define a scenario with a different dosing schedule, different covariate values, and different parameter values. Note that while subjects are different in each arm (according to the IIV and covariate distribution), scenarios re-use the same subjects but with different conditions. The following example shows how to define 3 scenarios with different values of the `KA` parameter and two scenarios with different distributions of the `BW` covariate:

```
scenarios <- Scenarios() %>%
  add(Scenario("THETA_KA=1", model=~.x %>%
               replace(Theta(name="KA", value=1)))) %>%
  add(Scenario("THETA_KA=3", model=~.x %>%
               replace(Theta(name="KA", value=3)))) %>%
  add(Scenario("THETA_KA=6", model=~.x %>%
               replace(Theta(name="KA", value=6)))) %>%
```

```
add(Scenario("BW ~ Uniform dist.", dataset=~.x %>%
             replace(Covariate("BW",
                               UniformDistribution(min=60, max=80)))) %>%
  add(Scenario("BW ~ Normal dist.", dataset=~.x %>%
             replace(Covariate("BW",
                               NormalDistribution(mean=70, sd=10))))
```

Executing a simulation

Once you have loaded the model and defined the dataset and maybe some scenarios, you can simulate the model using the `simulate()` function.

```
results <- model %>% simulate(dataset,
                             scenarios=scenarios, seed=1)
```

Additional arguments and shortcut functions are available to e.g. disable the variability components, run multiple replicates of a study.

campsisnca

`campsisnca` is a package that provides a set of functions to perform non-compartmental analysis (NCA) on the simulated data. As shown below, the input to the `NCAMetrics()` function is the simulated data, filtered by a particular time-range within which the metric of interest (AUC, C_{max}, etc.) is calculated. Custom metrics can be defined to derive e.g. the percentage of patients with a C_{max} above a certain threshold.

```
custom <- CustomMetric(fun=~Cmax > 13,
                       name="Cmax > 13",
                       unit="%", categorical=TRUE)
```

```
# Day 1
ncaD1 <- NCAMetrics(x=results %>% timerange(0, 24),
                   variable="CONC", scenario=c(day="Day 1")) %>%
  add(c(AUC(unit="ng/mL*h"), Cmax(unit="ng/mL"),
        Tmax(unit="h"), Ctrough(unit="ng/mL"), custom)) %>%
  calculate()
```

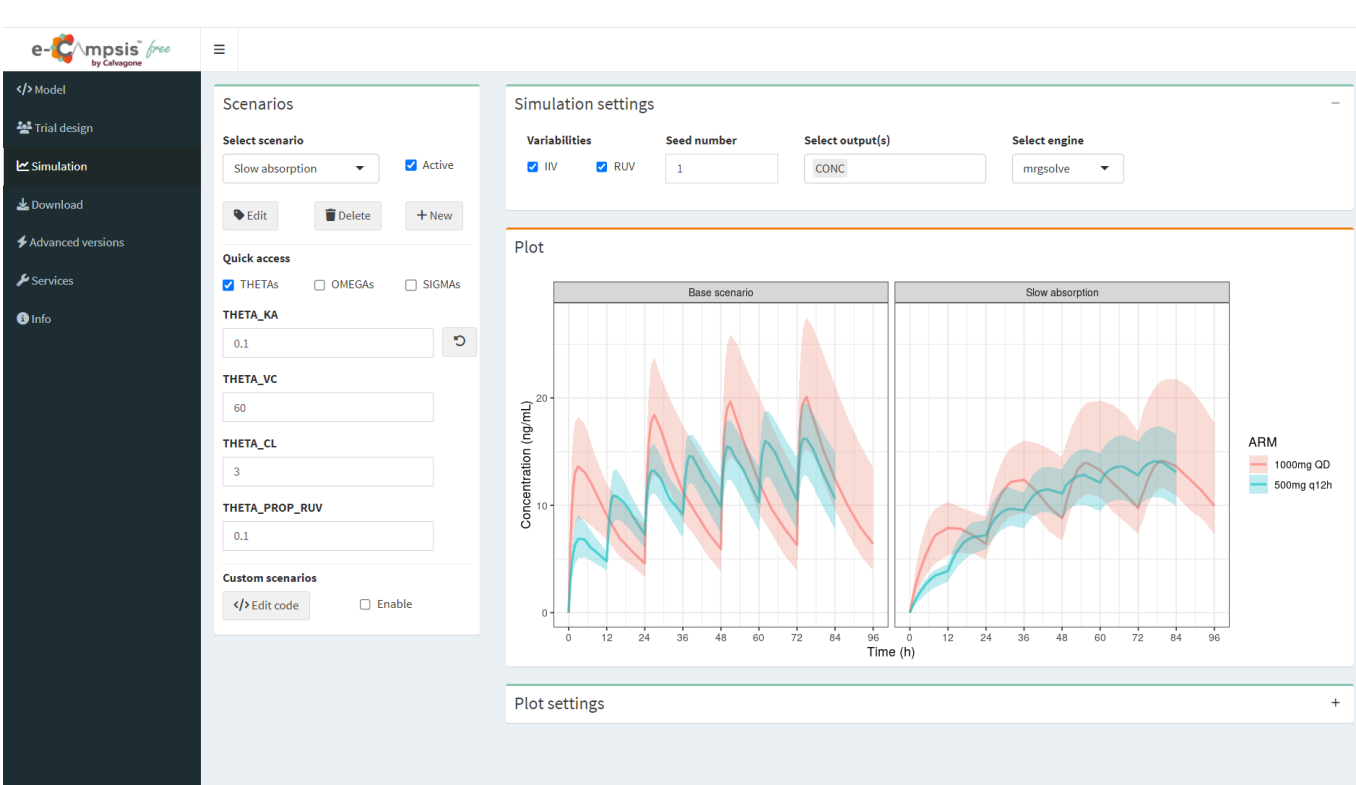
```
# Day 7
ncaD7 <- NCAMetrics(x=results %>% timerange(144, 168,
                                           rebase=TRUE), variable="CONC",
                   scenario=c(day="Day 7")) %>%
  add(c(AUC(), Cmax(), Tmax(), Ctrough(), custom)) %>%
  calculate()
```

```
table <- NCAMetricsTable() %>%
  add(c(ncaD1, ncaD7))
table %>% export(dest="gt", subscripts=TRUE) %>%
  as_raw_html()
```

Metric	Day 1, N = 200 ¹	Day 7, N = 200 ¹
AUC (ng/mL*h)	134 (104–168)	198 (130–304)
C _{max} (ng/mL)	10.1 (7.8–14.0)	14.7 (10.8–21.0)
t _{max} (h)	2.00 (1.00–6.00)	2.00 (1.00–6.00)
C _{trough} (ng/mL)	2.82 (1.51–4.39)	4.39 (1.96–7.86)
¹ Median (5%–95%)		

e-Campsis

`e-Campsis` is a graphical user interface that facilitates the configuration of models, setting up of a study design, exploration of scenarios, and visualization of results while running `campsis` and `campsismod` functions in the background. It allows the user to run simulations interactively in a browser window but also to download the full simulation code to reproduce the results offline in R.



Screenshot of e-Campsis interface

Go to www.e-campsis.com to try out the free version!

Additional convenience features, like automatic translation of NONMEM models to `campsis`, generation of sensitivity analysis and forest plots, are available as proprietary functions in a professional version of `e-Campsis`.

CONCLUSION

The `campsis`-suite aims to be a robust tool for both hands-on pharmacometricians and non-coding experts. Its versatility allows it to manage complex simulation settings, such as response-triggered dose adaptation. For simpler, yet often more impactful simulations, `e-Campsis` is set to revolutionize the way simulations are used in communication. It moves away from the traditional approach of using pre-prepared static slides, and instead, promotes an interactive and highly educational method of conveying information. This shift not only enhances understanding but also ensures the message is effectively communicated.

REFERENCES

- <https://cran.r-project.org/web/packages/campsismod/index.html>
- <https://cran.r-project.org/web/packages/campsis/index.html>
- <https://www.e-campsis.com>
- <https://github.com/nlmixr2/rxode2>
- <https://github.com/metrumresearchgroup/mrqsolve>