simulx() Function User Guide

Contents

Introduction

Simulx function overview **5**
Description . . . 5
Usage . . . 5
Arguments . . . 5
Details . . . 7
Value . . . 7

Installation **8**
Requirements . . . 8
Install mlxR . . . 8
Using mlxR when Monolix 2020 is installed . . . 8
Download the demo examples . . . 9

Calculation of functions of time

Function of time with analytical expression **10**
Introduction . . . 10
Example . . . 10

Ordinary differential equations **16**
Introduction . . . 16
Example . . . 16

Delay differential equations **19**
Introduction . . . 19
Example . . . 19

Model with regression variables **21**
Introduction . . . 21
Examples . . . 21

Simulation of dynamical systems – PK models

ODE based model with source terms **30**
Introduction . . . 30
Examples . . . 30

Simulation of PK models: single route of administration **40**
Introduction . . . 40
Example . . . 40

Simulation of PK models: multiple routes of administration **50**

Introduction . . . 50
Examples . . . 50

Simulation of PK models: weight-based dosing **56**
Introduction . . . 56
Examples . . . 56

Simulation of nonadherence **61**
Introduction . . . 61
Using random individual dosage regimens . . . 61
Defining nonadherence as a regression variable . . . 64
Defining nonadherence as a sequence of individual parameters . . . 67

Filling, emptying and resetting compartments **71**
Introduction . . . 71
Filling compartments . . . 71
Emptying compartments . . . 73
Resetting compartments . . . 77

Simulation of longitudinal data

Simulation of continuous data **83**
Introduction . . . 83
Example . . . 83

Simulation of categorical data **85**
Introduction . . . 85
Example using the probabilities . . . 85
Example using the cumulative logits . . . 86

Simulation of categorical data with Markovian dependence **89**
Introduction . . . 89
Examples . . . 90

Simulation of count data **94**
Introduction . . . 94
Examples . . . 94

Simulation of time-to-event data **98**
Introduction . . . 98
Simulation of a single event . . . 99
Simulation of repeated events . . . 101
Defining individual designs . . . 102

Simulation of a joint model **107**
Introduction . . . 107
Examples . . . 107

Censored data (data below/above a limit of quantification) **114**
Introduction . . . 114
Example . . . 114

Simulation of hierarchical models

Simulation of hierarchical model: Introduction **119**

Simulation of a hierarchical model: the individual parameters **120**
Introduction . . . 120
Examples . . . 120

Simulation of a hierarchical model: the individual covariates **127**
Introduction . . . 127
Example . . . 127

Model with categorical random covariates **133**
Introduction . . . 133
Example 1 . . . 133
Example 2 . . . 135

Simulation of a hierarchical model: the population parameters **138**
Introduction . . . 138
Example . . . 138

Extensions of the statistical model

Library of probability distributions **142**
Normal and transformations of normal distributions . . . 142
Continuous probability distributions . . . 143
Discrete probability distributions . . . 145

Correlation between variables **146**
Introduction . . . 146
Examples . . . 146

Inter occasion variability (mlxR >= 3.2.2) **150**
Introduction . . . 150
A basic example . . . 150
Combining IOV and time varying covariates . . . 154

Simulation of several individuals and replicates

Defining groups - part I **159**
Introduction . . . 159
Example . . . 159

Defining groups - part II **164**
Introduction . . . 164
Examples . . . 165

Simulating several replicates of the same experiment **171**

Computing summary statistics **174**

Reading data files and using data frames **178**
Introduction . . . 178
Individual parameters defined in a data frame . . . 178
Treatment defined in a data frame . . . 180
Observation times defined in a data frame . . . 182
Individual parameters, dose regimens and observation times defined in data frames . . . 183
Sampling/resampling individuals from a database . . . 185

Using Simulx with a Monolix project

Using and modifying the original design **188**

Resampling the patients of the original study **195**

Introducing uncertainty on the population parameters **200**

Miscellaneuous

Using a model defined inline **205**
Introduction . . . 205
Example . . . 205

Creation of new outputs **208**
Example 1 . . . 208
Example 2 . . . 210

Writing the results to a file or a folder **212**
Writing simulated data . . . 212
Using a Monolix project . . . 215

Reproducing the same random results **216**
Introduction . . . 216
Example . . . 216

Optimizing the computation time **218**
Introduction . . . 218
Example . . . 218

Simulating random variables in a block EQUATION: **221**
Introduction . . . 221
Using random variables in Section [LONGITUDINAL] . . . 221
Using random variables in Section [INDIVIDUAL] . . . 223

Using a R model instead of Mlxtran **225**
A basic PK model . . . 225
An ODE based model . . . 226
Comparing computational times between R and Mlxtran . . . 228
A model with regression variables . . . 230

Simulx function overview

simulx{mlxR}

Description

Compute predictions and sample data from `Mlxtran` models.

Usage

```
simulx(model = NULL, parameter = NULL, output = NULL, treatment = NULL,
       regressor = NULL, varlevel = NULL, group = NULL, data = NULL,
       project = NULL, nrep = 1, npop = NULL, fim = NULL,result.folder = NULL,
       result.file = NULL, stat.f = "statmlx", addlines=NULL, settings = NULL)
```

Arguments

model

a Mlxtran model used for the simulation.

```
simulx(model='myPKmodel.txt', ... )
```

parameter

a vector of parameters with their names and values

```
p <- c(a=1, b=-0.5, c=3)
simulx(...,parameter=p, ... )
```

output

a list, or a list of lists, with fields

- name : a vector of output names,
- time : a vector of times (only for the longitudinal outputs).
- lloq : lower limit of quantification (only for the longitudinal outputs).
- uloq : upper limit of quantification (only for the longitudinal outputs).
- limit : lower bound of the censoring interval (only for the longitudinal outputs).

```
o1 <- list(name=c('V','Cl'))
o2 <- list(name='C',time=seq(0,24,by=0.1))
o3 <- list(name='y', time=seq(0,24,by=2), lloq=0.1, limit=0)
simulx(..., output=list(o1, o2, o3), ... )
```

treatment

a list with fields

- time : a vector of input times,
- amount : a scalar or a vector of amounts,
- rate : a scalar or a vector of infusion rates (default=∞),
- tinf : a scalar or a vector of infusion times (default=0),
- type : the type of input (default=1),
- target : the target compartment (default=NULL).

```
tr <- list(time=c(0,12), amount=50, tinf=1)
simulx(...,treatment=tr, ... )
```

regressor

a list, or a list of lists, with fields

- name : a vector of regressor names,
- time : a vector of times,
- value : a vector of values.

```
reg <- list(name='x', time=c(0,2,5), value=c(5,10,7))
simulx(..., regressor=reg, ... )
```

varlevel

a list, or a dataframe, with fields

- name : name of the variable which defines the occasions,
- time : a vector of times (beginnings of occasions),

group

a list, or a list of lists, with fields

- size : size of the group (default=1),
- level : level of variability,
- parameter : if different parameters per group are defined,
- output : if different outputs per group are defined,
- treatment : if different treatements per group are defined,
- regressor : if different regression variables per group are defined.

```
g <- list(size=100, level='individual'))
simulx(...,group=g, ... )
```

data

a list (output of simulx when settings$data.in==TRUE)

project

the name of a Monolix project

```
simulx(..., project='monolix/pk_project.mlxtran', ... )
```

nrep

number of replicates

npop

number of population parameters to draw randomly

fim

a string with the Fisher Information Matrix to be used

result.folder

the name of the folder where the outputs of simulx should be stored

result.file

the name of the single file where the outputs of simulx should be saved

stat.f

a R function for computing some summary (mean, quantiles, survival,...) of the simulated data. Default = "statmlx".

addlines

a list with fields

- section : a string (default = "[LONGITUDINAL]"),
- block : a string (default = "EQUATION:"),
- formula : string, or vector of strings, to be inserted.

settings

a list of optional settings

- seed : initialization of the random number generator (integer),
- load.design : TRUE/FALSE (if load.design is not defined, a test is automatically performed to check if a new design has been defined),
- data.in : add columns id (when N=1) and group (when #group=1), TRUE/FALSE (default=FALSE)
- id.out : TRUE/FALSE (default=FALSE)
- kw.max : maximum number of trials for generating a positive definite covariance matrix (default = 100)
- sep : the field separator character (default = ",")
- digits : number of decimal digits in output files (default = 5)
- disp.iter : display replicate and population numbers, TRUE/FALSE (default=FALSE)
- replacement : sample id's with/without replacement, TRUE/FALSE (default=FALSE)
- out.trt : output of simulx includes treatment, TRUE/FALSE (default=TRUE)
- format.original : with a Monolix project, write data in result.file using the original format of the original data file TRUE/FALSE (default=FALSE)

```
simulx(..., settings=list(seed=s, digits=3), ... )
```

Details

simulx takes advantage of the modularity of hierarchical models for simulating different components of a model: models for population parameters, individual covariates, individual parameters and longitudinal data.

Furthermore, simulx allows to draw different types of longitudinal data, including continuous, count, categorical, and time-to-event data.

The models are encoded using the model coding language `Mlxtran`. These models are automatically converted into C++ codes, compiled on the fly and linked to `R` using the `Rcpp` package. That allows one to implement very easily complex models and to take advantage of the numerical sovers used by the C++ `MlxLibrary`.

Value

A list of data frames. Each data frame is an output of simulx.

Installation

Requirements

Using `mlxR` requires to install first the *Monolix Suite* from here

`mlxR 4.2` is compatible with the Monolix Suites *2019R1* and *2019R2* (both recommended), *2018R2* and *2018R1*.

When `mlxR 4.2` is used with the Monolix Suite *2019R1*or *2019R2*, it is necessary to install the R package `lixoftConnectors`. By default, the command line for installing `lixoftConnectors` is:

```
# for Windows OS
install.packages("C:/ProgramData/Lixoft/MonolixSuite2019R2/
                  connectors/lixoftConnectors.tar.gz",
                  repos = NULL, type="source")

# for MAC OS
install.packages("/Applications/MonolixSuite2019R2.app/Contents/Resources/
                  monolixSuite/connectors/lixoftConnectors.tar.gz",
                  repos = NULL, type="source")
```

Note that the `lixoftConnectors` package requires the `RJSONIO` package which can be installed from CRAN:

```
install.packages("RJSONIO")
```

Look at the installation procedure for more details.

Install mlxR

You can either

- install `mlxR 4.2` from **CRAN**:

```
install.packages("mlxR")
```

- install the development version of `mlxR` from **GitHub**:

```
devtools::install_github("MarcLavielle/mlxR")
```

See the release notes for more details on the current versions of `mlxR` on CRAN and GitHub.

Using mlxR when Monolix 2020 is installed

Monolix has evolved a lot in the 2020 version, and the Lixoft connectors also. As a consequence, `mlxR` is not compatible with Monolix 2020R1, but a new R package is available on the CRAN which is compatible with Monolix 2020R1: `RsSimulx`

You need Monolix <= 2019R2 if you want to use `mlxR`. It's not a problem since you can have both the 2019 and the 2020 versions installed on the same machine.

Assume that you have installed Monolix 2020R1 and Monolix2019R2 on your machine. Assume that you have installed `lixoftConnectors (2020)` package in the default library. Then, the solution is to keep `mlxR` linking to the 2019 version of Monolix and use the 2019 connectors:

- install 'lixoftConnectors 2019R2 package in a specific directory, the Monolix 2019R2 directory for instance :

```
monolix2019R2.path <-  "C:/ProgramData/Lixoft/MonolixSuite2019R2"
LC2019 <- file.path( monolix2019R2.path, "connectors/lixoftConnectors.tar.gz")
install.packages(LC2019, lib= monolix2019R2.path , repos = NULL, type = "source")
```

- when you want to use `mlxR`, load `lixoftConnectors` 2019, load `mlxR` and link it to Monolix2019R2:

```
library(lixoftConnectors, lib.loc = monolix2019R2.path )
library(mlxR)
initMlxR(path = monolix2019R2.path)   #(adapt the path if necessary).
```

Download the demo examples

Download the `R` scripts and `Mlxtran` codes used for the user guide and the case studies:

- mlxR41_demos.zip, October 14th, 2019

Function of time with analytical expression

R script: analytical.R

Mlxtran code: model/analytical.txt

Introduction

Functions of time are implemented in a block `EQUATION` of the section `[LONGITUDINAL]` of a script `Mlxtran`.

We consider parametric functions of time of the form $f(t;\phi)$, where ϕ is a vector of structural parameters.

The `[LONGITUDINAL]` section starts with the definition of the vector of input parameters ϕ in a list `input`.

Then, the simulx script should provide

- the model which is used (i.e. the name of the `Mlxtran` or PharmML script),
- the values of the components of ϕ,
- the list of desired outputs and the times at which they should be evaluated.

Results are returned as a list of data frames.

Example

We consider in this example a model where two functions f_1 and f_2 depending on a vector of parameters $\phi = (ka, V, k)$ are defined:

$$f_1(t) = \frac{D}{V} e^{-k\,t}$$
$$f_2(t) = \frac{D\,k_a}{V(k_a - k)} (e^{-k\,t} - e^{-k_a\,t})$$

where $D = 100$ is given.

This model is implemented in the model file `model/analytical.txt`:

We will start evaluating only f_1 every 0.1h between time 0 and time 25, with $k_a = 0.5$, $V = 10$ and $k = 0.2$:

```
f <- list(name = 'f1',  time = seq(0, 25, by=0.1))

p <- c(ka = 0.5, V = 10, k = 0.2)

res <- simulx(model     = 'model/analytical.txt',
              parameter = p,
              output    = f)
```

`res` is a list with element `f1` which is a data frame with 251 rows and 2 columns:

```
res$f1[1:5,]
```

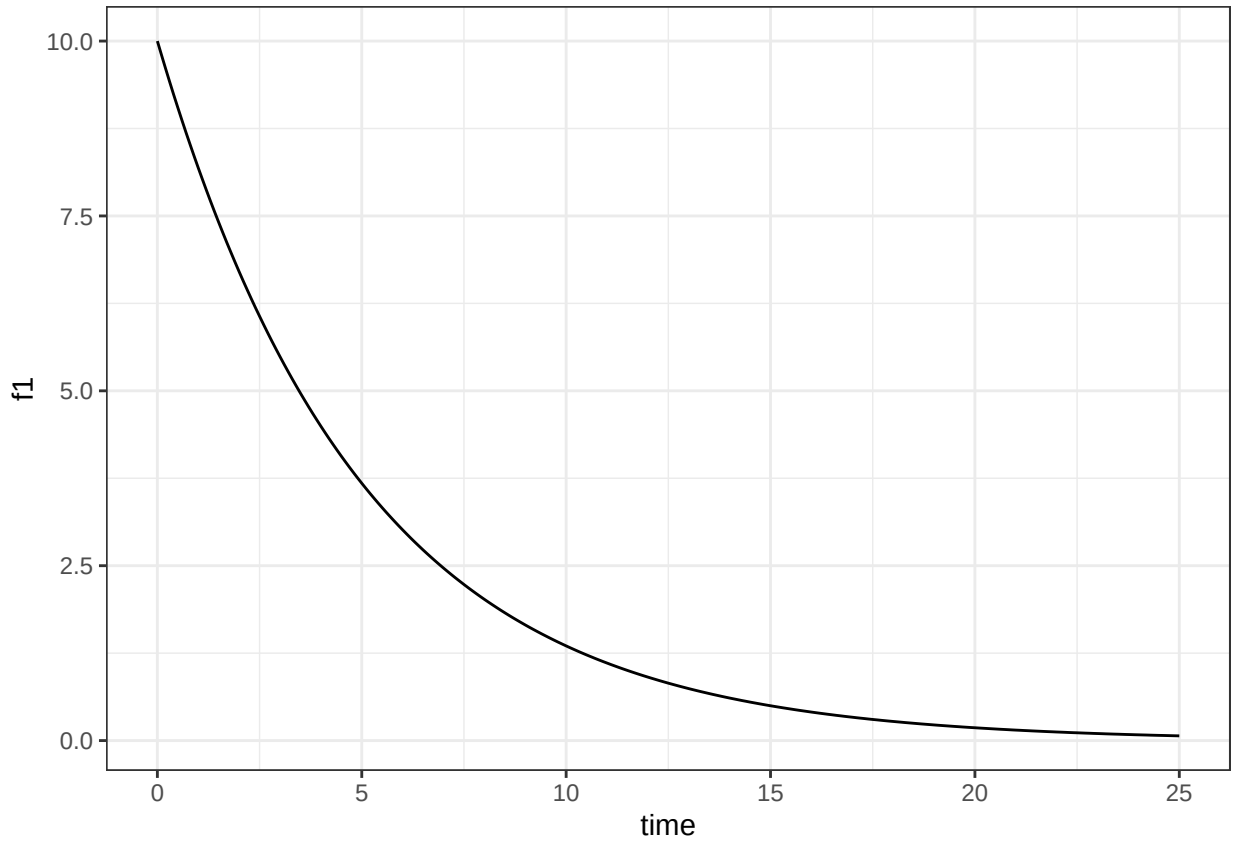
```
##   time        f1
## 1  0.0 10.000000
## 2  0.1  9.801987
## 3  0.2  9.607894
## 4  0.3  9.417645
## 5  0.4  9.231163
```

```
res$f1[248:251,]
```

```
##     time         f1
## 248 24.7 0.07154598
## 249 24.8 0.07012928
## 250 24.9 0.06874063
## 251 25.0 0.06737947
```

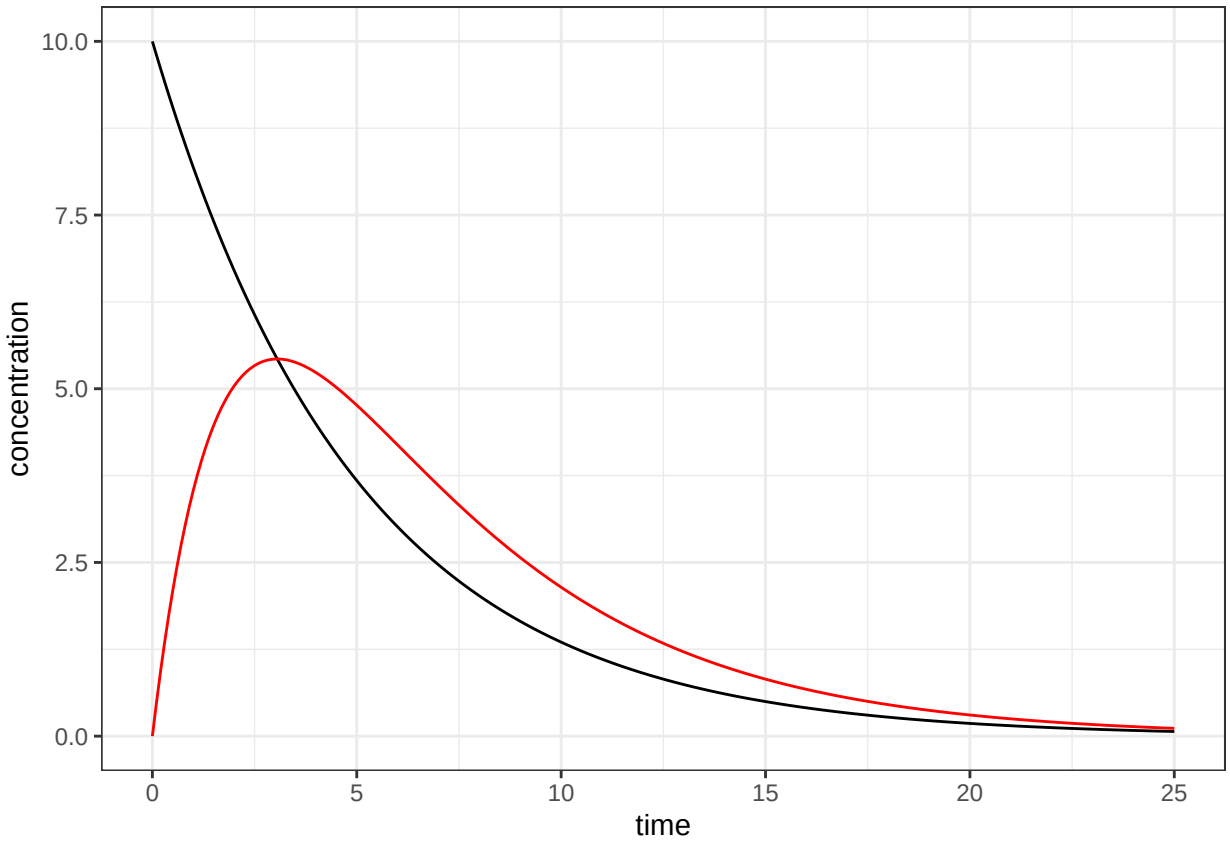
we can plot this data using `ggplot`

```
print(ggplot(data=res$f1) + geom_line(aes(x=time, y=f1)))
```



We way now want evaluate both f_1 and f_2 at the same time points. We can still define the output in a single list, but the field `name` has now two elements `f1` and `f2`:

```
f <- list(name = c('f1','f2'), time = seq(0, 25, by=0.1))

res <- simulx(model     = 'model/analytical.txt',
              parameter = p,
              output    = f)

print(ggplot() + geom_line(data=res$f1, aes(x=time, y=f1), color="black") +
                 geom_line(data=res$f2, aes(x=time, y=f2), color="red") +
                 ylab('concentration'))
```



It can be useful to reshape the data before plotting it with `ggplot`. We merge the two outputs into a unique data frame `r` with three columns: `time`, `f` (which takes the values `f1` or `f2`) and `value`.

```
library(reshape2)
r <- merge(res$f1,res$f2)
r <- melt(r ,  id = 'time', variable.name = 'f', value.name="concentration")
r[c(1:4),]
```

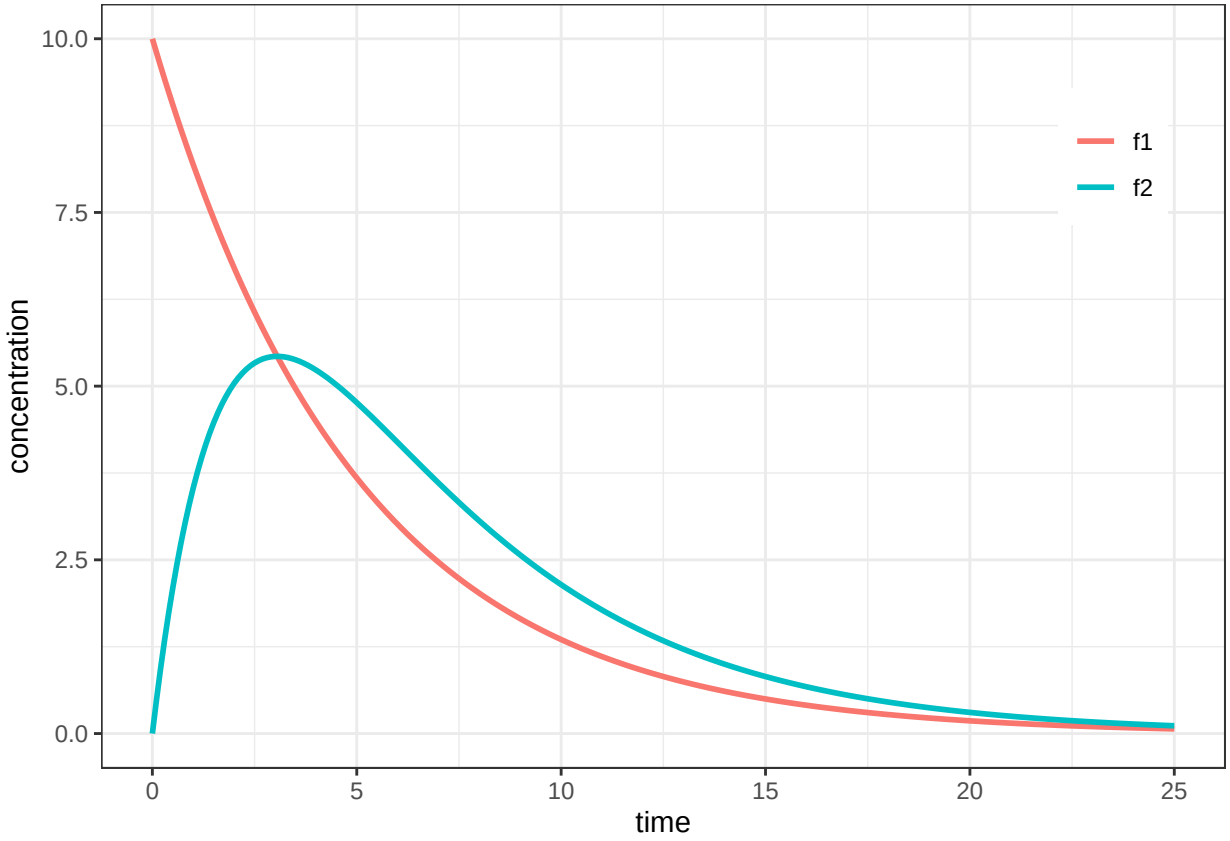
```
##   time  f concentration
## 1  0.0 f1     10.000000
## 2  0.1 f1      9.801987
## 3  0.2 f1      9.607894
## 4  0.3 f1      9.417645
```

```
r[c(250:253),]
```

```
##     time  f concentration
## 250 24.9 f1    0.06874063
## 251 25.0 f1    0.06737947
## 252  0.0 f2    0.00000000
## 253  0.1 f2    0.48282081
```

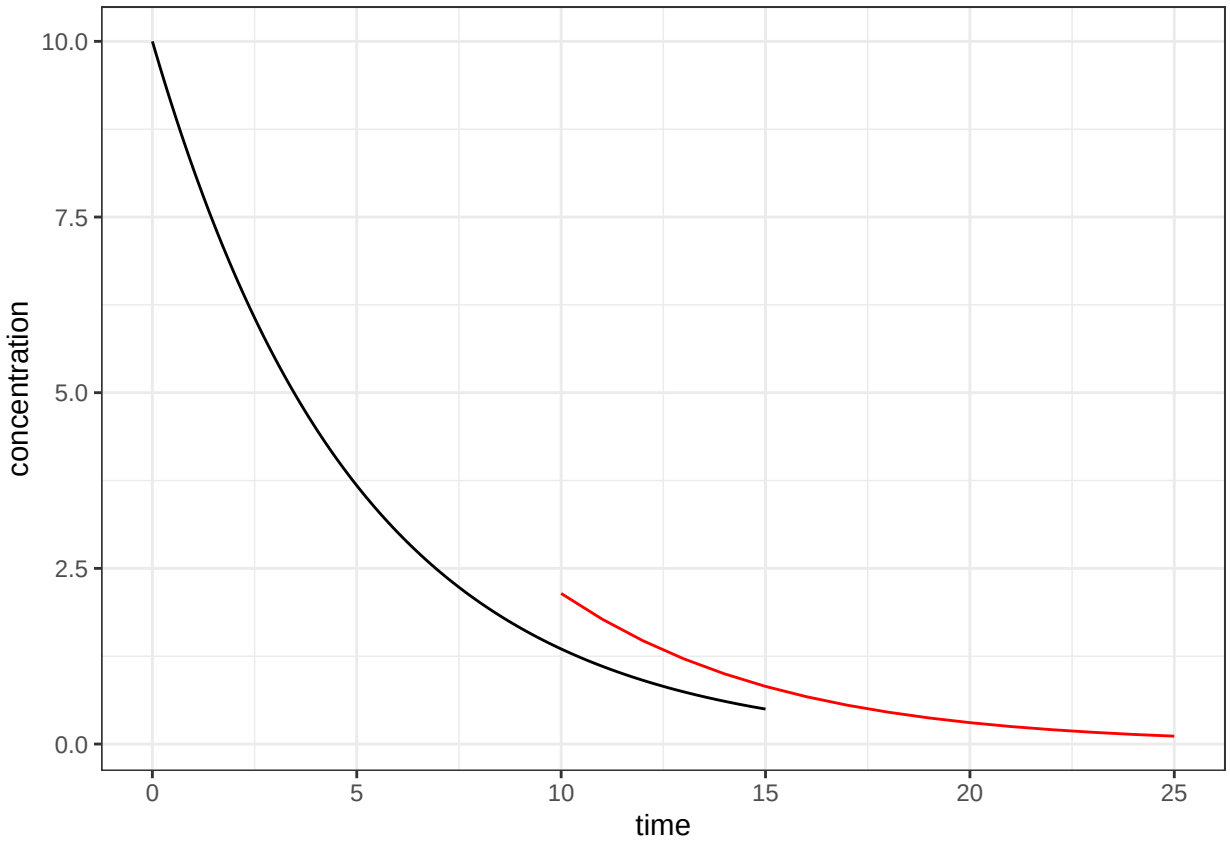
This format is now suitable for `ggplot`:

```
print(ggplot(r, aes(time,concentration)) + geom_line(aes(colour = f),size=1) +
        guides(colour=guide_legend(title=NULL)) + theme(legend.position=c(.9, .8)))
```



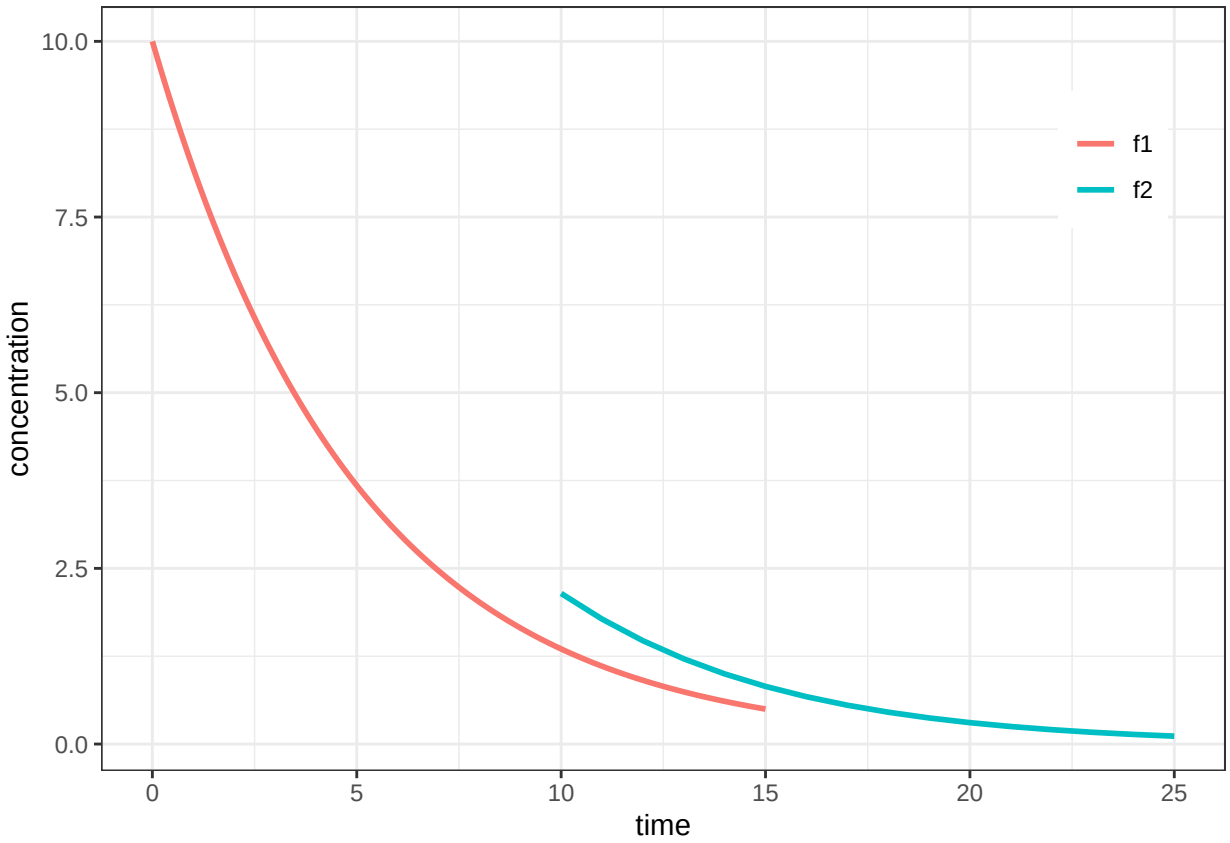
It is also possible to evaluate f_1 and f_2 at different time points. The two outputs should be then defined in two separate lists:

```
f1 <- list(name = 'f1', time = seq(0, 15, by=0.1))
f2 <- list(name = 'f2', time = seq(10, 25, by=1))

res <- simulx(model     = 'model/analytical.txt',
              parameter = p,
              output    = list(f1,f2))

plot(ggplot() + geom_line(data=res$f1, aes(x=time, y=f1), color="black") +
                geom_line(data=res$f2, aes(x=time, y=f2), color="red") +
                ylab('concentration'))
```



We may again prefer to reshape the data before plotting it with `ggplot`

```
r <- merge(res$f1,res$f2,all=TRUE)
r <- melt(r ,  id = 'time', variable.name = 'f', value.name="concentration")
r=r[(!is.na(r$concentration)),]
print(ggplot(r, aes(time,concentration)) + geom_line(aes(colour = f),size=1) +
       guides(colour=guide_legend(title=NULL)) + theme(legend.position=c(.9, .8)))
```


10.0
7.5
5.0
2.5
0.0
concentration
0
5
10
15
20
25
time
f1
f2

Ordinary differential equations

R script: ode.R

Introduction

A system of ordinary differential equations (ODEs) can be implemented in a block `EQUATION` of the section `[LONGITUDINAL]` of a script Mlxtran.

The differentiation variable is t.

The problem is defined by the equations of the derivatives, the initial time point and the initial values of the components. The stiffness of the problem can be defined. The problem accepts input source terms, which allows to bring some discontinuity into the defined derivatives.

Derivatives

The keywords prefixed with ddt_ in front of a variable name, as ddt_a and ddt_b, define the derivatives of an ODE system. The variable names denote the components of the solution. These variables are defined at the whole section level through their derivatives. The derivatives themselves aren't variables but keywords, and cannot be referenced by other equations nor be defined under a conditional statement.

The structure of the ODE system is part of the structure of the whole model and cannot be conditional. Adding conditional intermediate variables for the values of the derivatives easily provides this flexibility.

Initial values

The keyword t0 defines the initial point t_0 of an ODE problem, while the variable names with suffix _0 define the initial values of the system. More precisely, the components are defined for $t \leq t_0$ as the equations provided for the initial values, and for $t > t_0$ they are defined as the solution of the ODE system with initial values fixed at $t = t_0$.

Initial values cannot depend on t, but can be functions of t_0. By default, an initial value is null.

For example, the following model

$$\dot{x}(t) = \frac{\mathrm{d}\,x}{\mathrm{d}\,t}(t) = -k\,x^c, \quad \text{for} \quad t \geq 5 \tag{1}$$

$$x(t) = x_0, \quad \text{for} \quad t \leq 5 \tag{2}$$

is implemented as follows with Mlxtran:

Type of ODE's

The keyword odeType defines the type of the ODE system. The problem can be indicated as being stiff, non-stiff, or linear. By default, the problem is viewed as non-stiff.

Example

We consider in this example a model where two functions f_1 and f_2 depending on a vector of parameters $\phi = (a, b, c)$ are defined by a system of ODEs

$$\dot{f}_1(t) = a\, f_2(t) - \frac{b\, f_1(t)}{1 + c\, f_1(t)} \tag{3}$$

$$\dot{f}_2(t) = \frac{b\, f_1(t)}{1 + c\, f_1(t)} - a\, f_2(t) \tag{4}$$

with the following initial condition:

$$f_1(t) = 10 \quad \text{for } t \leq 0 \tag{5}$$

$$f_2(t) = 0 \quad \text{for } t \leq 0 \tag{6}$$

Here, this model is implemented as an *inline model* in the R script (we could use equivalently the model file "model/ode.txt"):

```
ode.model <- inlineModel("
[LONGITUDINAL]
input = {a, b, c}

EQUATION:
t0     = 0
f1_0   = 10
f2_0   = 0
ddt_f1 = a*f2 - b*f1/(1+c*f1)
ddt_f2 = b*f1/(1+c*f1) - a*f2
")
```

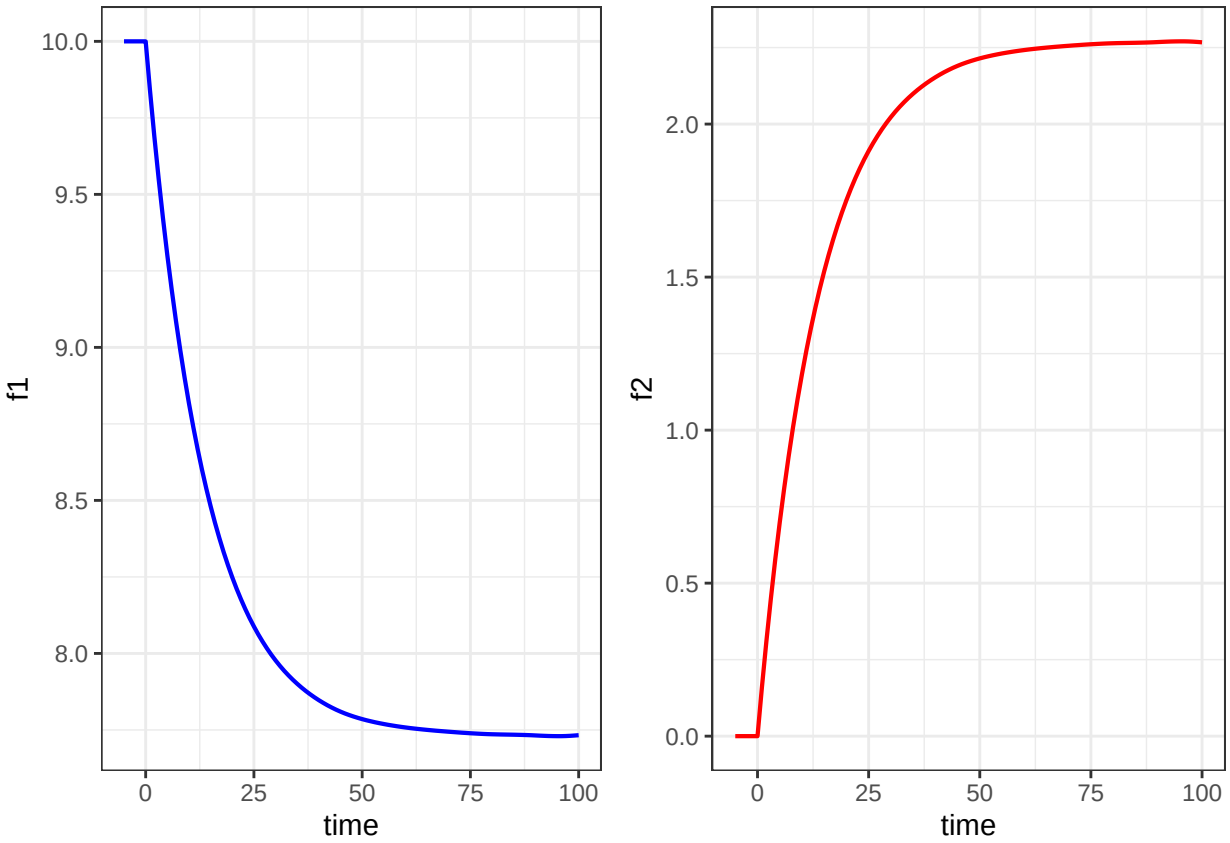
We will use simulx for evaluating f_1 and f_2 every hour between time -5h and time 100h, with $a = 0.07$, $b = 0.1$ and $c = 0.5$:

```
p <- c(a = 0.07, b = 0.1, c = 0.5)

out <- list(name=c('f1', 'f2'), time=-5:100)

res <- simulx(model     = ode.model,
              parameter = p,
              output    = out)
```

`res` is a list with two fields `f1` and `f2` which are two data frames with 106 rows and 2 columns.

We can now plot this data:

```
library(gridExtra)
plot1=ggplot(data=res$f1, aes(x=time, y=f1)) + geom_line(colour='blue',size=0.75)
plot2=ggplot(data=res$f2, aes(x=time, y=f2)) + geom_line(colour='red',size=0.75)
grid.arrange(plot1, plot2, ncol=2)
```


f1
10.0
9.5
9.0
8.5
8.0
0
25
50
75
100
time
f2
2.0
1.5
1.0
0.5
0.0
0
25
50
75
100
time

Delay differential equations

R script: dde.R

Introduction

A system of delay differential equations (DDEs) can be implemented in a block `EQUATION` of the section `[LONGITUDINAL]` of a script Mlxtran.

Mlxtran provides the command delay(x,T) where x is a one-dimensional component and T is the explicit delay. Therefore, DDEs with a nonconstant past of the form

$$\dot{x}(t) = f(x(t), x(t-T_1), x(t-T_2), \ldots), \quad \text{for} \quad t \geq 0 \tag{7}$$
$$x(t) = x_0(t) \quad \text{for} \quad \min(T_k) \leq t \leq 0 \tag{8}$$

can be solved.

Example

We consider in this example a model where two functions f_1 and f_2 depending on a vector of parameters $\phi = (a, b, \tau_1, \tau_2)$ are defined by a system of DDEs. Here, τ_1 and τ_2 are delays.

$$\dot{f}_1(t) = a\, f_2(t-\tau_2) - b\, f_1(t-\tau_1) \tag{9}$$
$$\dot{f}_2(t) = b\, f_1(t-\tau_1) - a\, f_2(t-\tau_2) \tag{10}$$

with the following initial condition:

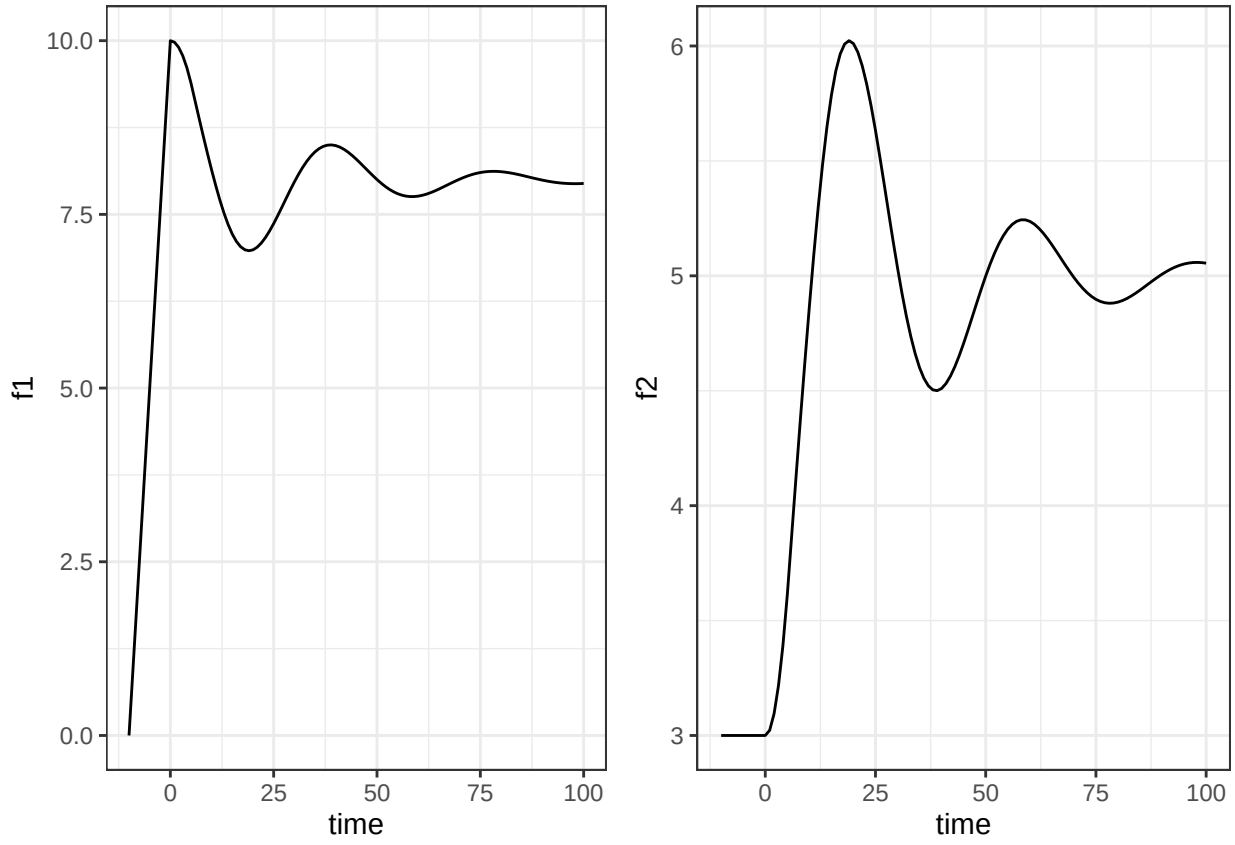
$$f_1(t) = \max(10+t, 0) \quad \text{for } t \leq 0 \tag{11}$$
$$f_2(t) = 3 \qquad \text{for } t \leq 0 \tag{12}$$

This model can easily be implemented with Mlxtran:

```
dde.model <- inlineModel("
[LONGITUDINAL]
input = {a, b, tau1, tau2}

EQUATION:
  t0 = 0
f1_0 = 10+t
f2_0 = 3
ddt_f1 = a*delay(f2,tau2) - b*delay(f1,tau1)
ddt_f2 = b*delay(f1,tau1) - a*delay(f2,tau2)
")
```

We will use `simulx` for evaluating f_1 and f_2 every hour between time -10h and time 100h, with $a = 0.08$, $b = 0.05$, $\tau_1 = 5$ and $\tau_2 = 10$:

```
p <- c(a = 0.08, b  = 0.05, tau1 = 5, tau2 = 10)

out <- list(name=c('f1', 'f2'), time=seq(-10,100,by=1))

res <- simulx(model     = dde.model,
              parameter = p,
              output    = out)

plot1=ggplot(data=res$f1, aes(x=time, y=f1)) + geom_line()
plot2=ggplot(data=res$f2, aes(x=time, y=f2)) + geom_line()
gridExtra::grid.arrange(plot1, plot2, ncol=2)
```



Model with regression variables

R script: regression.R

Mlxtran code: model/regression1a.txt, regression1b.txt, regression2.txt, regression3a.txt, regression3b.txt, regression4.txt

---

Introduction

A regression variable is a variable x which is a given function of time, which is not defined in the model but which is used in the model.

A regression variable is defined in the `R` script as a vector $(x_1, x_2, \ldots, x_m)$ together with a vector of times $(t_1, t_2, \ldots, t_m)$, where $x_j = x(t_j)$ is the value of x at time t_j.
Then, this regression variable is used as an input of the Mlxtran code.

x is only defined at time points $t_1, t_2, \ldots, t_m$ but x is a function of time that should be defined for any t. Then, Mlxtran defines the function x by intepolating the given values $(x_1, x_2, \ldots, x_m)$. In the current version of Mlxtran, interpolation is performed by using the last given value:

$$x(t) = x_j \quad \text{for} \quad t_j \leq t < t_{j+1}$$

Examples

Example 1

Consider a `Emax` model where the effect $E(t)$ at time t is function of the concentration $C(t)$:

$$E(t) = E_{\max} \frac{C(t)}{EC_{50} + C(t)}$$

Assume that C is given at times $t_1, t_2, \ldots, t_m$. We therefore use C as a *regression variable* in the Mlxtran code `model/regression1a.txt`

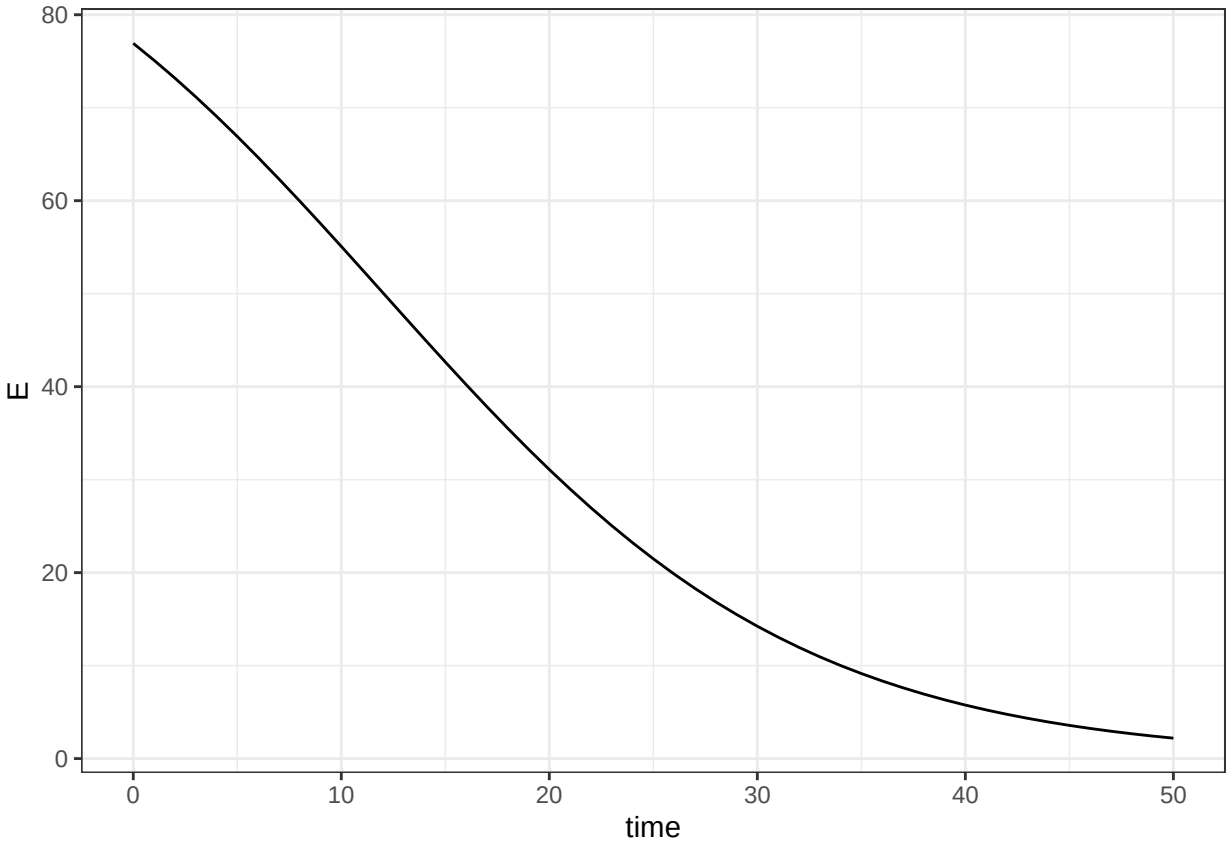
A regression variable is defined as an input argument regression of simulx. It is a list with three elements: name, time and value.

Assume in this example that

$$x(t) = e^{-0.1\,t}$$

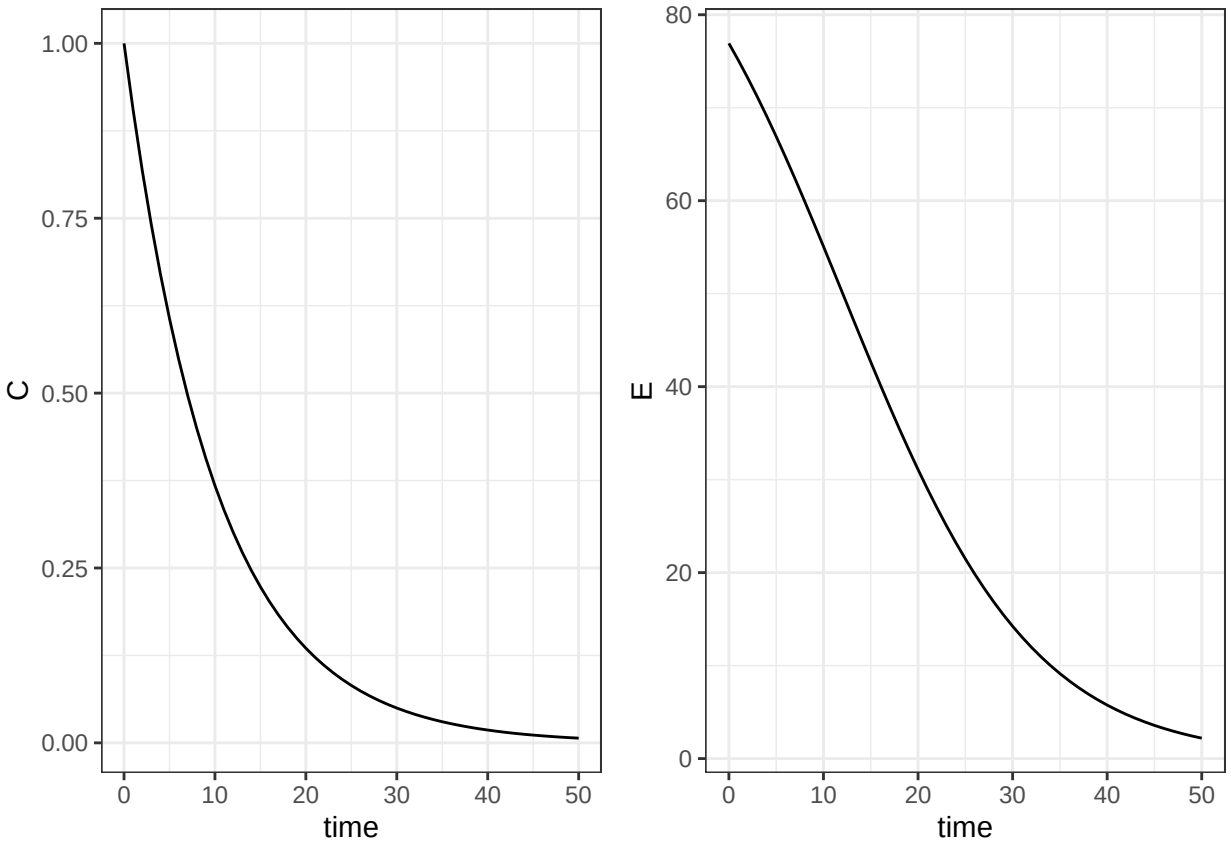
is given for $t = 0, 1, 2, \ldots, 50$.

```
t   <- seq(0,50,by=1)
reg <- list(name='C',
            time=t,
            value=exp(-0.1*t))

out <- list(name='E',
            time=t)

res <- simulx( model     = "model/regression1a.txt",
               parameter = c(Emax=100, EC50=0.3),
               regressor = reg,
               output    = out)

plot(ggplot(data=res$E) + geom_line(aes(x=time, y=E)))
```



If we want to define the regression variable as an output of the model, then, we have to define a new variable which will be defined as an output of the model:

```
library(gridExtra)

out <- list(name=c('E','Cout'),
            time=t)

res <- simulx( model     = "model/regression1b.txt",
               parameter = c(Emax=100, EC50=0.3),
               regressor = reg,
               output    = out)

names(res[2]) <- "C"
names(res$C)  <- c("time","C")
plot1 <- ggplot(data=res$C) + geom_line(aes(x=time, y=C))
plot2 <- ggplot(data=res$E) + geom_line(aes(x=time, y=E))
grid.arrange(plot1, plot2, ncol=2)
```

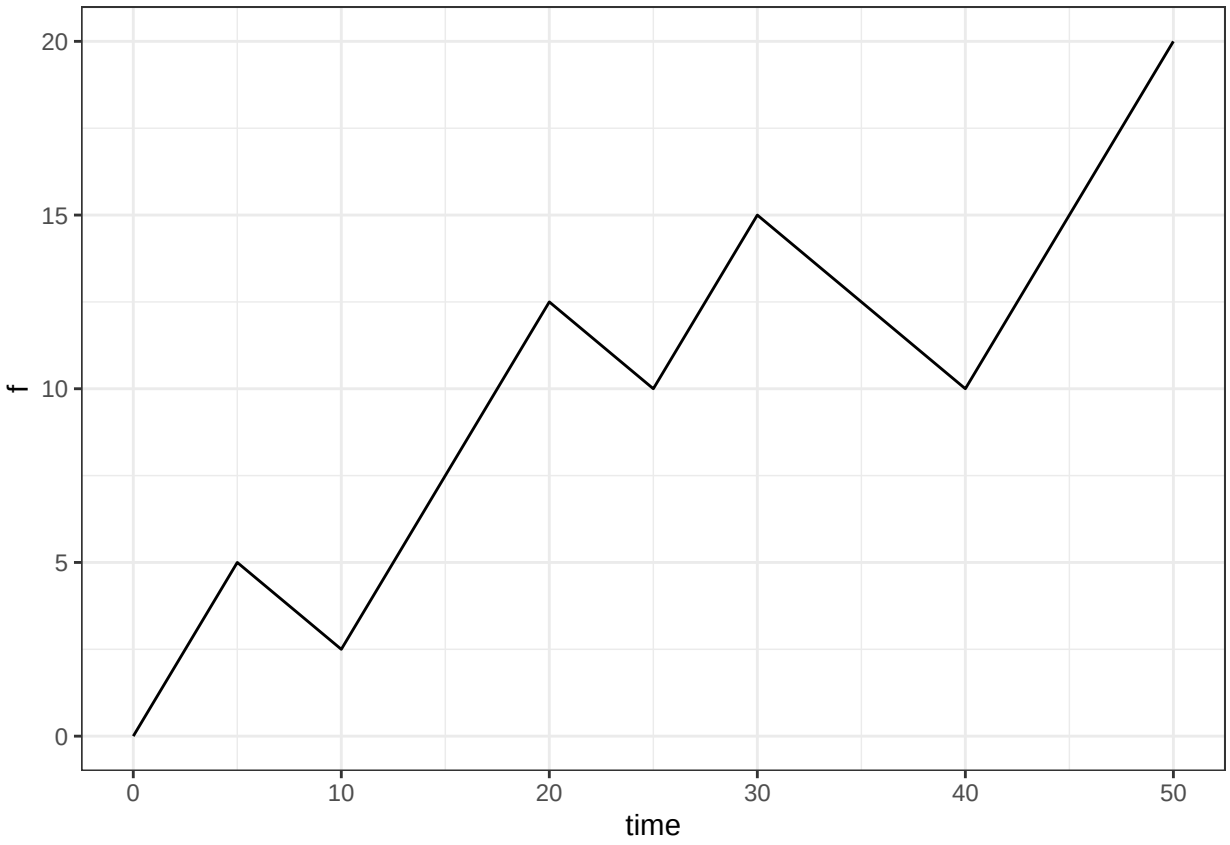


Example 2

A regression variable can also be used as a state. Model `regression2.txt` assumes that there exist two states -1 and $+1$, such that the derivative of a variable f depends on the state (here, $\dot{f}(t) = a$ if $x(t) = 1$ and $\dot{f}(t) = b$ if $x(t) = -1$):

In this example, the state changes every 5 hours: function f is therefore a piecewise linear function, which slope abruptly changes every 5 hours.

```
x <- list(name='x',
          time=c(0,5,10,20,25,30,40),
          value=c(1,-1,1,-1,1,-1,1))

f <- list(name='f',
          time=seq(0, 50, by=1))

res <- simulx( model     = "model/regression2.txt",
               parameter = c(a=1, b=-0.5),
               regressor = x,
               output    = f)

print(ggplot(data=res$f) + geom_line(aes(x=time, y=f)))
```

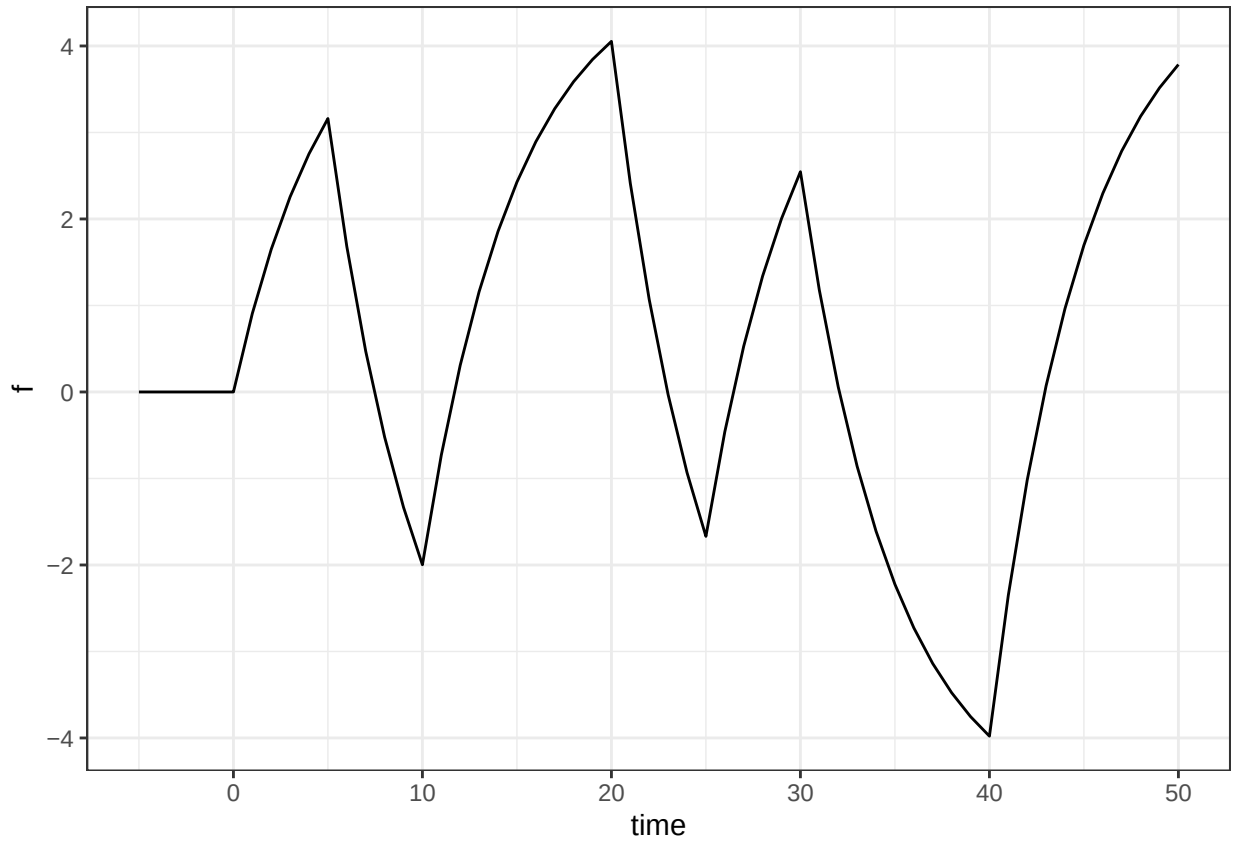


Example 3

In this new model `regression3a.txt`, x is used as rate function.

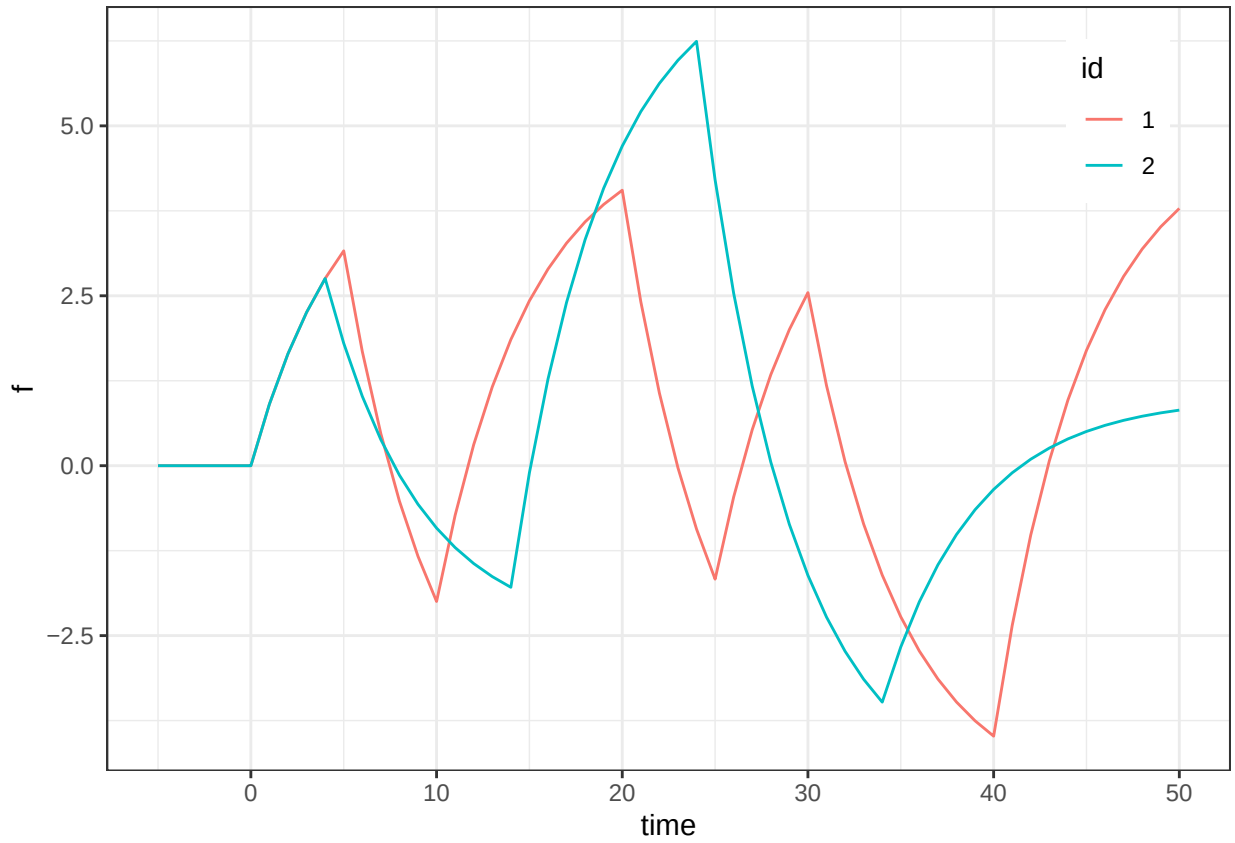
This rate function is a piecewise constant function which changes abruptly every 5 hours.

```
x <- list(name='x',
          time=c(0,5,10,20,25,30,40),
          value=c(1,-1,1,-1,1,-1,1))

f <- list(name='f',
          time=seq(-5, 50, by=1))

res <- simulx( model     = "model/regression3a.txt",
               parameter = c(k=0.2, f0=0),
               regressor = x,
               output    = f)

print(ggplot(data=res$f) + geom_line(aes(x=time, y=f)))
```



Different values of the same regression variable can be defined per group. In this example, the regression variable x is defined every at different time points for groups 1 and 2 (see Defining groups: part I for more details about the use of groups with <ttsimulx):

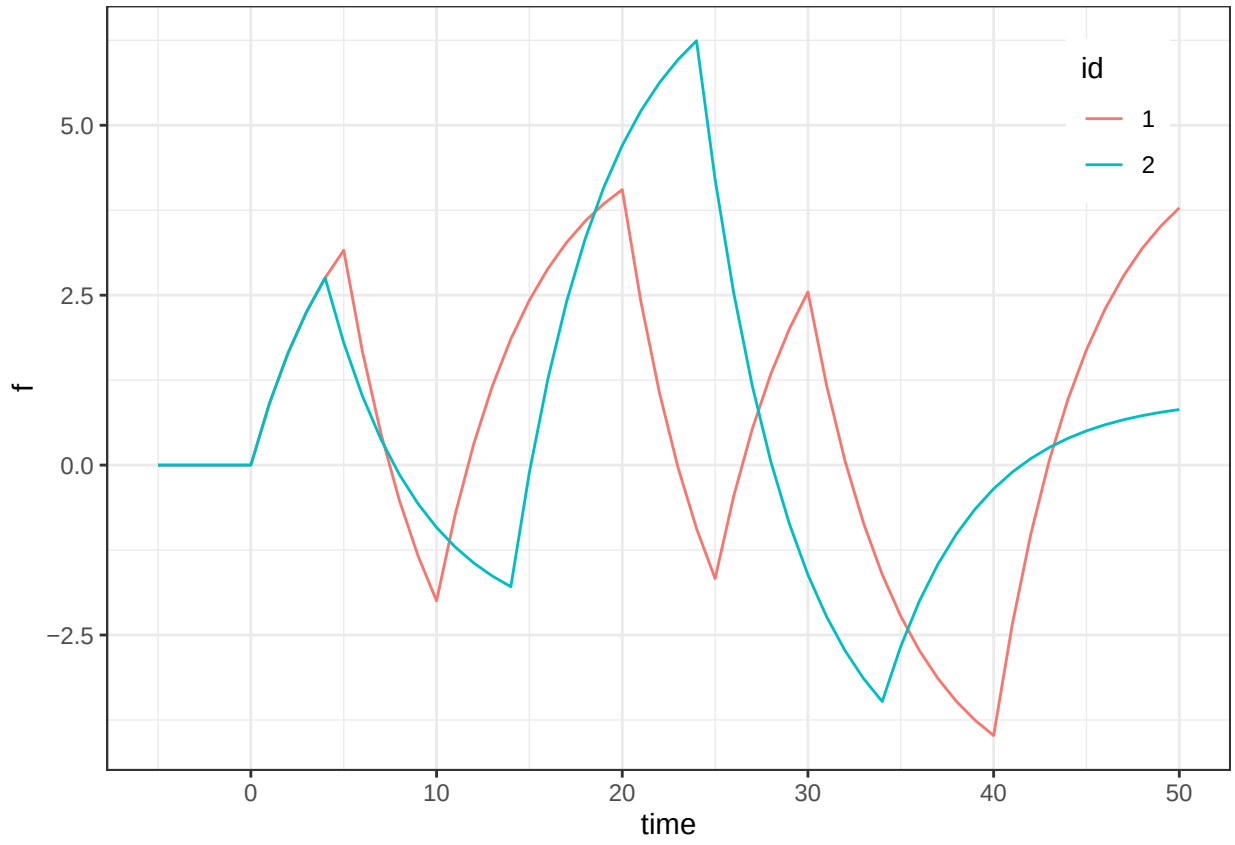
```
x1 <- list(name='x',
           time=c(0,5,10,20,25,30,40),
           value=c(1,-1,1,-1,1,-1,1))
x2 <- list(name='x',
           time=c(0,4,14,24,34),
           value=c(1,-0.5,1.5,-1,0.2))
g1  <- list(regressor = x1)
g2  <- list(regressor = x2)

f <- list(name='f',
          time=seq(-5, 50, by=1))

res <- simulx( model     = "model/regression3a.txt",
               parameter = c(k=0.2, f0=0),
               group     = list(g1,g2),
               output    = f)

print(ggplot(data=res$f) + geom_line(aes(x=time, y=f, colour=id)) +
        theme(legend.position=c(0.9, 0.85)))
```



Alternatively, individual values of the same regression variables can be defined in a datafile (converted here into a data frame). See Using data files and data frames for more details.

```
x  <- inlineDataFrame("
id   time  x
1    0     1
1    5    -1
1   10     1
1   20    -1
1   25     1
1   30    -1
1   40     1
2    0   1.0
2    4  -0.5
2   14   1.5
2   24  -1.0
2   34   0.2
")

f <- list(name='f',
          time=seq(-5, 50, by=1))

res <- simulx( model     = "model/regression3a.txt",
               parameter = c(k=0.2, f0=0),
               regressor = x,
               output    = f)
```

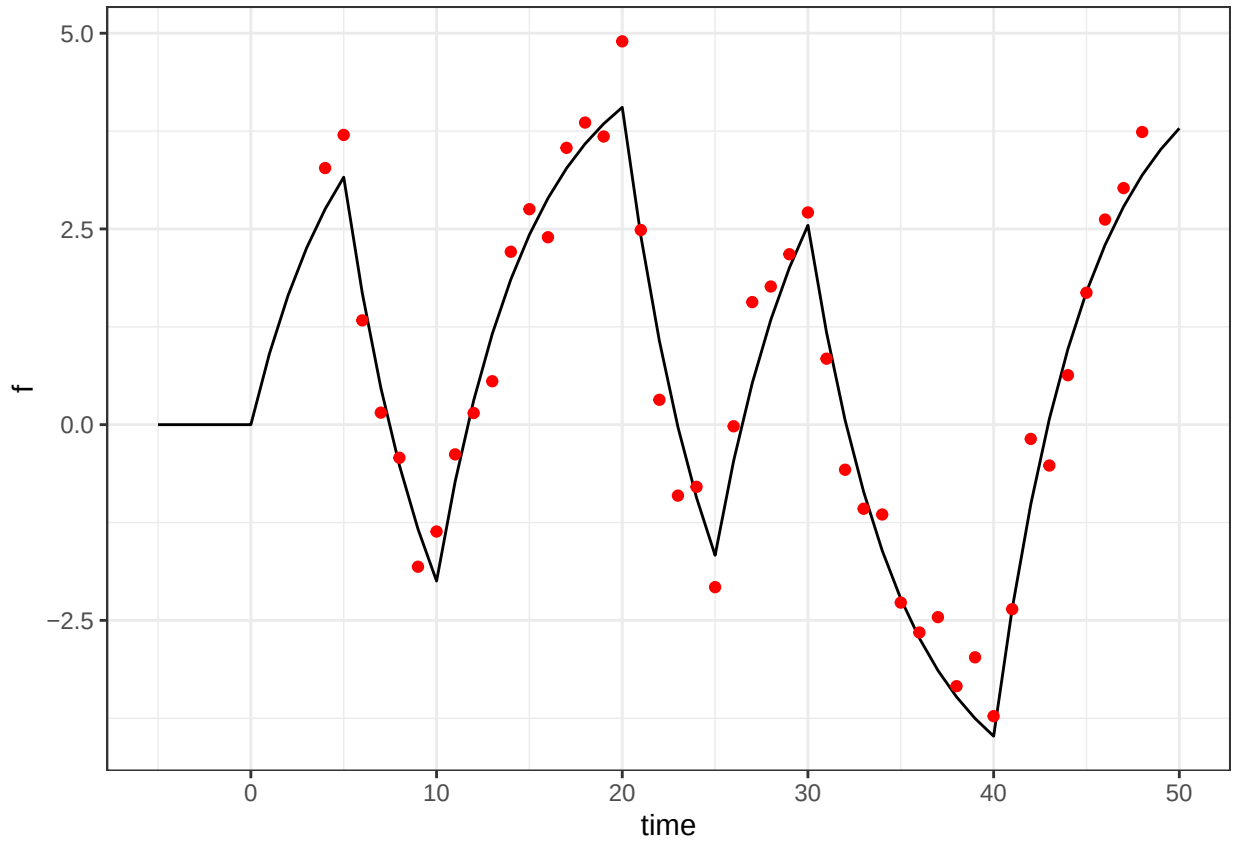
```
print(ggplot(data=res$f) + geom_line(aes(x=time, y=f, colour=id)) +
        theme(legend.position=c(0.9, 0.85)))
```



A regression variable is used to define a function of time f. This function computed at some given time points $t_1, t_2, \ldots, t_n$ can then be used as a prediction for some continuous data $y_1, y_2, \ldots, y_n$, as in the following model `regression3b.txt`:

The regression variable x, the function f and the longitudinal data y can be defined at different times (see Continuous data for more details about the definition of continuous data model).

```
x <- list(name='x',
          time=c(0,5,10,20,25,30,40),
          value=c(1,-1,1,-1,1,-1,1))

y <- list(name='y', time=seq(4, 48, by=1))

res <- simulx( model     = "model/regression3b.txt",
               parameter = c(k=0.2, f0=0, a=0.5),
               regressor = x,
               output    = list(f,y))

print(ggplot() + geom_line(data=res$f, aes(x=time, y=f), colour="black") +
        geom_point(data=res$y, aes(x=time, y=y), colour="red"))
```

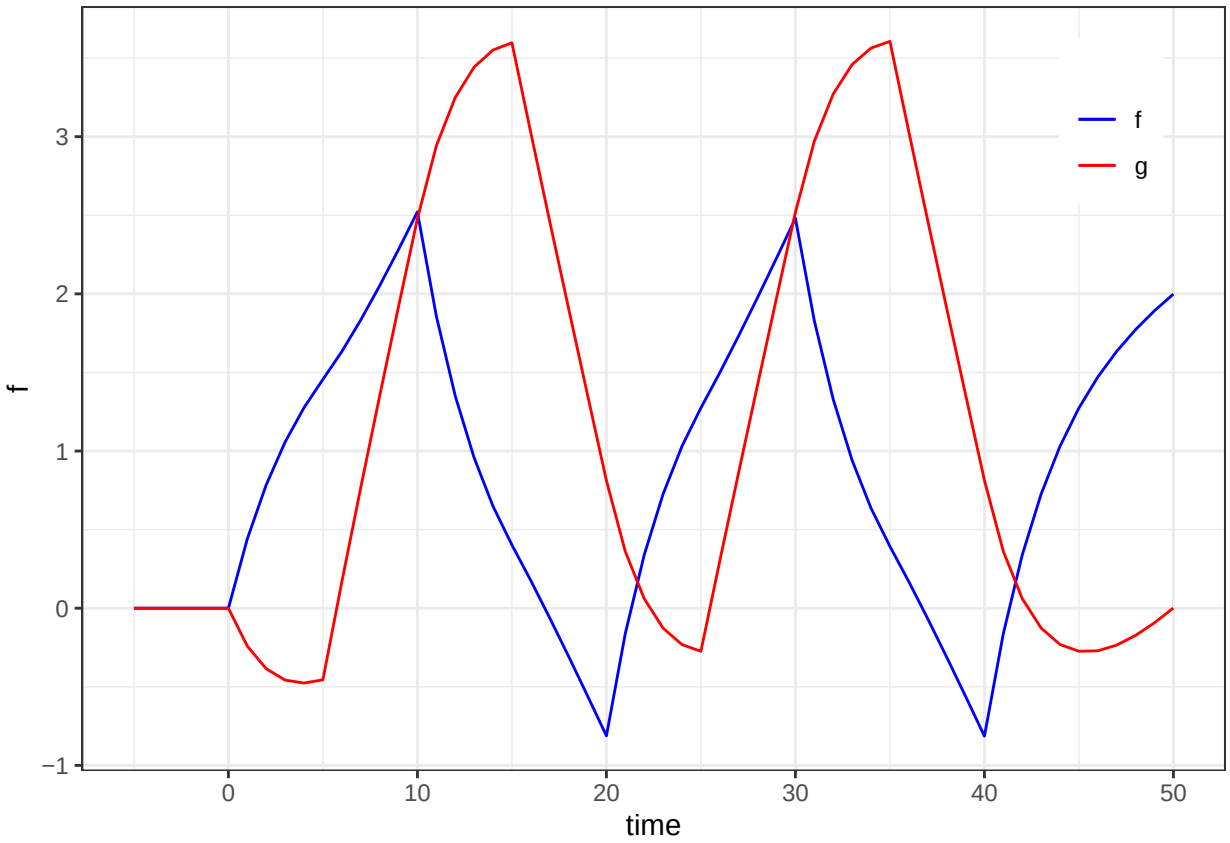


Example 4

Several regression variables can be used in the same model (`regression4.txt`).

These regression variables can be defined at different time points. Each of them will be interpolated using the same method (i.e. using the last given value).

```
x1 <- list(name='x1',
           time=c(0,10,20,30,40),
           value=c(1,-1,1,-1,1)*0.5)
x2 <- list(name='x2',
           time=c(5,15,25,35),
           value=c(1,-1,1,-1)*0.3)

fg <- list(name=c('f','g'),
           time=seq(-5, 50, by=1))

res <- simulx( model     = "model/regression4.txt",
               parameter = c(k1=0.2, k2=0.1, f0=0, g0=0),
               regressor = list(x1, x2),
               output    = fg)

print(ggplot() + geom_line(data=res$f, aes(x=time, y=f, colour="blue")) +
  geom_line(data=res$g, aes(x=time, y=g, colour="red")) +
  scale_colour_manual(name="",values=c('blue'='blue','red'='red'),labels=c('f','g')) +
    theme(legend.position=c(0.9, 0.85)))
```


f
g
3
2
1
0
−1
0
10
20
30
40
50
time

ODE based model with source terms

R script: source1.R ; source2.R

Mlxtran code: model/source1a.txt ; source1b.txt ; source2.txt ; source3a.txt ; source3b.txt ; source4.txt

---

Introduction

A system of ordinary differential equations defines a dynamical system. Then *sources terms* are inputs that modify the dynamics of the system by acting on the *state variables*, i.e on the components of the ODE system.

Source terms are defined in the `R` code, using the input argument treatment of simulx. Their action, i.e. the target variables, are defined either in the PK block of the Mlxtran code or in the `R` code itself, as an element of the input argument treatment.

In the case of a single type of source terms, treatment is a list with the following elements:

- amount : the values of the source terms. Can be a scalar or a vector if different values are used at different times.
- time : a vector of times.
- rate (or tinf) : constant input rate (or input duration = amount/rate). Can be a scalar or a vector (default rate = ∞, i.e. default duration = 0).
- type : type of input, when several inputs and targets are defined in the PK block of the Mlxtran code (default type = 1).
- target : name of the target component of the system (if not defined in the PK block).

Several types of source terms can be defined by defining treatment as a list of elementary source terms.

Examples

Example 1

We consider here the very simple following ODE system:

$$\dot{h}(t) = c$$

with $h(t) = 0$ for $t \leq 0$. Of course, the solution is a straight line with slope c, i.e. $h(t) = c\,t$

We modify this basic system by adding 2 source terms such that:

$$h(30^+) = h(30^-) - 2 \tag{13}$$
$$h(50^+) = h(50^-) + 0.5 \tag{14}$$

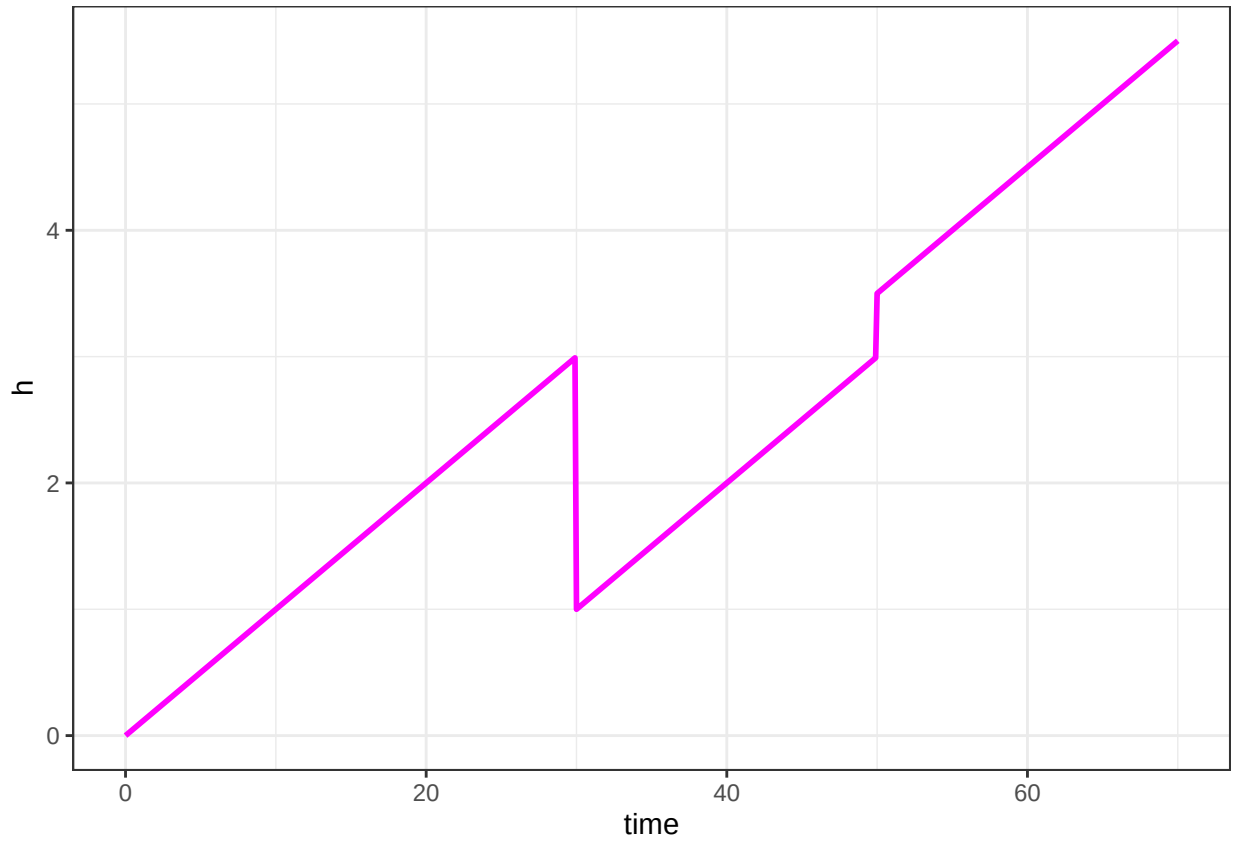
(here, $h(t^+)$ is the value of h *just after* t and $h(t^-)$ the value *just before* t).

This model is implemented in `source1a.txt`: the ODE system is defined in the `EQUATION` block and the target is defined as h in the `PK` block.

We will use simulx for computing h between $t = 0$ and $t = 70$ with $r = 0.1$. The values and times of the inputs are defined in the list treatment:

```
tr <- list(amount = c(-2, 0.5),
           time   = c(30, 50))

out  <- list(name = 'h',
```

```
              time = seq(0, 70, by=0.1))

p    <- c( r = 0.1 )

res <- simulx(model='model/source1a.txt',
              parameter=p,
              output=out,
              treatment=tr)

print(ggplot(data=res$h) + geom_line(aes(x=time,y=h), colour="magenta" ,size=1))
```

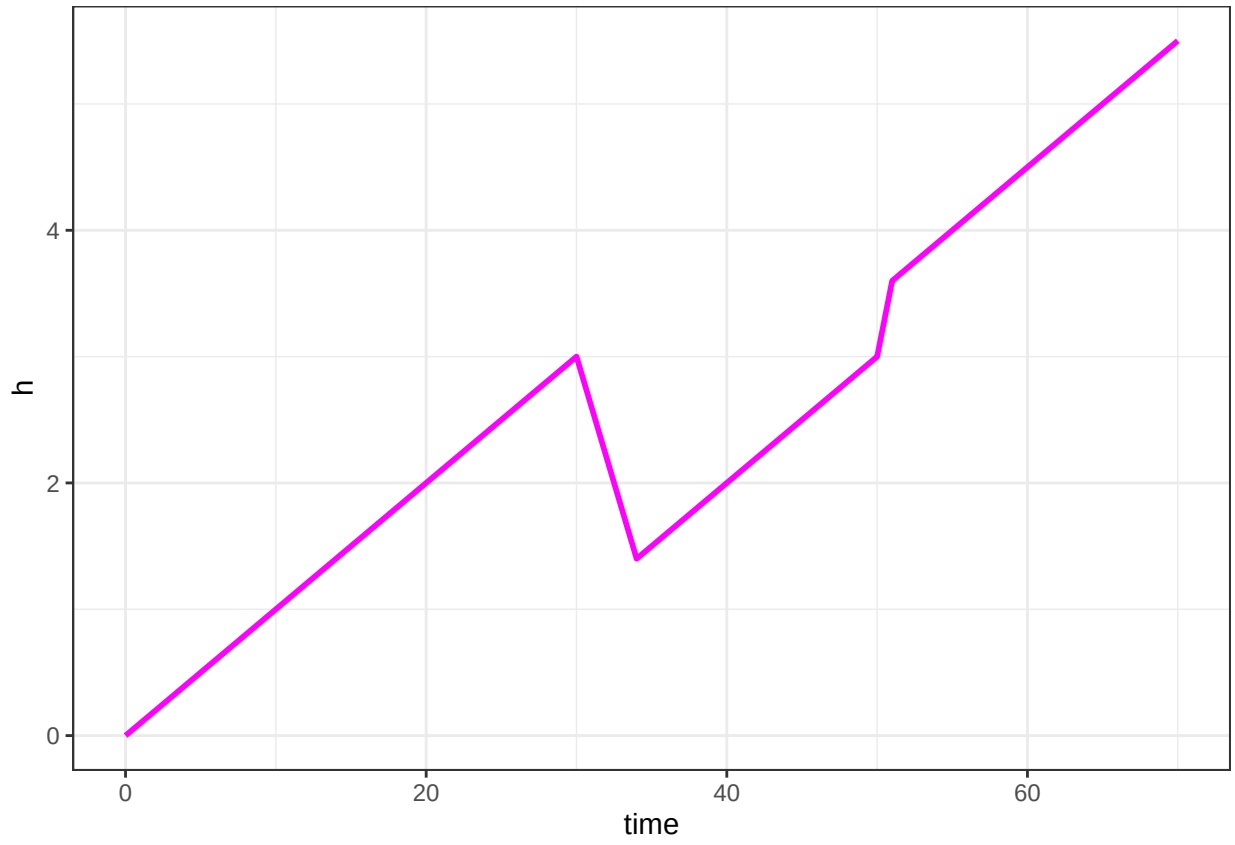


Assume now that the state variable h is not instantaneously modified, but modified with a constant rate $r = 0.5$. The new model is therefore:

$$\dot{h}(t) = c \quad \text{for} \quad t \leq 30 \tag{15}$$
$$\dot{h}(t) = c - 0.5 \quad \text{for} \quad 30 \leq t \leq 34 \tag{16}$$
$$\dot{h}(t) = c \quad \text{for} \quad 34 \leq t \leq 50 \tag{17}$$
$$\dot{h}(t) = c + 0.5 \quad \text{for} \quad 50 \leq t \leq 51 \tag{18}$$
$$\dot{h}(t) = c \quad \text{for} \quad t \geq 30 \tag{19}$$

The ODE system define in the Mlxtran code remains the same, but the source term are now defined by a rate $r = 0.5$:

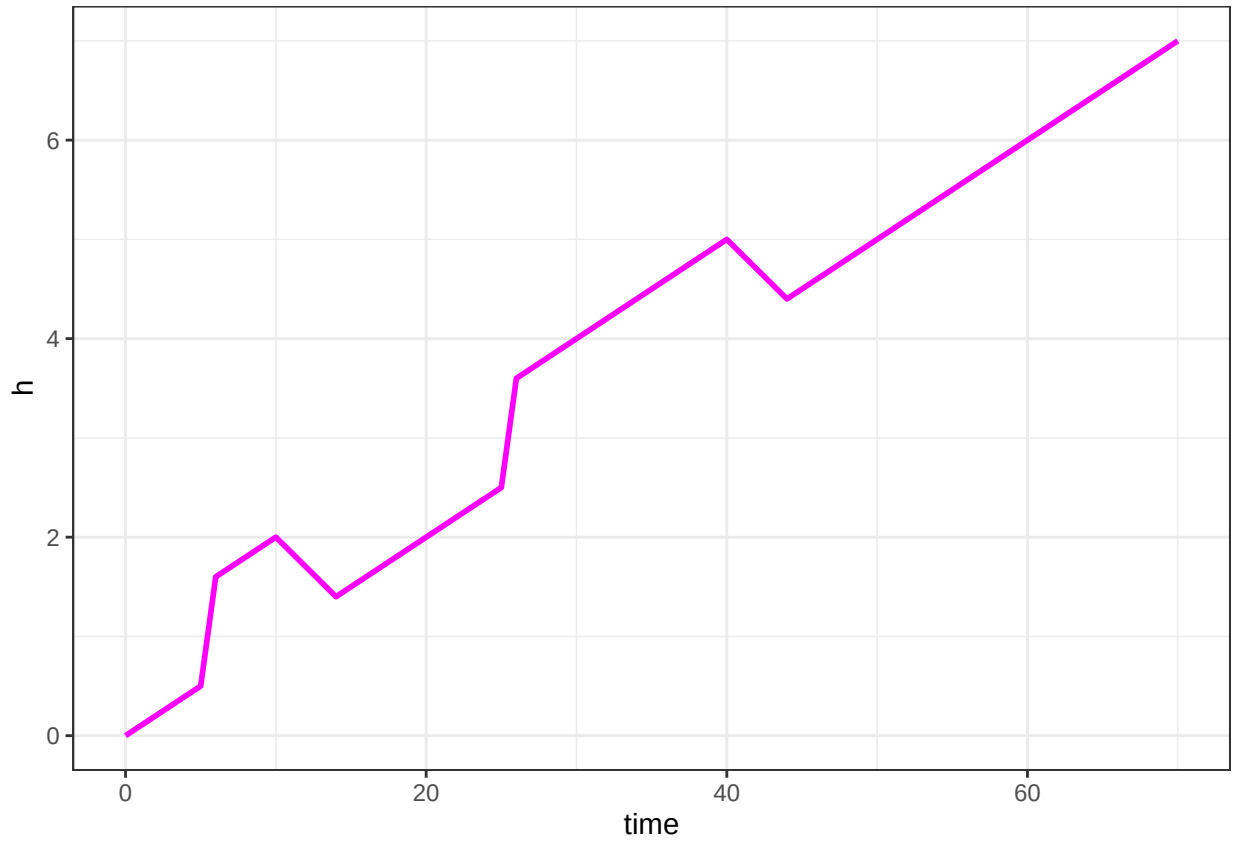
```
tr <- list(amount = c(-2, 0.5),
           rate   = 0.5,
           time   = c(30, 50))

res <- simulx(model='model/source1a.txt',
              parameter=p,
              output=out,
              treatment=tr)

print(ggplot(data=res$h) + geom_line(aes(x=time,y=h), colour="magenta" ,size=1))
```



Several source terms can be combined:

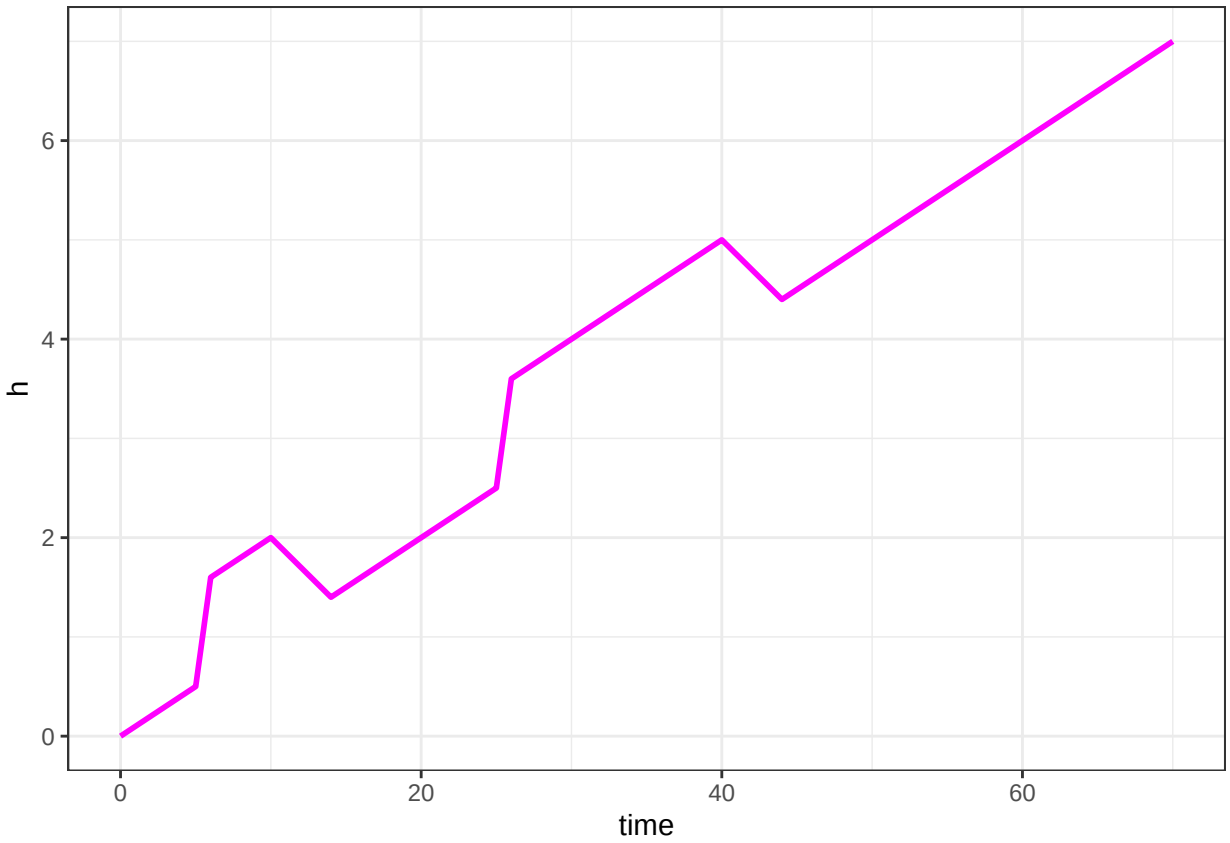
```
ton  <- list(amount = 1,
             rate   = 1,
             time   = c(5, 25))

toff <- list(amount = -1,
             rate   = 0.25,
             time   = c(10, 40))

res <- simulx(model='model/source1a.txt',
              parameter=p,
              output=out,
              treatment=list(ton, toff))
```

```
print(ggplot(data=res$h) + geom_line(aes(x=time,y=h), colour="magenta", size=1))
```



Remark: Instead of defining 2 sequeneces of source terms (*on* and *off*), it would be equivalent to define a unique sequence of source terms:

```
tr  <- list(amount = c(1,-1,1,-1),
            rate   = c(1,0.25,1,0.25),
            time   = c(5, 10, 25, 40))

res <- simulx(model='model/source1a.txt',
              parameter=p,
              output=out,
              treatment=tr)

print(ggplot(data=res$h) + geom_line(aes(x=time,y=h), colour="magenta", size=1))
```



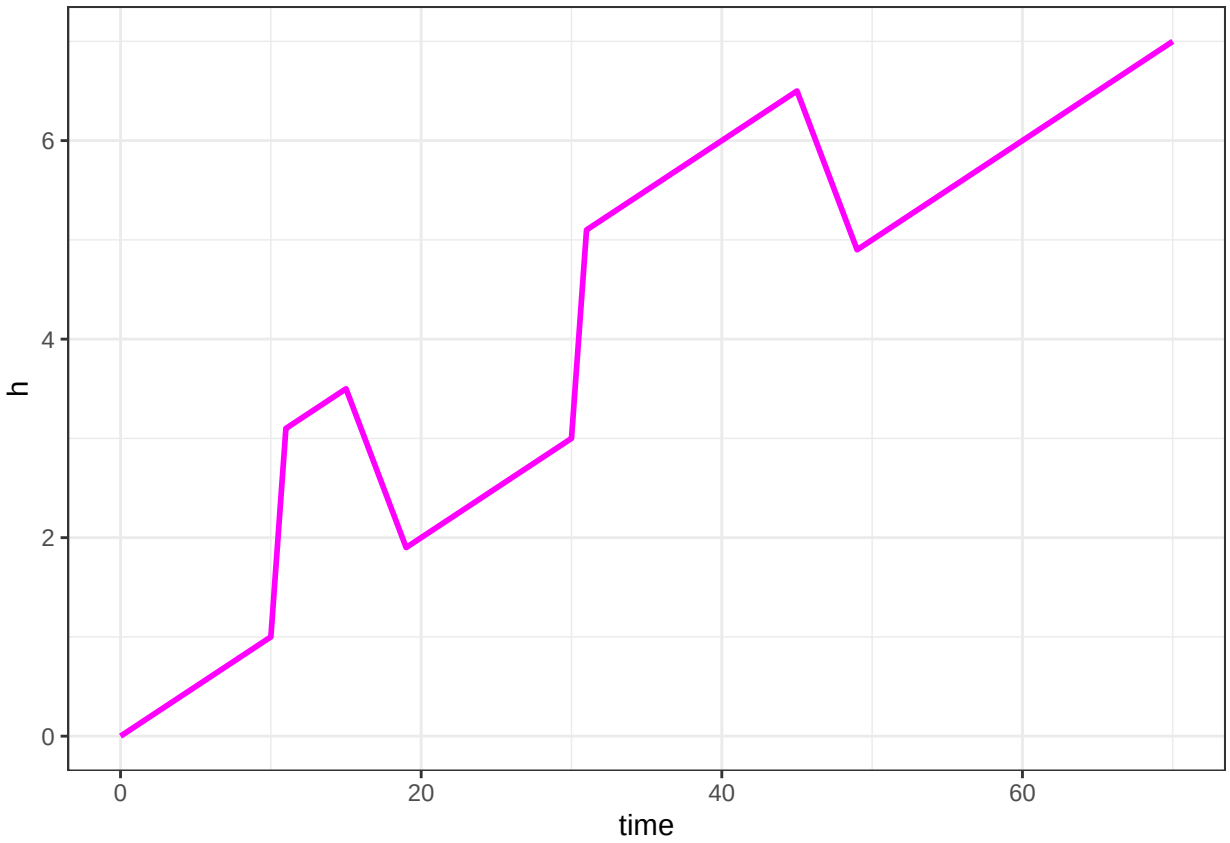
If a source term with value A_s is added at time t_s, we can also assume that the source term acts with a delay τ and a factor F:

$$h((t_s + \tau)^+) = h((t_s + \tau)^-) + F\, A_s \tag{20}$$

The delay τ and the factor F are defined in the block `PK` of `source2.txt` as additional arguments Tlag and p of depot. Then, τ and F are additional parameters of the model defined in the list input:

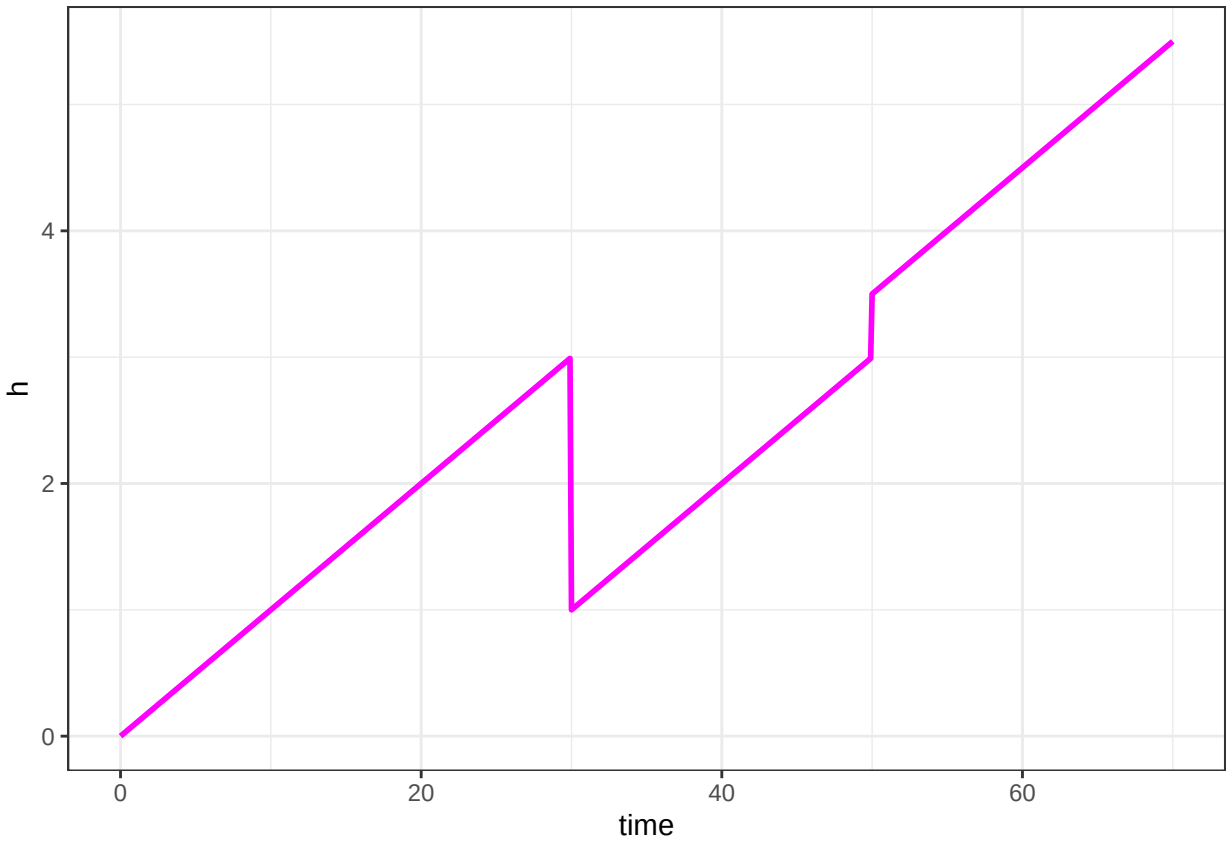
Values of τ and F are defined in the simulx script:

```
p    <- c( r=0.1, tau=5, F=2 )

res <- simulx(model='model/source2.txt',
              parameter=p,
              output=out,
              treatment=list(ton, toff))

print(ggplot(data=res$h) + geom_line(aes(x=time,y=h), colour="magenta", size=1))
```



Remark: It is possible to define the target component in the simulx script. Then, there is no more `PK` block in the Mlxtran code `source1b.txt`:

```
tr <- list(amount = c(-2, 0.5),
           time   = c(30, 50),
           target = 'h')

res <- simulx(model='model/source1b.txt',
              parameter=p,
              output=out,
              treatment=tr)

print(ggplot(data=res$h) + geom_line(aes(x=time,y=h), colour="magenta" ,size=1))
```



Example 2

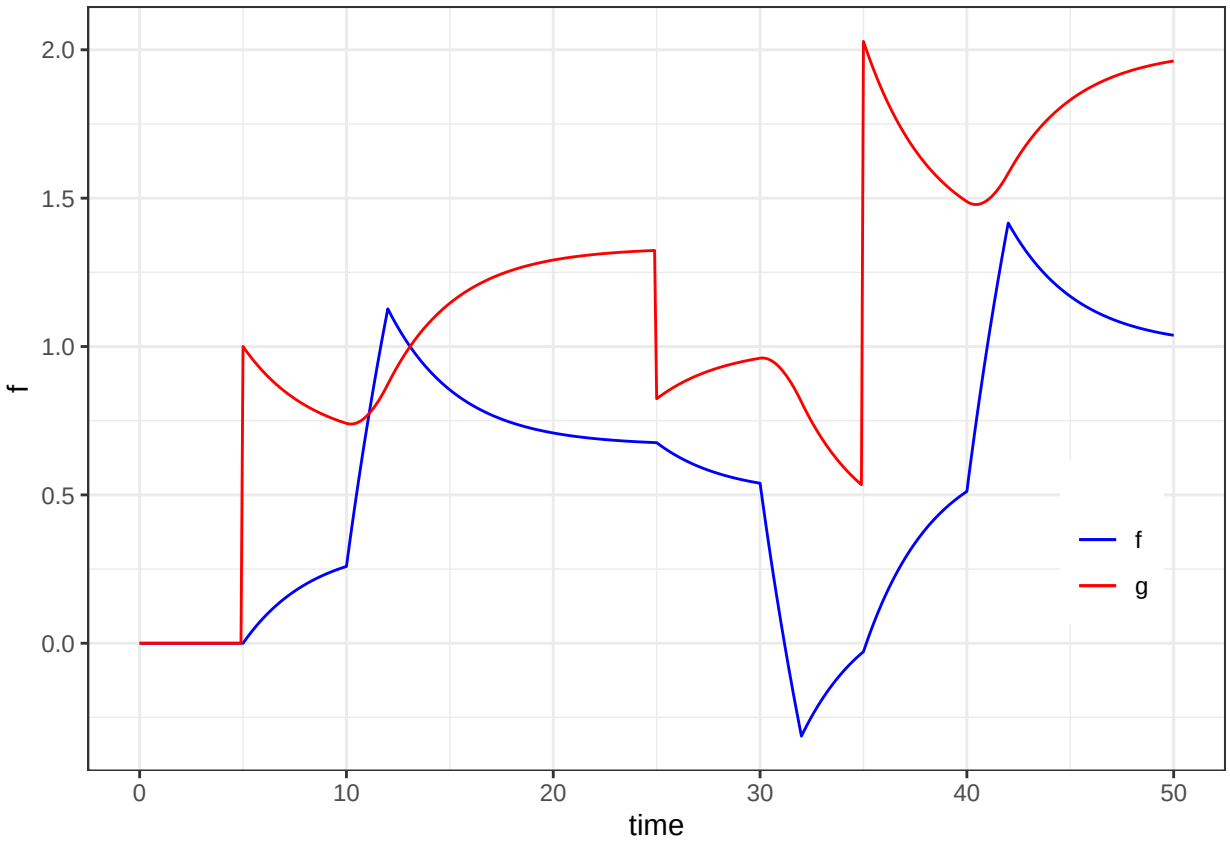
When the system consists of several variables, it is possible to define several source terms that act on different components of the system.

type now defines the type of each source term. A type is associated with a target component of the system.

An example with two types of source terms is implemented in `source3a.txt`. in this example, source terms of type 1 will act on f while source terms of type 2 will act on g.

Each source term defined in the `R` script now has a type:

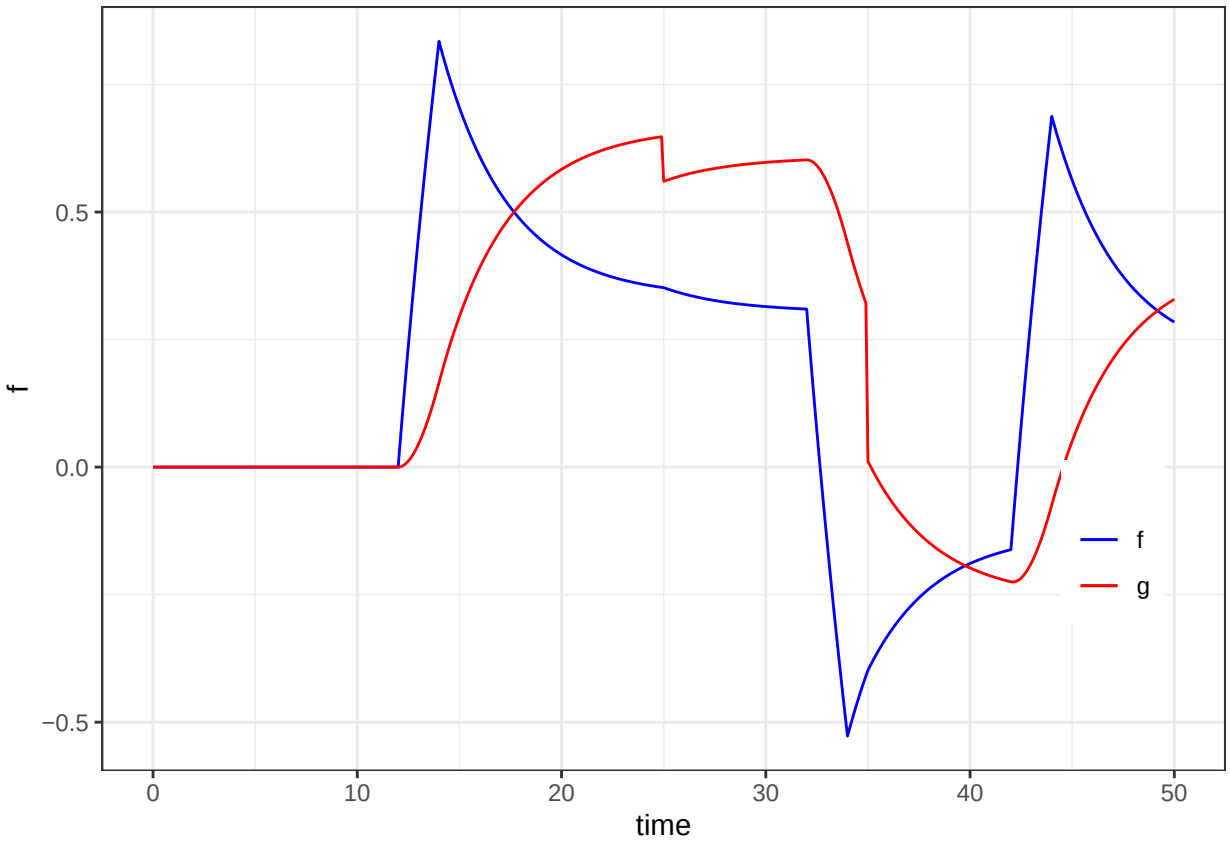
```
s1 <- list(type=1, time=c(10, 30, 40), amount=c(1,-1,1), rate=0.5)
s2 <- list(type=2, time=c( 5, 25, 35), amount=c(1,-0.5,1.5))

fg <- list(name=c('f','g'), time=seq(0, 50, by=0.1))

res <- simulx( model     = "model/source3a.txt",
               parameter = c(k1=0.2, k2=0.1, f0=0, g0=0),
               treatment = list(s1, s2),
               output    = fg)

print(ggplot() + geom_line(data=res$f, aes(x=time, y=f, colour="blue")) +
        geom_line(data=res$g, aes(x=time, y=g, colour="red")) +
        scale_colour_manual(name="",values=c('blue'='blue','red'='red'),labels=c('f','g')) +
        theme(legend.position=c(0.9, 0.3)))
```



Each source term may act with a delay and a multiplicative factor that can be function of the system itself.

Assume for instance that source terms of type 1 act on f with a delay τ and that source term of type 2 at time t_s act on g with a multiplicatve factor $f(t_s)/2$. This new model is implemented in `source4.txt`.

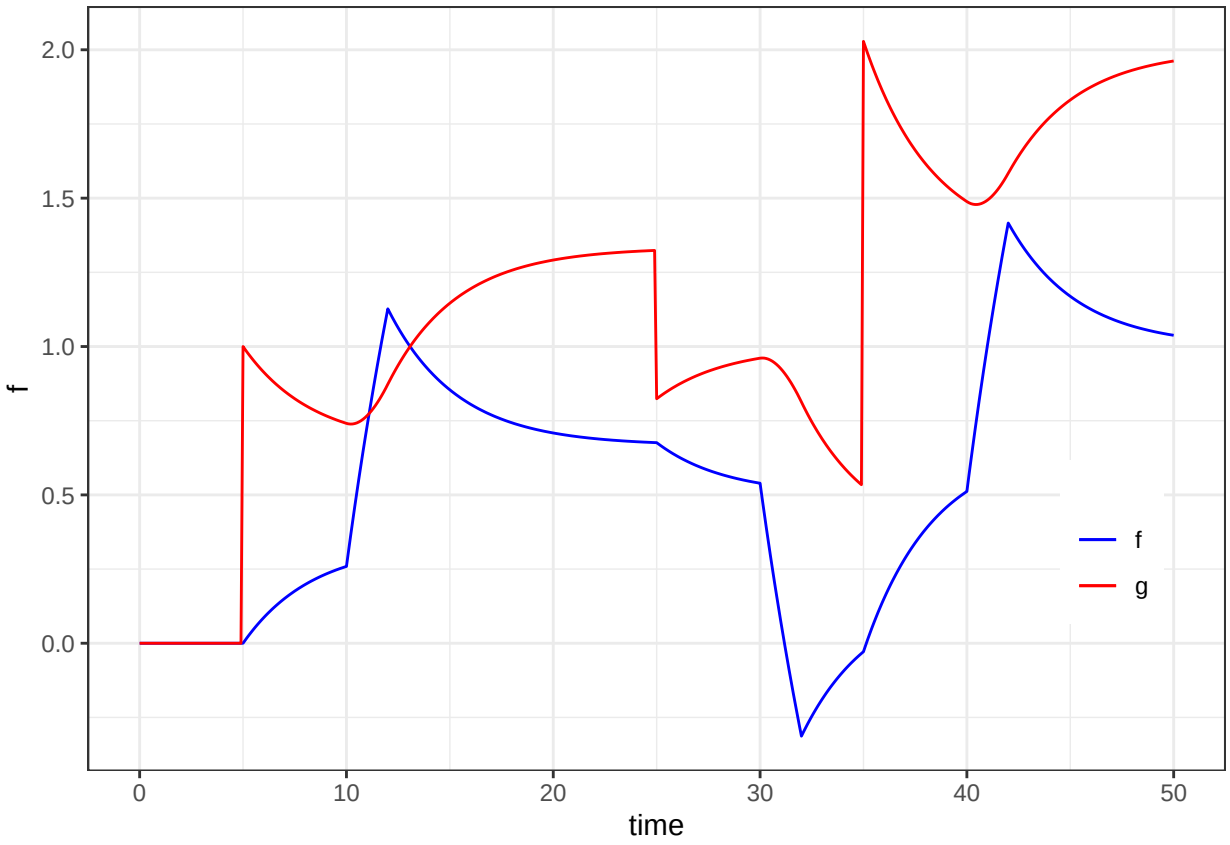
```
res <- simulx( model     = "model/source4.txt",
               parameter = c(k1=0.2, k2=0.1, f0=0, g0=0, tau=2),
               treatment = list(s1, s2),
               output    = fg)

print(ggplot() + geom_line(data=res$f, aes(x=time, y=f, colour="blue")) +
        geom_line(data=res$g, aes(x=time, y=g, colour="red")) +
        scale_colour_manual(name="",values=c('blue'='blue','red'='red'),labels=c('f','g')) +
        theme(legend.position=c(0.9, 0.3)))
```



Another version of model `source3a.txt` is implemented in `source3b.txt`. There is no `PK` block in this model and only the dynamical system is defined in the block `EQUATION`:

Then, both target components are defined with the input argument treatment of simulx:

```
s1 <- list(target="f", time=c(10, 30, 40), amount=c(1,-1,1), rate=0.5)
s2 <- list(target="g", time=c( 5, 25, 35), amount=c(1,-0.5,1.5))

res <- simulx( model     = "model/source3b.txt",
               parameter = c(k1=0.2, k2=0.1, f0=0, g0=0),
               treatment = list(s1, s2),
               output    = fg)

print(ggplot() + geom_line(data=res$f, aes(x=time, y=f, colour="blue")) +
        geom_line(data=res$g, aes(x=time, y=g, colour="red")) +
        scale_colour_manual(name="",values=c('blue'='blue','red'='red'),labels=c('f','g')) +
        theme(legend.position=c(0.9, 0.3)))
```


2.0
1.5
1.0
0.5
0.0
f
0
10
20
30
40
50
time
f
g

Simulation of PK models: single route of administration

R script: pk1.R

Mlxtran code: model/pk1a.txt ; model/pk1b.txt

Introduction

Once a drug is administered, we usually describe subsequent processes within the organism by the pharmacokinetics (PK) process known as *ADME*: absorption, distribution, metabolism, excretion.

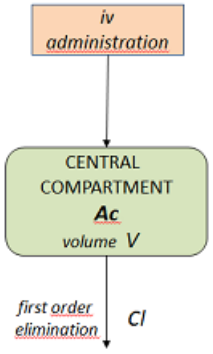
A PK model is a dynamical system mathematically represented by a system of ordinary differential equations (ODEs) which describe transfers between compartments and elimination from the central compartment.

See this web animation for more details.

Example

The PK model

Let us start with a very basic one compartment model for intraveinous (iv) administration with linear elimination



$$\dot{A}_c(t) = -k\, A_c(t) \tag{21}$$
$$A_c(t) = 0 \quad \text{for} \quad t < 0 \tag{22}$$

Here, $A_c(t)$ and $C_c(t) = A_c(t)/V$ are, respectively, the amount and the concentration of drug in the central compartment at time t.

Doses can then arrive in the central compartment at any times τ_1, $\tau_2, \ldots$. An iv bolus administration assumes that

$$A_c(\tau_j^+) = A_c(\tau_j^-) + D_j$$

where D_j is the amount of drug administrated at time τ_j.

On the other hand, an iv infusion with rate R_j assumes that

$$A_c(t) = A_c(\tau_j) + \min(R_j\,(t - \tau_j), D_j),$$

for $\tau_j \leq t \leq \tau_{j+1}$

The Mlxtran model file

This model is implemented in the file `pk1a.txt` using the `pkmodel` function, which recognizes the model according to the list of its input arguments.

Remark: The `pkmodel` function is extremely convenient for implementing easily about 150 basic PK models. Nevertheless, there exists several other ways to implement the same PK model with Mlxtran.

It is possible, for instance, to use some PK macros defined in the block `PK`:

It is also possible to implement the PK model as a system of ODEs in the block `EQUATION` and define the target compartment (here the central compartment) in the block `PK`.

> The `pkmodel`function and the PK macros should be preferred when it is possible since they use an analytic solution when the system is linear instead of soving an ODE system.

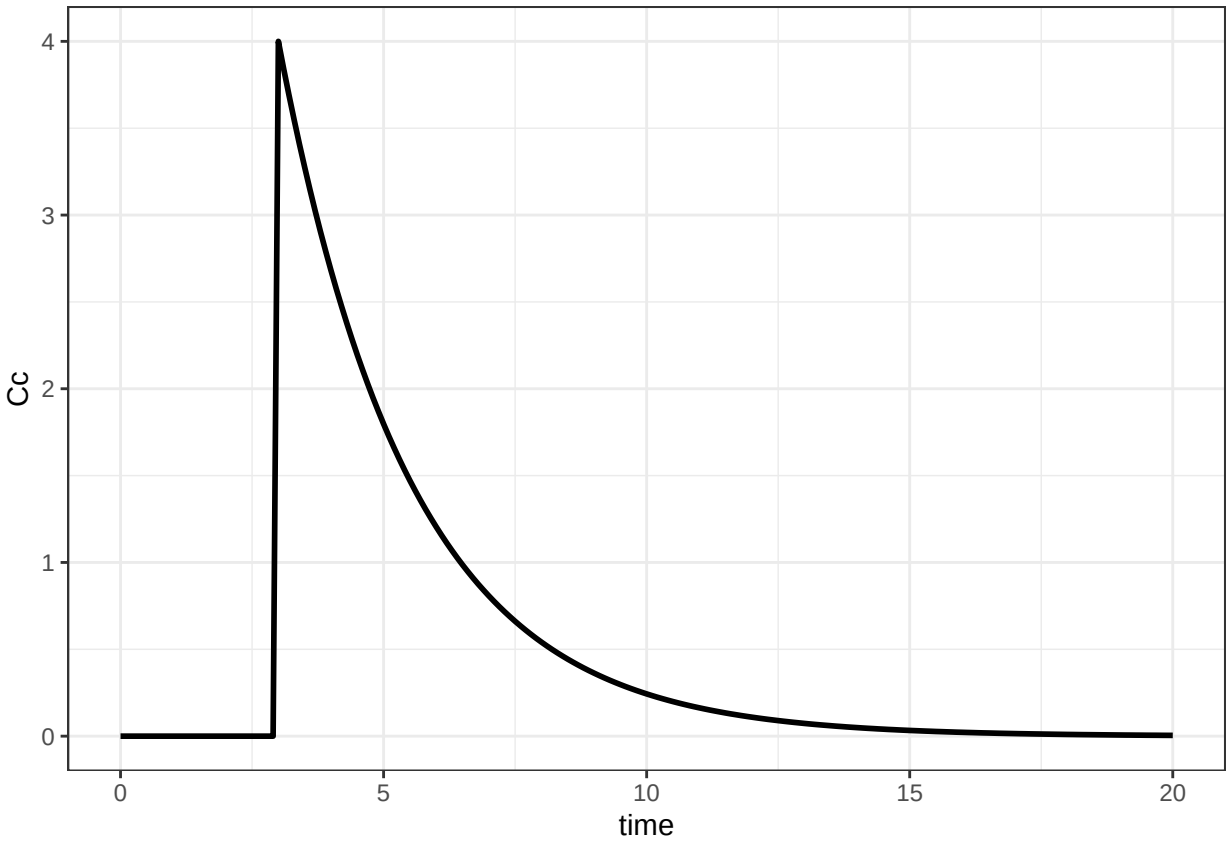
Single dose administration

Let us now use this model with simulx for computing the concentration in the central compartment between time 0 and 20, when a dose of 40g is administered at time 40 as an iv bolus. We will assume the following PK parameters in the next examples:

$$V = 10\text{l} \quad ; \quad Cl = 4\text{l/h}$$

The dosage regimen is defined in the input argument treatment of simulx. For an iv bolus administration, treatment has two fields: time, the time(s) of administration, and amount, the amount(s) of drug.

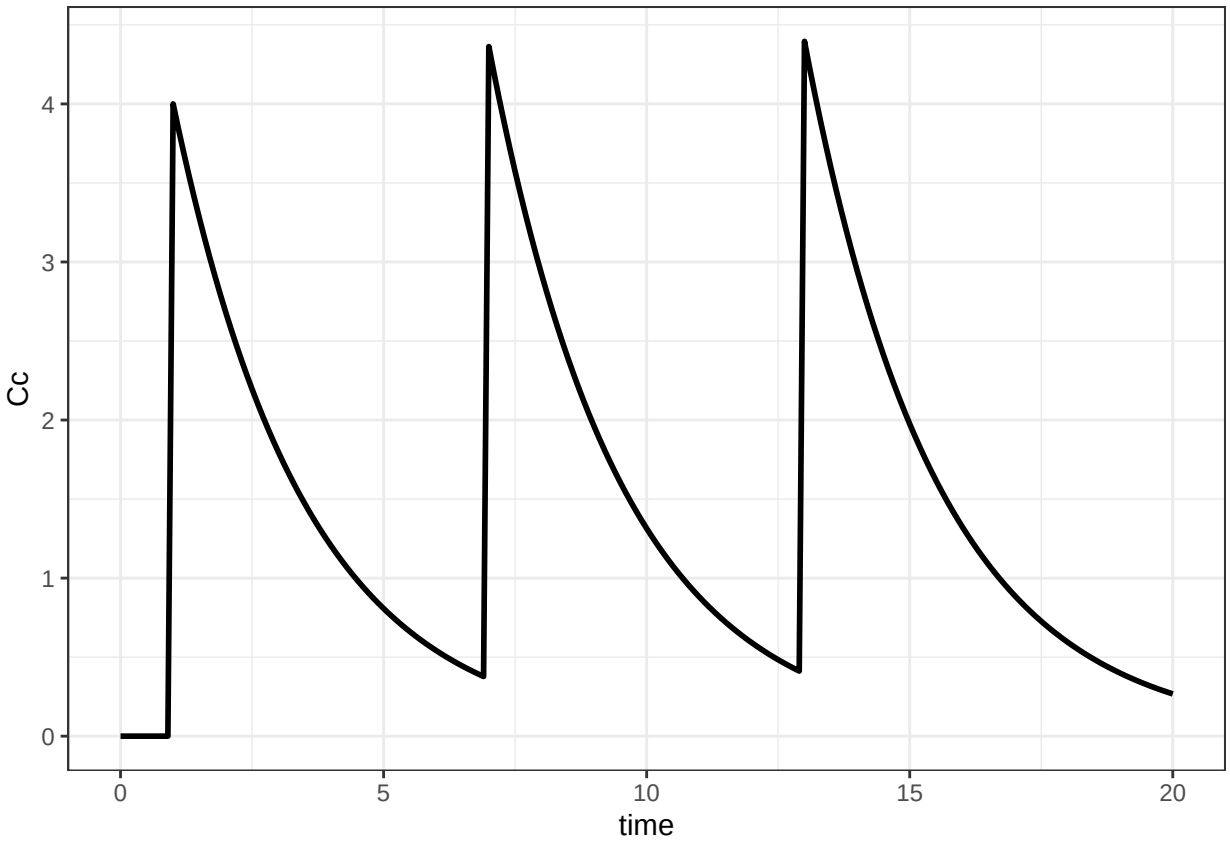
```
adm <- list(time=3, amount=40)

Cc  <- list(name='Cc',time=seq(from=0, to=20, by=0.1))
p   <- c(V=10, Cl=4)

res <- simulx(model='model/pk1a.txt',
              parameter=p,
              output=Cc,
              treatment=adm)

print(ggplot(data=res$Cc, aes(x=time, y=Cc)) + geom_line(size=1))
```



Multiple doses administration

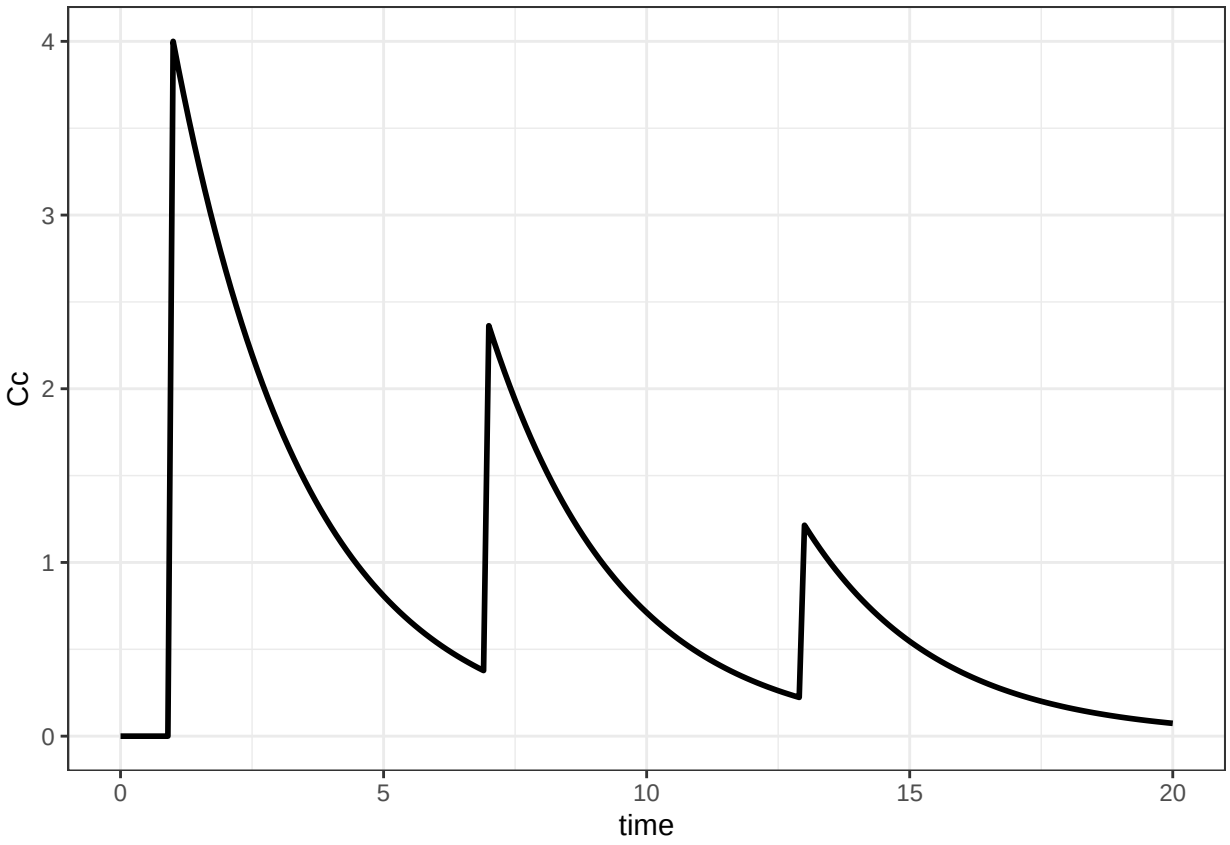
Multiple doses of a same amount are obtained by defining adm$time as a vector

```
adm <- list(time=c(1, 7, 13), amount=40)
res <- simulx(model='model/pk1a.txt', parameter=p, output=Cc, treatment=adm)
print(ggplot(data=res$Cc, aes(x=time, y=Cc)) + geom_line(size=1))
```



Different amounts can be used by defining adm$amount as a vector.

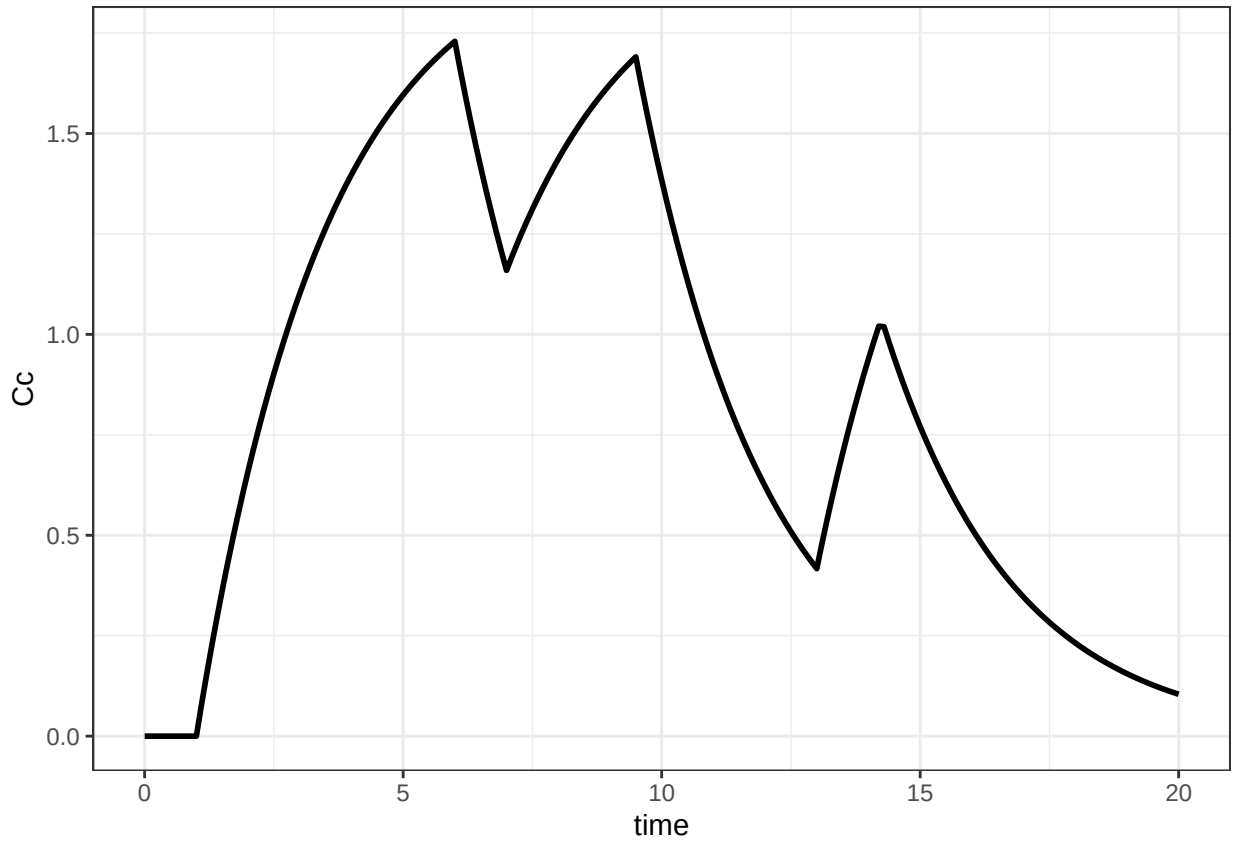
```
adm <- list(time=c(1, 7, 13), amount=c(40, 20, 10))
res <- simulx(model='model/pk1a.txt', parameter=p, output=Cc, treatment=adm)
print(ggplot(data=res$Cc, aes(x=time, y=Cc)) + geom_line(size=1))
```



Infusion administration

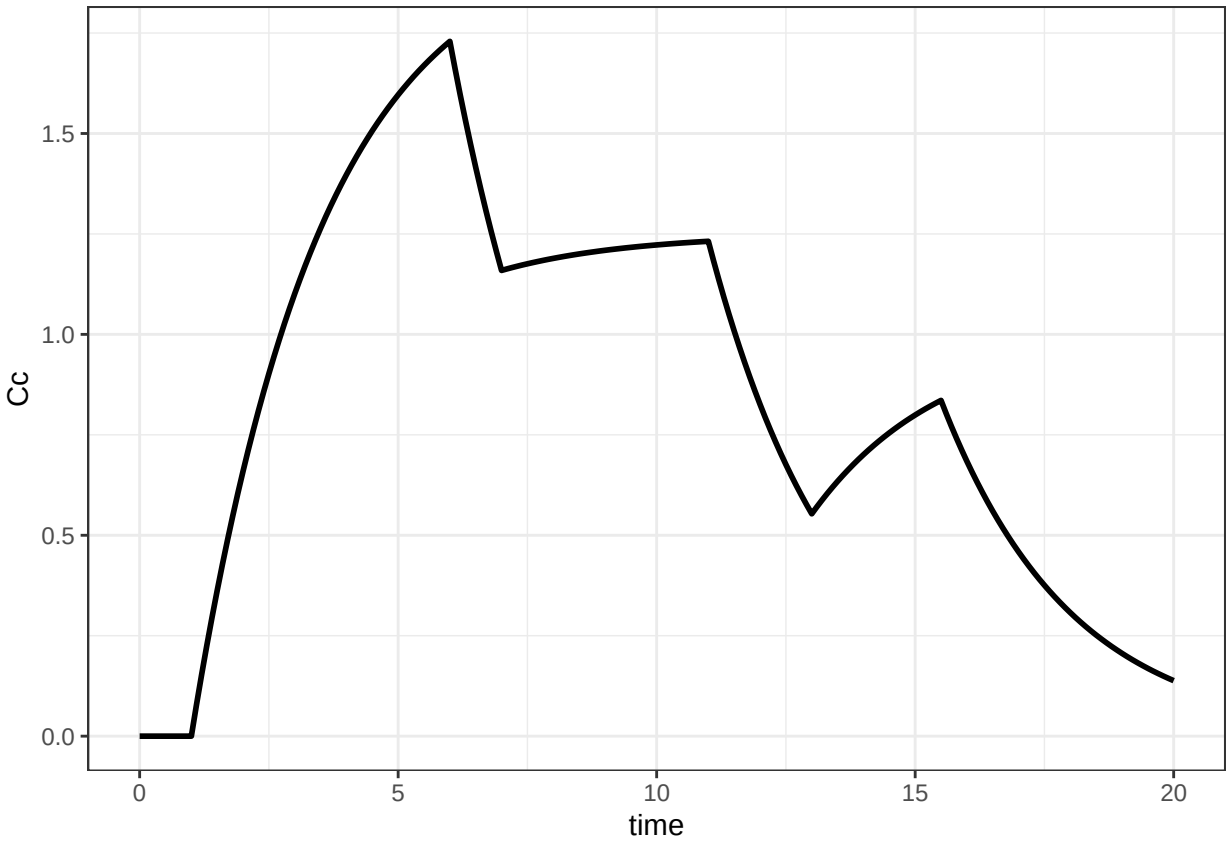
An infusion is defined by adding the field rate (infusion rate) to the list treatment.

```
adm <- list(time=c(1, 7, 13), amount=c(40, 20, 10), rate=8)
res <- simulx(model='model/pk1a.txt', parameter=p, output=Cc, treatment=adm)
print(ggplot(data=res$Cc, aes(x=time, y=Cc)) + geom_line(size=1))
```



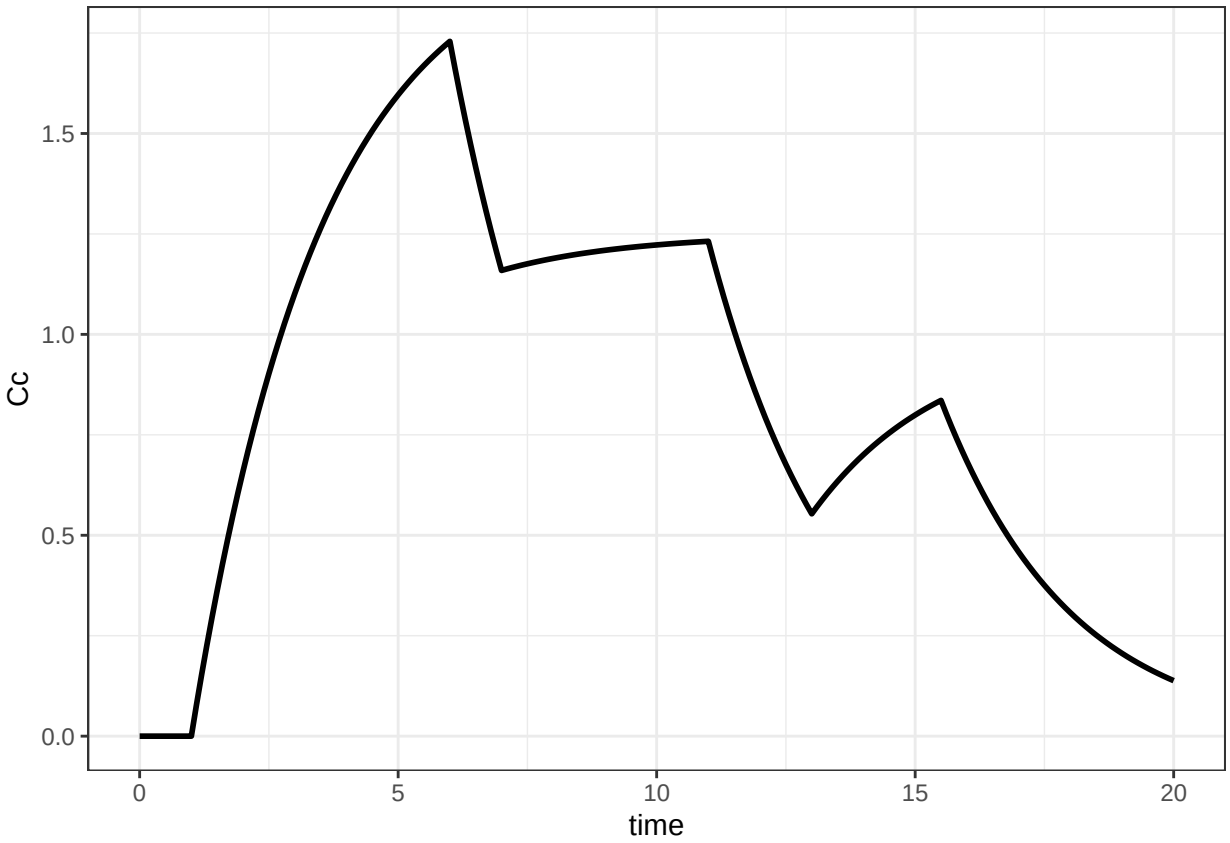
Different rates can be used by defining rate as a vector.

```
adm <- list(time=c(1, 7, 13), amount=c(40, 20, 10), rate=c(8, 5, 4))
res <- simulx(model='model/pk1a.txt', parameter=p, output=Cc, treatment=adm)
print(ggplot(data=res$Cc, aes(x=time, y=Cc)) + geom_line(size=1))
```



Instead of the infusion rate, the duration of the infusion can be equivalently defined as tinf, with tinf=amount/rate

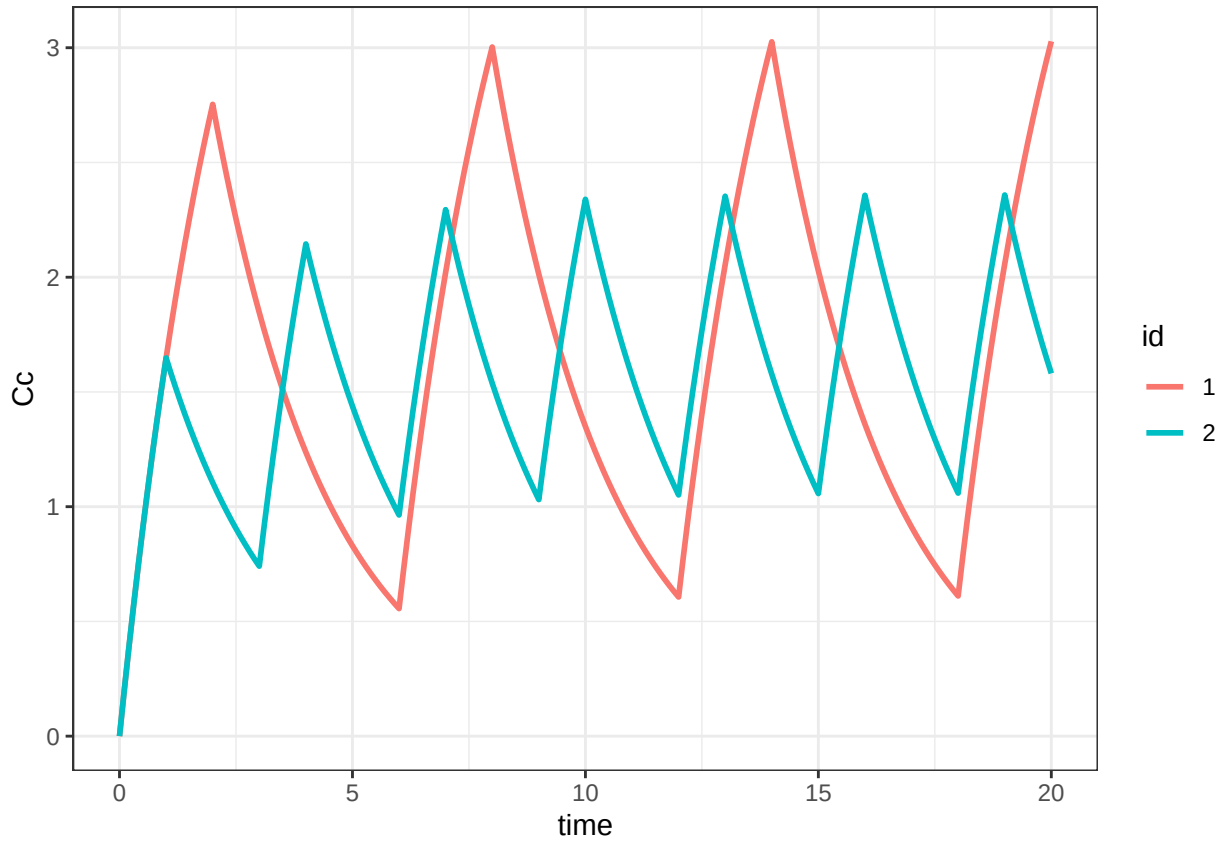
```
adm <- list(time=c(1, 7, 13), amount=c(40, 20, 10), tinf=c(5, 4, 2.5))
res <- simulx(model='model/pk1a.txt', parameter=p, output=Cc, treatment=adm)
print(ggplot(data=res$Cc, aes(x=time, y=Cc)) + geom_line(size=1))
```



Defining several dosage regimens

It is also possible to define several dosage regimens using the input argument group. In this example, each element of group has its own dosage regimen

```
adm1 <- list(time=seq(0,20, by=6), amount=40, tinf=2)
adm2 <- list(time=seq(0,20, by=3), amount=20, tinf=1)
g1   <- list(treatment=adm1)
g2   <- list(treatment=adm2)
res <- simulx(model='model/pk1a.txt', parameter=p, output=Cc, group=list(g1, g2))
print(ggplot(data=res$Cc, aes(x=time, y=Cc, colour=id)) + geom_line(size=1))
```



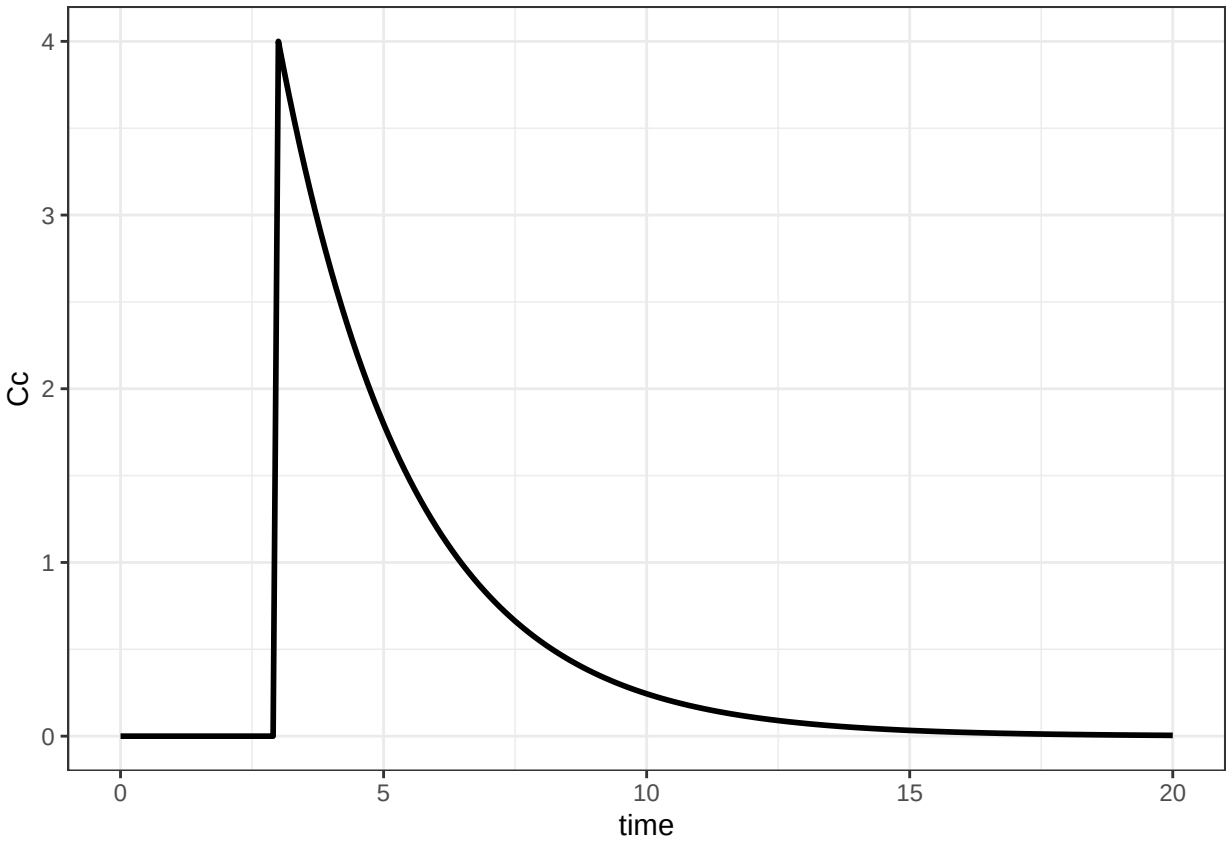
Defining the target compartment

Instead of defining the target compartment in the `PK` block of the Mlxtran code, it is possible to define it in the `R` code.

Here, the Mlxtran `pk1b.txt` code has no block `PK`:

The target compartement (i.e. the central compartment which amount is A_c) is defined as an additional field target of treatment.

```
adm <- list(target="Ac", time=3, amount=40)
res <- simulx(model='model/pk1b.txt', parameter=p, output=Cc, treatment=adm)
print(ggplot(data=res$Cc, aes(x=time, y=Cc)) + geom_line(size=1))
```


Cc
4
3
2
1
0
0
5
10
15
20
time

Simulation of PK models: multiple routes of administration

R script: pk2.R

Mlxtran code: model/pk2a.txt ; model/pk2b.txt ; model/pk3.txt

---

Introduction

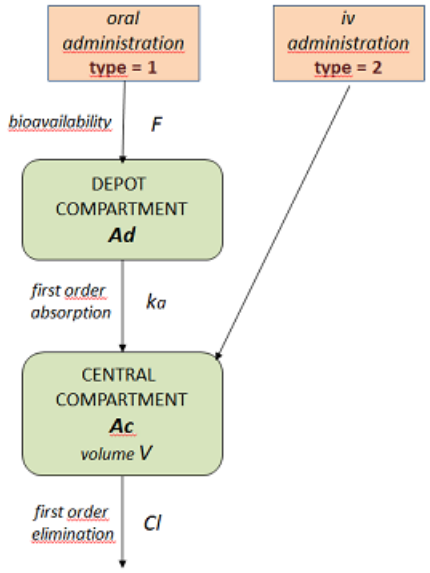
It is possible to define elementary administrations and then define a treatment as a combination of these elementary administrations.

Examples

Example 1

Assume now that we combine oral and iv administrations of the same drug.

We assume here a one compartment model with first-order absorption process from the depot compartment (oral administration) and a linear elimination process from the central compartment. We further assume that only a fraction F (bioavailability) of the drug orally administered is absorbed.



Let A_d and A_c be, respectively, the amounts in the depot compartment (gut) and the central compartment (bloodtsream). Kinetics of A_d and A_c are described by the following system of ODEs

$$\dot{A}_d(t) = -ka\, A_d(t)$$
$$\dot{A}_c(t) = ka\, A_d(t) - k\, A_c(t)$$

The concentration C_c in the central compartment is defined by $C_c(t) = A_c(t)/V$

The target compartment is the depot compartment (A_d) for oral administrations and the central compartment (A_c) for iv administrations. This model is implemented in `pk2a.txt` using a system of ODEs

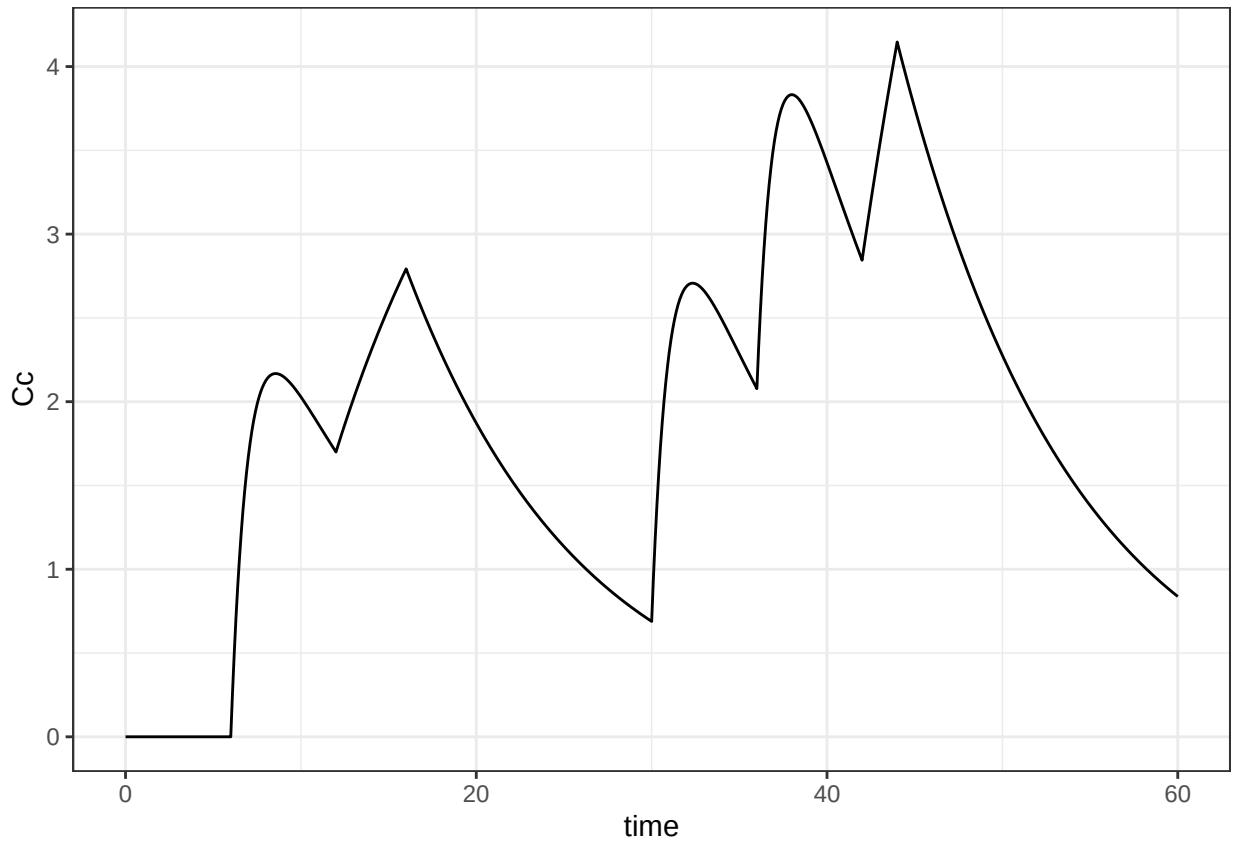
Remark: the same model could be efficiently implemented using PK macros instead of ODEs (indeed, the system here is linear and there is no need to sove ODE systems when analytical solutions are available)

Here, list adm1 defines the oral administrations. Then, an aditional field type=1 is used to link this administration with the PK model defined in the Mlxtran code.

List adm2 defines the iv infusion administrations with type=2.

When each elementary administration has been defined, treatment is defined as a list that includes all the elementary administrations.

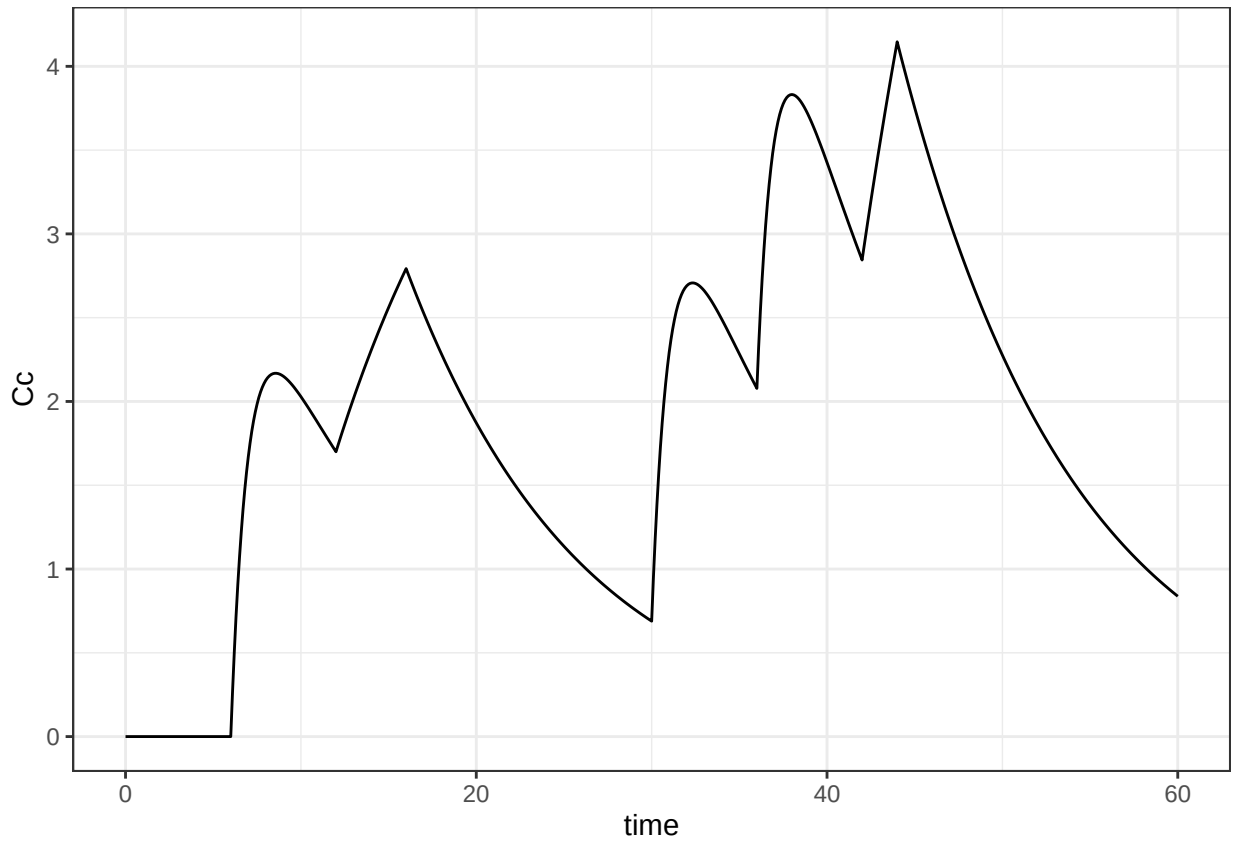
```
adm1 <- list(type   = 1,
             time   = c(6, 30, 36),
             amount = 40)

adm2 <- list(type   = 2,
             time   = c(12,42),
             amount = 20,
             rate   = c(5, 10))

p    <- c(F=0.7, ka=1, V=10, k=0.1)

Cc   <- list(name="Cc", time=seq(0, 60, by=0.1))

res  <- simulx(model     = "model/pk2a.txt",
               parameter = p,
               output    = Cc,
               treatment = list(adm1, adm2))

print(ggplot(data=res$Cc, aes(x=time, y=Cc)) + geom_line())
```



Instead of defining the target compartments (i.e. the target components of the system) in the `PK` block of the Mlxtran code, it is possible to define them in the `R` code. Here, the Mlxtran `pk2b.txt` code has no block `PK`:

The target components are defined as additional fields of adm1 and adm2.

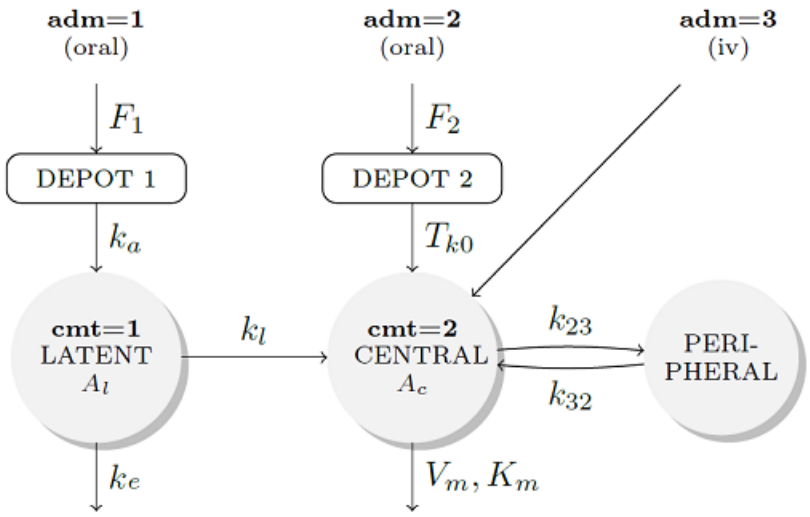
```
adm1 <- list(target = "Ad",
             time   = c(6, 30, 36),
             amount = 40*p[["F"]])

adm2 <- list(target = "Ac",
             time   = c(12,42),
             amount = 20,
             rate   = c(5, 10))

res  <- simulx(model     = "model/pk2b.txt",
               parameter = p,
               output    = Cc,
               treatment = list(adm1, adm2))

print(ggplot(data=res$Cc, aes(x=time, y=Cc)) + geom_line())
```



Example 2

In this example, one type of dose is administered orally (adm=1) and absorbed into a latent compartment following a first-order absorption process, a second type is administered orally (adm=2) and absorbed into the central compartment following a zero-order absorption process, and a third type is directly administered intravenously to the central compartment (adm=3).

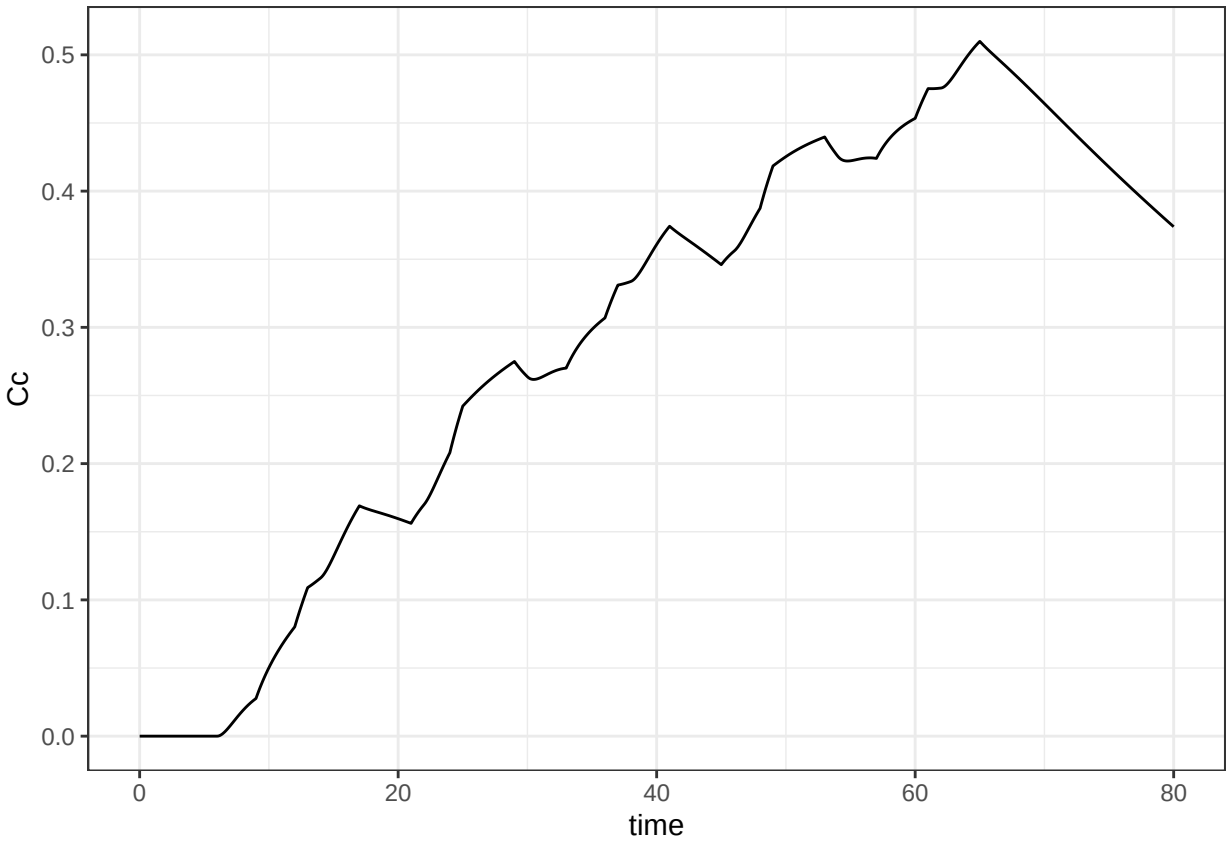
There is linear transfer from the latent to the central compartment. A peripheral compartment is linked to the central compartment. The drug is eliminated by a linear process from the latent compartment and a nonlinear process from the central one. Here, A_l and A_c are the drug amounts in the latent and central compartments.



This model is implemented in `pk3.txt` using PK macros:

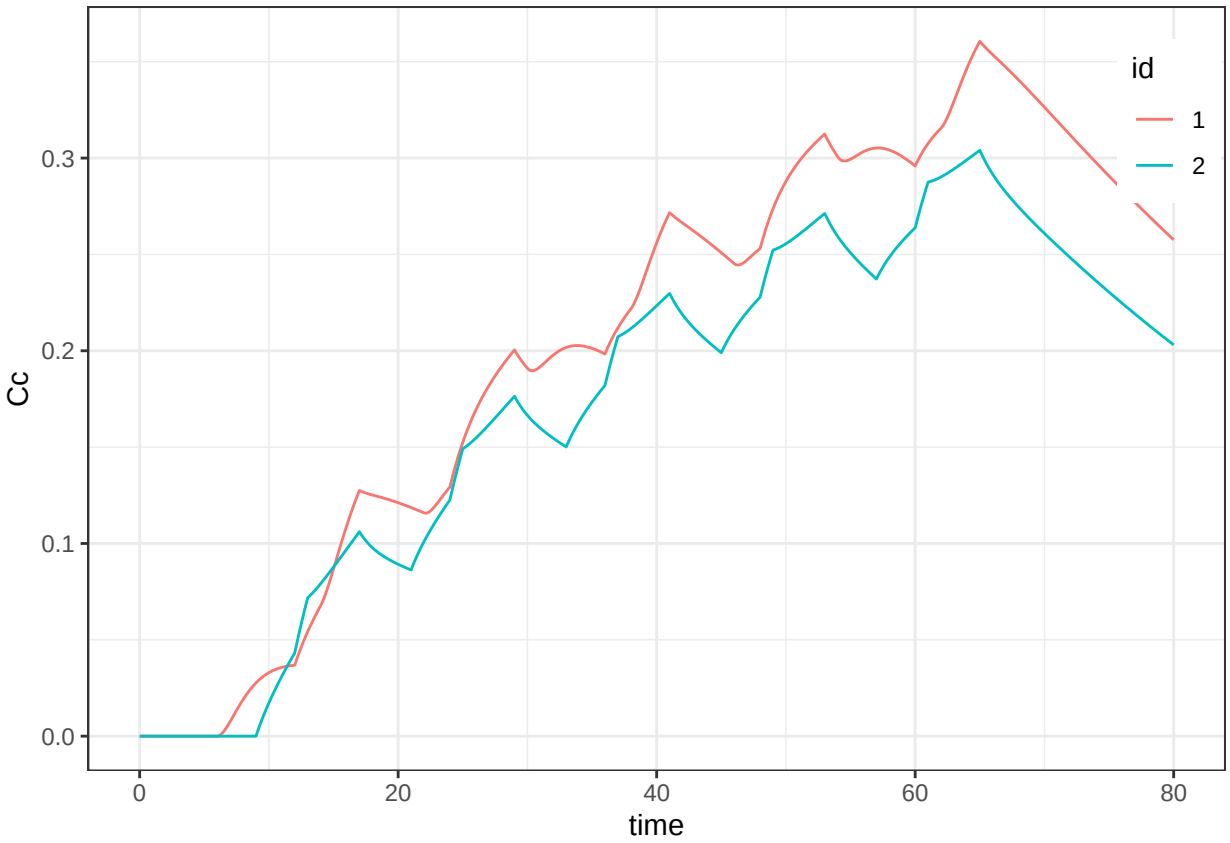
We can then combine these three types of administration by defining three lists adm1, adm2 and adm3.

```
adm1 <- list(type=1, time=seq(6, 66, by=8),  amount=2)
adm2 <- list(type=2, time=seq(9, 57, by=12), amount=1)
adm3 <- list(type=3, time=seq(12,60, by=12), amount=1,rate=0.2)

p <- c(F1=0.5, F2=0.8, ka=0.5, Tk0=4, kl=0.5, k23=0.3,
       k32=0.5, V=10, k=0.2, Vm=0.5, Km=1)

Cc <- list(name = "Cc", time = seq(0,to=80,by=0.1))

res <- simulx( model     = "model/pk3.txt",
               parameter = p,
               output    = Cc,
               treatment = list(adm1, adm2, adm3))

print(ggplot(data=res$Cc, aes(x=time, y=Cc)) + geom_line())
```



Using the same model, we can also define different combinations of administration; In this example, will will consider a combination of adm1 and adm3 and a combination of adm2 and adm3.

```
g1  <- list(treatment=list(adm1, adm3))
g2  <- list(treatment=list(adm2, adm3))

res <- simulx( model     = "model/pk3.txt",
               parameter = p,
               output    = Cc,
               group     = list(g1, g2))
print(ggplot(data=res$Cc) + geom_line(aes(x=time, y=Cc, colour=id)) +
  theme(legend.position=c(.95, .85)))
```


id
1
2
Cc
0.0
0.1
0.2
0.3
0
20
40
60
80
time

Simulation of PK models: weight-based dosing

R script: pk3.R

Introduction

The dose amount can easily be defined a the function of some individual covariate, such as the weight.

In the two following examples, the dose amount is defined per kilo (2 mg per kilo) and the weight is used as the fraction which is absorbed.

Examples

Defining the weight in the Mlxtran model file

We use the function `pkmodel` in this example with the input argument `p`.

The weight w is defined as a covariate which is randomly sampled from the model defined in section `[COVARIATE]`.

We use a PK model for iv bolus administration, assuming that the volume of the central compartment is equal to 1. The concentration at $t = 0$ is therefore the dose amount administrated at $t = 0$, i.e. twice the individual weight.

```
pkmodel1 <- inlineModel("
[LONGITUDINAL]
input = {V, k, w}
EQUATION:
Cc = pkmodel(V, k, p=w)

;----------------------------------------------
[COVARIATE]
input = {w_pop, omega_w}
DEFINITION:
w = {distribution = normal, mean = w_pop, sd = omega_w}
")

p   <- c(w_pop=70, omega_w=12, k=0.1, V=10)
dosePerKg <- 2
trtPerKg <- list(amt=dosePerKg, time=c(0, 12, 24))
Cc  <- list(name='Cc', time=seq(0, 36, by=1))
w   <- list(name='w')

res <- simulx(model     = pkmodel1,
              parameter = p,
              treatment = trtPerKg,
              output    = list(Cc,w),
              group     = list(size=3, level="covariate"),
              settings  = list(seed=123))

print(res$parameter)
```

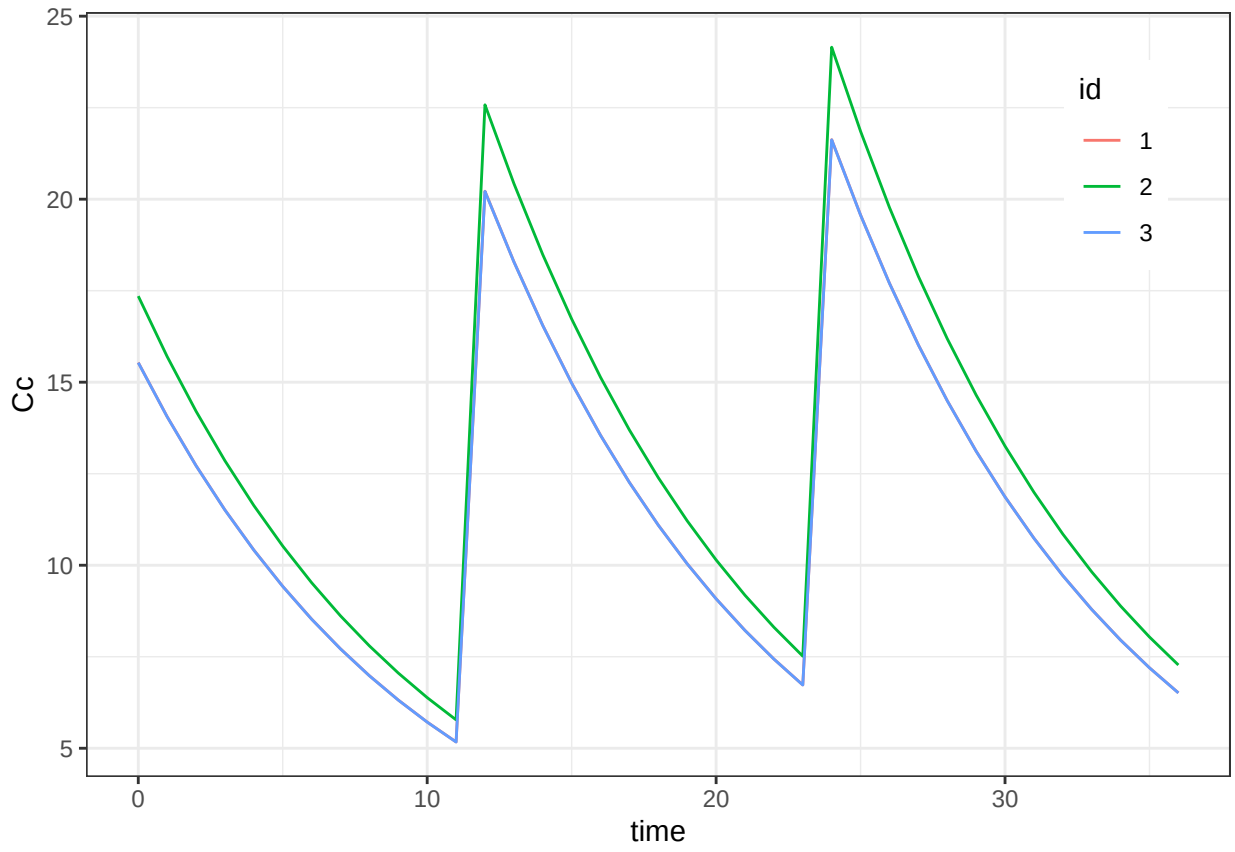
```
##   id        w
## 1  1 77.68822
```

```
## 2  2 86.75795
## 3  3 77.66391
```

```
print(res$Cc[res$Cc$time==0,])
```

```
##    id time       Cc
## 1   1    0 15.53764
## 38  2    0 17.35159
## 75  3    0 15.53278
```

```
print(ggplot(data=res$Cc) + geom_line(aes(x=time, y=Cc, colour=id)) +
  theme(legend.position=c(.9, .8)))
```



Defining the individual weights in the R script

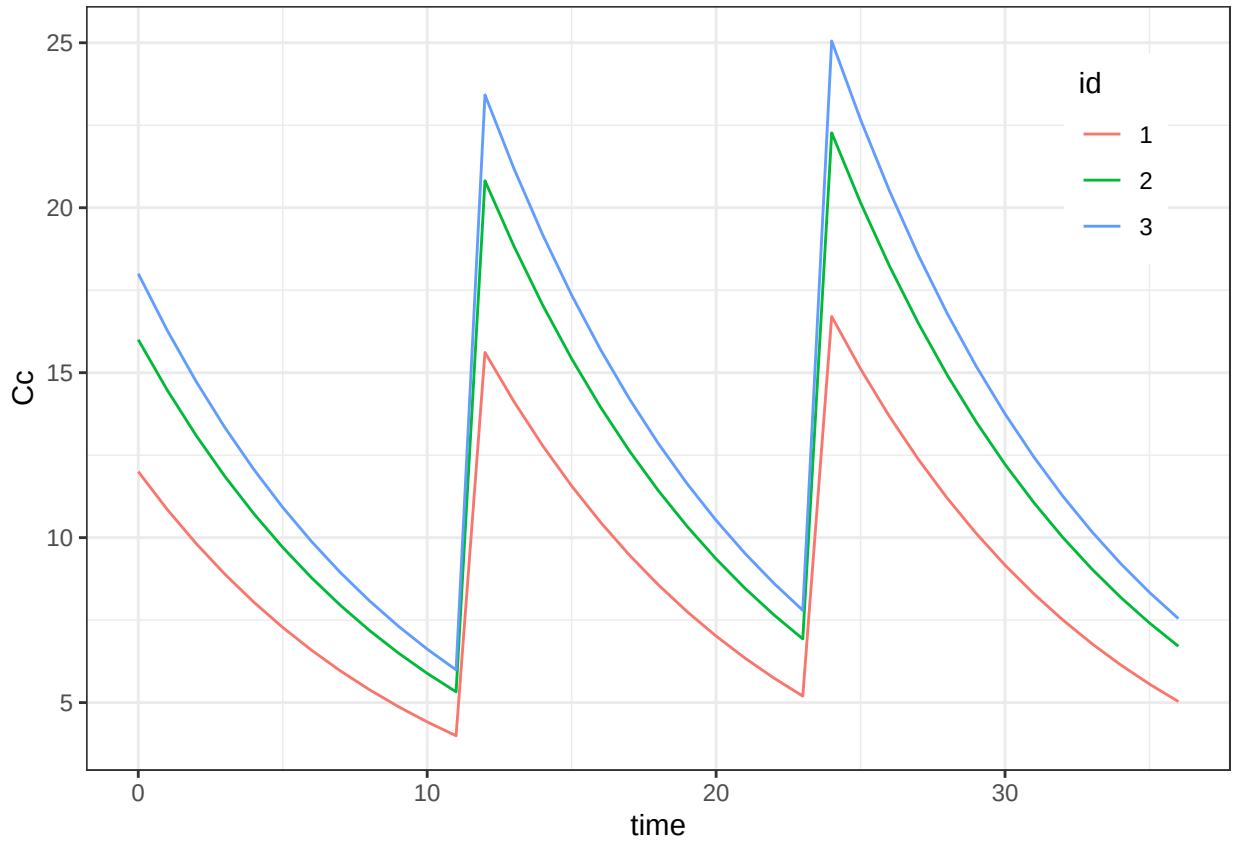
Of course, it is also possible to define the weights in the R script as a data frame and use it as an input argument of the model:

```
pkmodel2 <- inlineModel("
[LONGITUDINAL]
input = {V, k, w}
EQUATION:
Cc = pkmodel(V, k, p=w)
")

weight <- data.frame(id=(1:3), w=c(60,80,90))

res <- simulx(model     = pkmodel2,
```

```
              parameter = list(p, weight),
              treatment = trtPerKg,
              output    = Cc)

print(res$Cc[res$Cc$time==0,])
```

```
##    id time Cc
## 1   1    0 12
## 38  2    0 16
## 75  3    0 18
```

```
print(ggplot(data=res$Cc) + geom_line(aes(x=time, y=Cc, colour=id)) +
  theme(legend.position=c(.9, .8)))
```



Defining the individual dose regimens in the R script

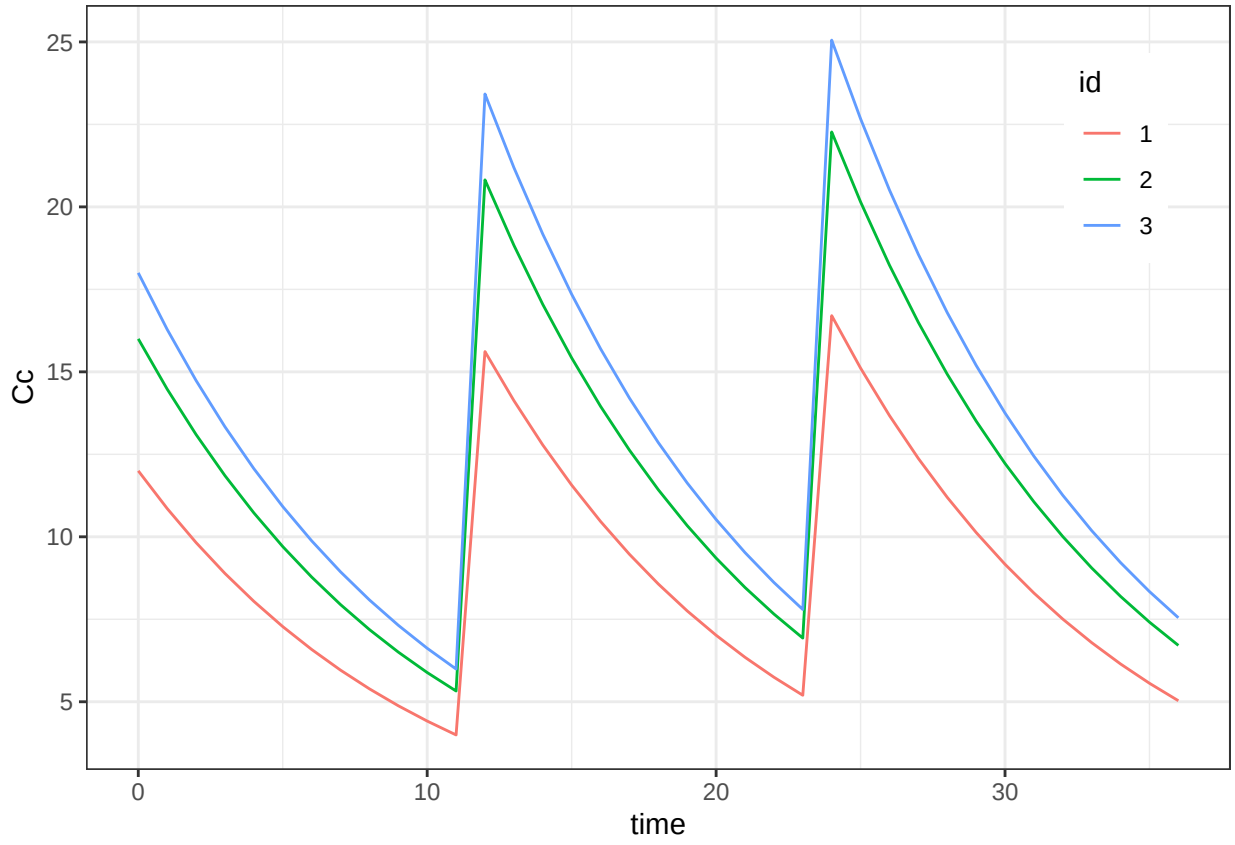
The doses can also be defined in the R script. In such case, the treatment should be defined a a data frame:

```
pkmodel3 <- inlineModel("
[LONGITUDINAL]
input = {V, k}
EQUATION:
Cc = pkmodel(V, k)
")

w <- c(60,80,90)
N <- length(w)
```

```
times <- c(0, 12, 24)
nbTimes <- length(times)
trt <- data.frame(id     = rep(1:N,each=nbTimes),
                  time   = rep(times,N),
                  amount = rep(dosePerKg*w,each=nbTimes))

res <- simulx(model     = pkmodel3,
              parameter = p,
              treatment = trt,
              output    = Cc)

print(res$Cc[res$Cc$time==0,])
```

```
##    id time Cc
## 1   1    0 12
## 38  2    0 16
## 75  3    0 18
```

```
print(ggplot(data=res$Cc) + geom_line(aes(x=time, y=Cc, colour=id)) +
  theme(legend.position=c(.9, .8)))
```



Using an ODE based model with *depot*

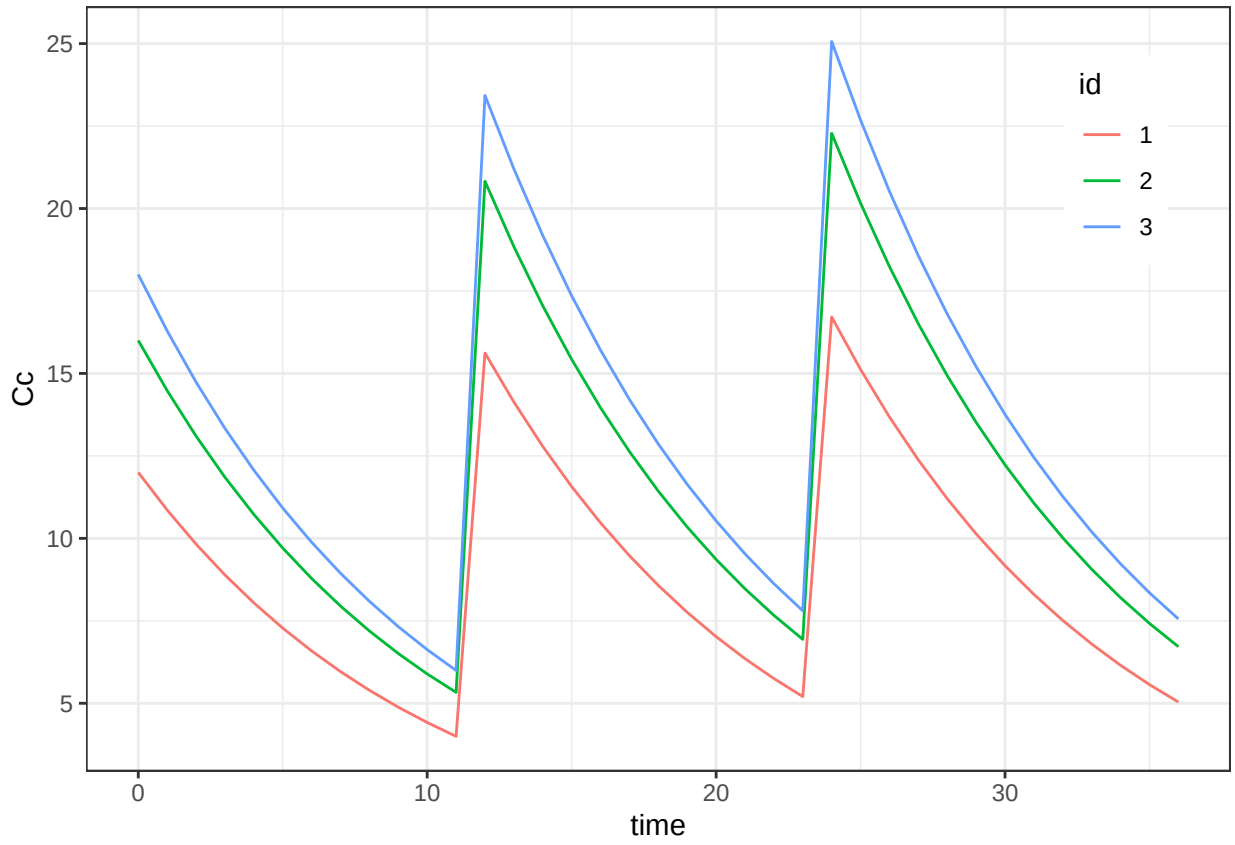
The same PK model is defined here by an ODE with the PK macro `depot` and the input argument `p`.

The individual weights w are defined as an input of the model.

```
pkmodel4 <- inlineModel("
[LONGITUDINAL]
input = {V, k, w}

PK:
depot(target=Ac, p=w)
EQUATION:
ddt_Ac = -k*Ac
Cc = Ac/V
")

res <- simulx(model     = pkmodel4,
              parameter = list(p, weight),
              treatment = trtPerKg,
              output    = list(Cc))

print(res$Cc[res$Cc$time==0,])
```

```
##    id time Cc
## 1   1    0 12
## 38  2    0 16
## 75  3    0 18
```

```
print(ggplot(data=res$Cc) + geom_line(aes(x=time, y=Cc, colour=id)) +
  theme(legend.position=c(.9, .8)))
```



Simulation of nonadherence

R script: adherence.R

Introduction

The dosage regimen is the modality of drug administration that is chosen to reach some therapeutic objective. Adherence is defined as the extent to which patients are able to follow the recommendations for prescribed treatments. On the other hand, nonadherence means that the prescribed dosage regimen was not respected by the patient.

We will consider different types of nonadherence:

- A prescribed dose is not taken at all,
- patients may use more or less than the prescribed amount,
- patients may use their medication at the wrong time.

Then, several options are available for simulating nonadherence using `simulx`.

Using random individual dosage regimens

We will consider a simple example using a basic PK model. Our objective is to predict the exposure (i.e. to compute the predicted plasmatic concentration) for 3 patients.

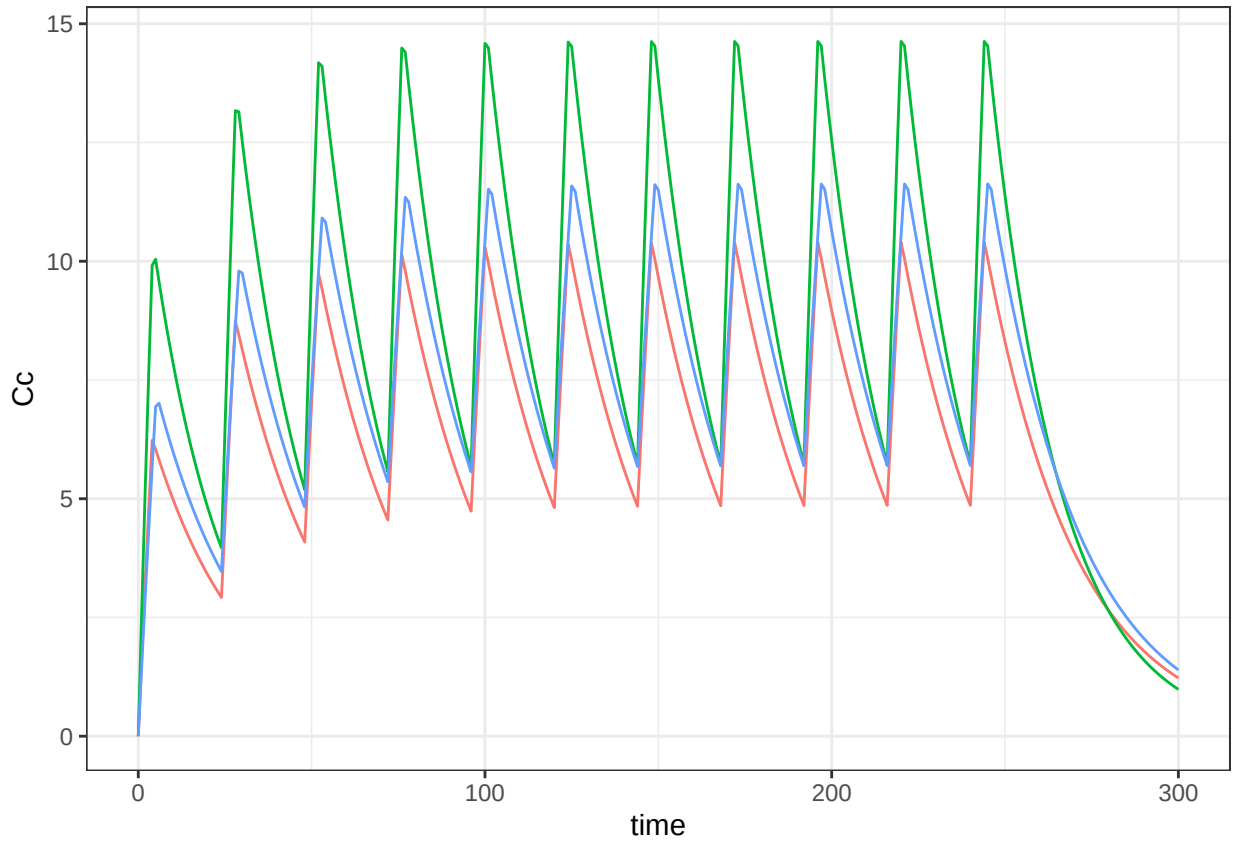
```
N <- 3

myPKmodel1 <- inlineModel("
[LONGITUDINAL]
input={Tk0,V,Cl}

EQUATION:
Cc = pkmodel(Tk0, V, Cl)

[INDIVIDUAL]
input={Tk0_pop,V_pop,Cl_pop,omega_Tk0,omega_V,omega_Cl}

DEFINITION:
Tk0   = {distribution=lognormal,   prediction=Tk0_pop,  sd=omega_Tk0}
V     = {distribution=lognormal,   prediction=V_pop,    sd=omega_V}
Cl    = {distribution=lognormal,   prediction=Cl_pop,   sd=omega_Cl}
")

pk.param   <- c(
  Tk0_pop   = 5,    omega_Tk0   = 0.2,
  V_pop     = 10,   omega_V     = 0.5,
  Cl_pop    = 0.5,  omega_Cl    = 0.2
)

out <- list(name="Cc", time=0:300)
```

The precribed dosage regimen consists in repeated oral administrations of 100mg every 24 hours:

```
tdose <- seq(0,240,by=24)
amtdose <- 100
adm1 <- list(amount=amtdose, time=tdose)
```

We can simulate 3 patients and compute there predicted concentrations, assuming a full adherence to this dosage regimen:

```
res1 <- simulx(model     = myPKmodel1,
               parameter = pk.param,
               treatment = adm1,
               output    = out,
               group     = list(size=N, level="individual"),
               settings  = list(seed=12345))

pl1 <- ggplot(res1$Cc) + geom_line(aes(time,Cc,colour=id)) + theme(legend.position = "none")
print(pl1)
```



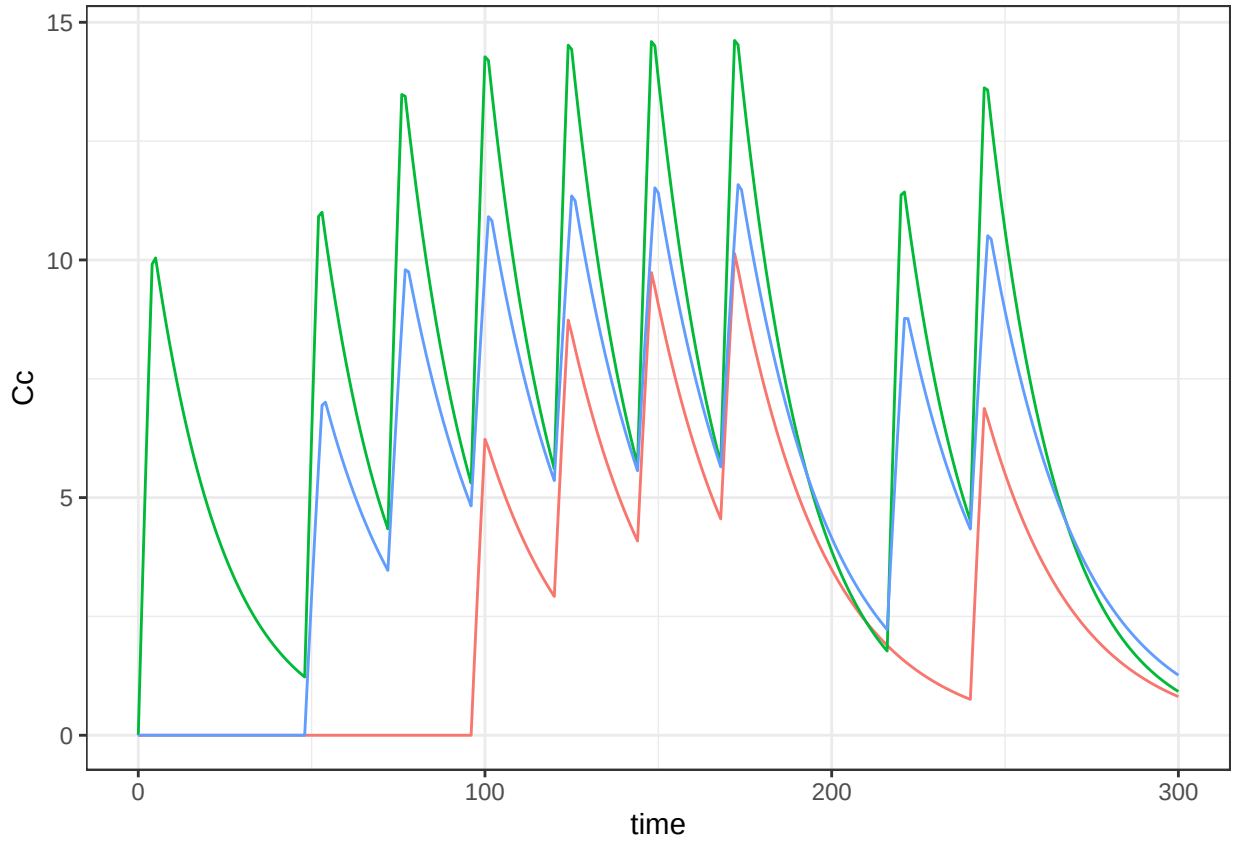
Individual dosage regimens could be equivalently defined instead of a global one:

```
#---- individual dosage regimens  ----------------------
ndose <- length(tdose)
adm2 <- data.frame(id = rep(1:N,each=ndose),
                   amount=amtdose,
                   time=rep(tdose,N))
res2 <- simulx(model     = myPKmodel1,
               parameter = pk.param,
               treatment = adm2,
```

```
                output     = out,
                settings   = list(seed=12345))
```

Removing a row from the data frame adm2 means that the prescribed dose was not taken. If we assume, for instance, that only 70% of the prescribed doses are taken, we can randomly remove some rows from adm2:

```
#------  non adherence <=> remove treatment lines  --------------------
adherence.rate <- 0.7
adm3 <- adm2[runif(N*ndose)<adherence.rate,]
res3 <- simulx(model      = myPKmodel1,
               parameter  = pk.param,
               treatment  = adm3,
               output     = out,
               settings   = list(seed=12345))

pl3 <- ggplot(res3$Cc) + geom_line(aes(time,Cc,colour=id)) + theme(legend.position = "none")
print(pl3)
```



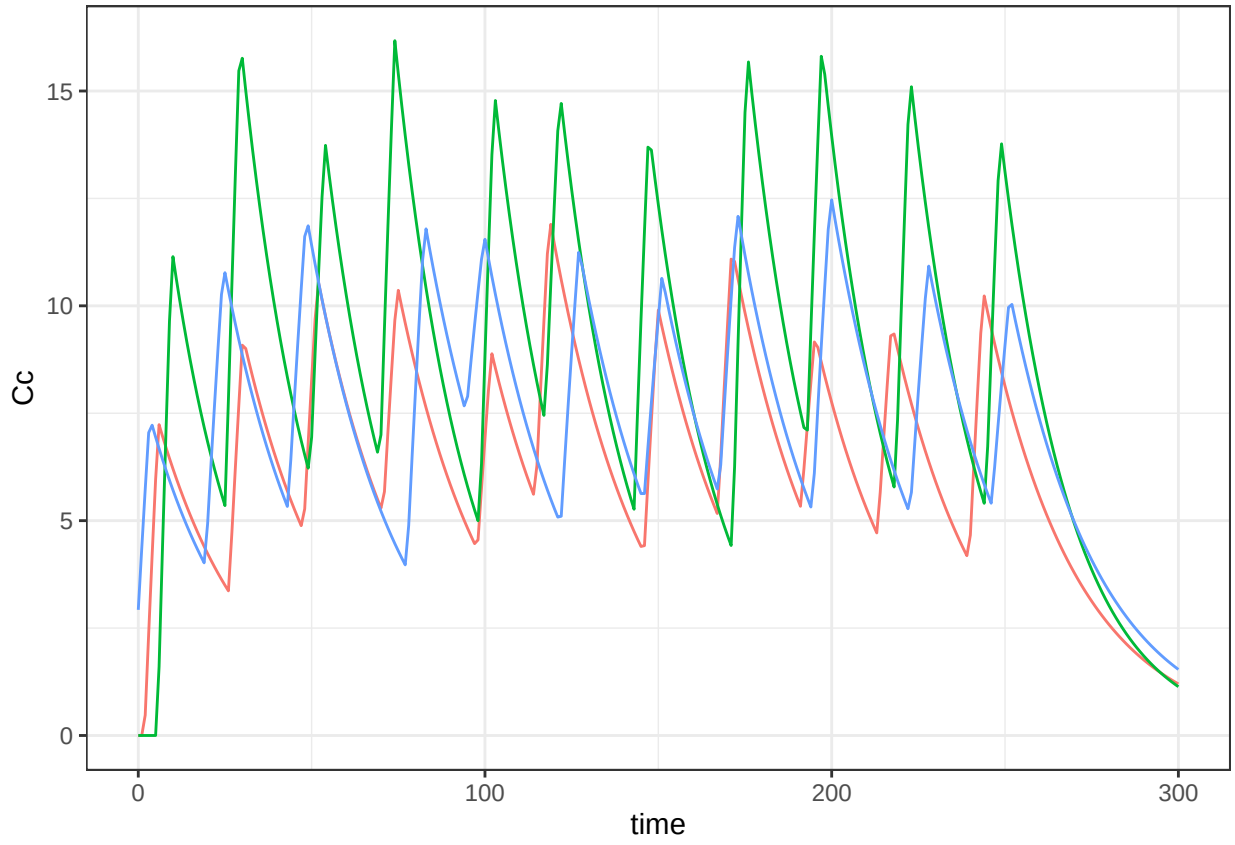
If we now assume that the prescribed amounts and times are not exactly respected, we can consider the amount and time as random variables. Normal distributions are used in the following example:

```
#------- variability on dosing times and amounts -----------------
sd.time <- 3
sd.amount <- 10
adm4 <- adm2
adm4$time <- adm4$time + rnorm(N*ndose)*sd.time
adm4$amount <- adm4$amount + rnorm(N*ndose)*sd.amount
```

```
head(adm4)
```

```
##   id    amount       time
## 1  1 116.32446   1.756586
## 2  1 102.54271  26.128398
## 3  1 104.91188  47.672090
## 4  1  96.75913  70.639508
## 5  1  83.37950  97.817662
## 6  1 117.67734 114.546132
```

```
res4 <- simulx(model     = myPKmodel1,
               parameter = pk.param,
               treatment = adm4,
               output    = out,
               settings  = list(seed=12345))

pl4 <- ggplot(res4$Cc) + geom_line(aes(time,Cc,colour=id)) + theme(legend.position = "none")
print(pl4)
```



Defining nonadherence as a regression variable

We can equivalently define in the structural model the fraction of dose which is effectively administrated

```
myPKmodel2 <- inlineModel("
[LONGITUDINAL]
input={Tk0,V,Cl,fd}
fd = {use=regressor}
```

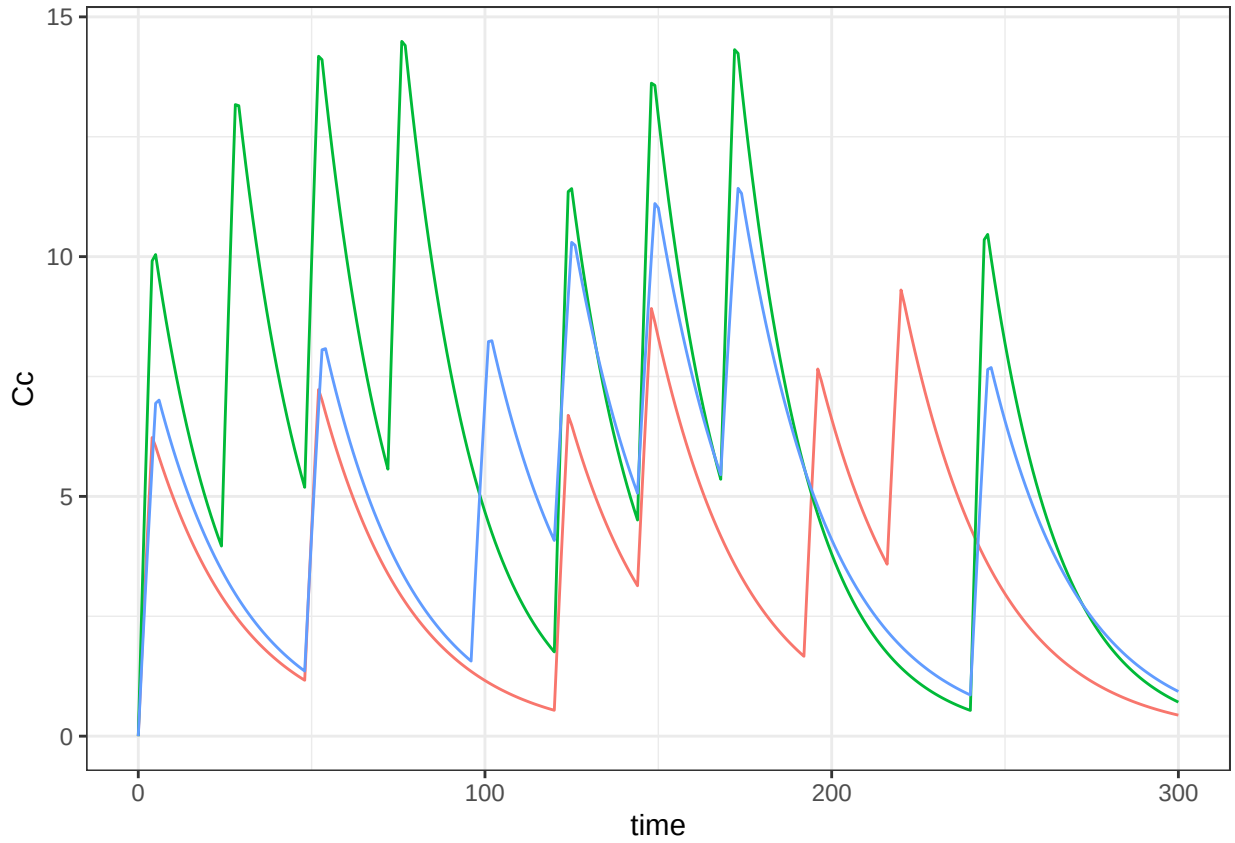
```
EQUATION:
Cc = pkmodel(Tk0, V, Cl, p=fd)

[INDIVIDUAL]
input={Tk0_pop,V_pop,Cl_pop,omega_Tk0,omega_V,omega_Cl}

DEFINITION:
Tk0   = {distribution=lognormal,   prediction=Tk0_pop,  sd=omega_Tk0}
V     = {distribution=lognormal,   prediction=V_pop,    sd=omega_V}
Cl    = {distribution=lognormal,   prediction=Cl_pop,   sd=omega_Cl}
")
```

Here, $fd = 1$ means that the prescribed amount has been taken while $fd = 0$ means that the dose was not taken at all:

```
set.seed(123)
ndose <- length(tdose)
fd <- adm2[-2]
fd$fd=as.numeric(runif(N*ndose)<adherence.rate)
head(fd)
```

```
##   id time fd
## 1  1    0  1
## 2  1   24  0
## 3  1   48  1
## 4  1   72  0
## 5  1   96  0
## 6  1  120  1
```

```
res5 <- simulx(model     = myPKmodel2,
               parameter = pk.param,
               treatment = adm1,
               output    = out,
               regressor = fd,
               settings  = list(seed=12345))

pl5 <- ggplot(res5$Cc) + geom_line(aes(time,Cc,colour=id)) + theme(legend.position = "none")
print(pl5)
```

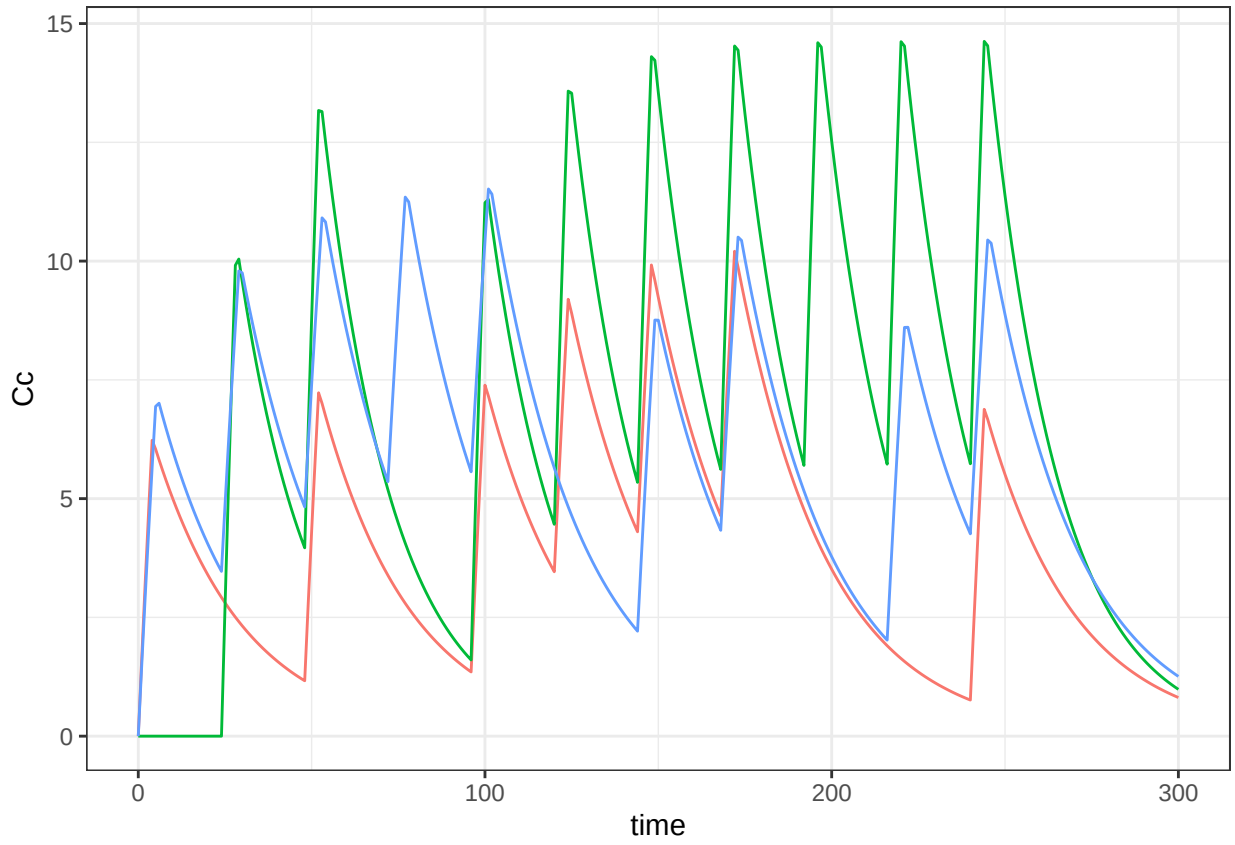


On the other hand, $fd \neq 1$ means that the prescribed amount was not respected.

```
set.seed(123)
fd$fd=adm2$amount*rnorm(ndose,1,0.1)
head(fd)
```

```
##   id time        fd
## 1  1    0  94.39524
## 2  1   24  97.69823
## 3  1   48 115.58708
## 4  1   72 100.70508
## 5  1   96 101.29288
## 6  1  120 117.15065
```

```
fd <- data.frame(id = rep(1:N,each=ndose),
                 time=rep(tdose,N),
                 fd=as.numeric(runif(N*ndose)<adherence.rate))
res6 <- simulx(model     = myPKmodel2,
               parameter = pk.param,
               treatment = adm1,
               output    = out,
               regressor = fd,
               settings  = list(seed=12345))

pl6 <- ggplot(res6$Cc) + geom_line(aes(time,Cc,colour=id)) + theme(legend.position = "none")
print(pl6)
```



Defining nonadherence as a sequence of individual parameters

The fraction of dose which is effectively administrated can be alternatively defined as a sequence of individual parameters in the model itself. If we assume that this fraction may vary from a dose to another one for a given patient, an occasion should be defined for each dose and inter occasion variability (IOV) should be introduced for this parameter.

In the following example, a logitnormal distribution is used in order to constraint this fraction to be between 0 and 1 (any other distribution for positive random variables could be used):

```
myPKmodel3a <- inlineModel("
[LONGITUDINAL]
input={Tk0,V,Cl,fd}

EQUATION:

Cc = pkmodel(Tk0, V, Cl, p=fd)

[INDIVIDUAL]
input={Tk0_pop,V_pop,Cl_pop,fd_pop,omega_Tk0,omega_V,omega_Cl,gamma_fd}

DEFINITION:
Tk0   = {distribution=lognormal,   prediction=Tk0_pop,  sd=omega_Tk0}
V     = {distribution=lognormal,   prediction=V_pop,    sd=omega_V}
Cl    = {distribution=lognormal,   prediction=Cl_pop,   sd=omega_Cl}
fd    = {distribution=logitnormal,      prediction=fd_pop, varlevel={id, id*occ}, sd={0,gamma_fd}}
")
```

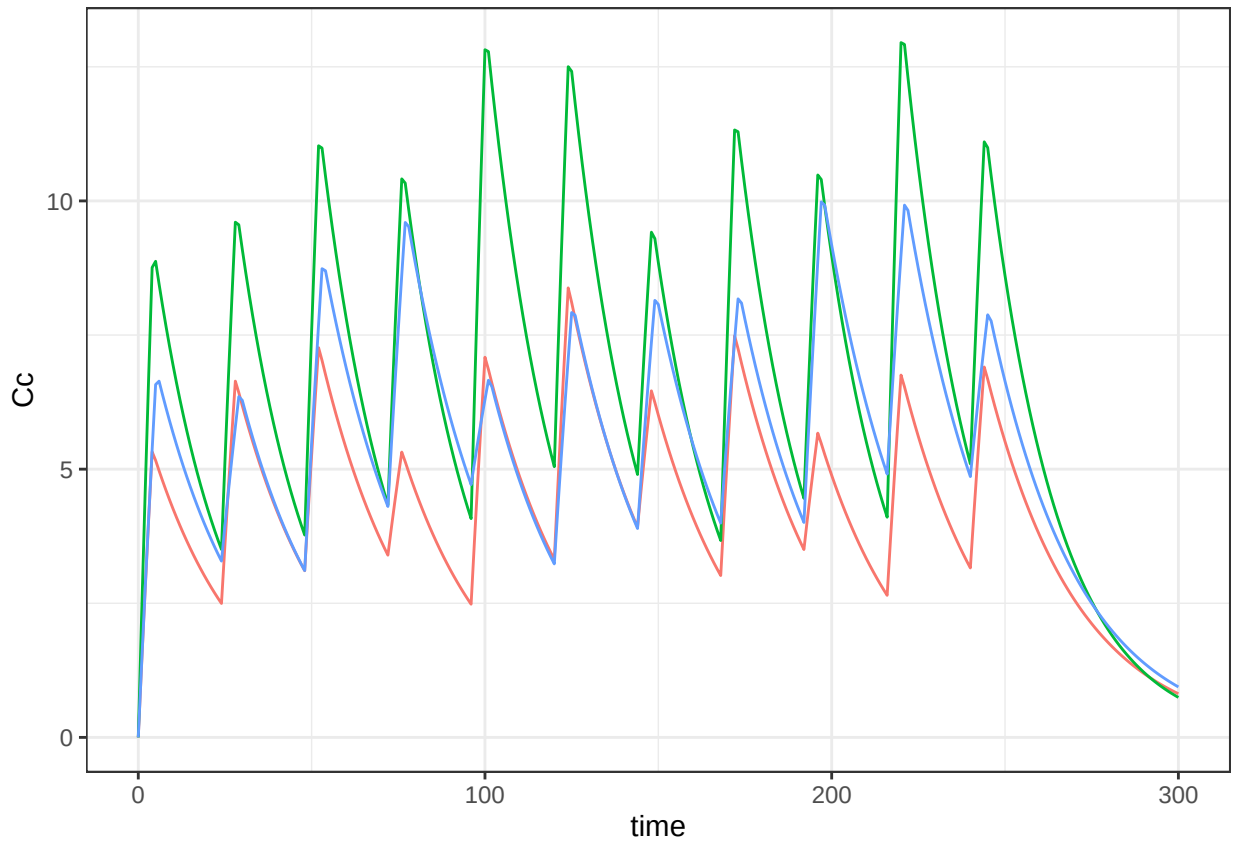
Setting $fd_{\text{pop}} = 1$ and $\gamma_{fd} = 0$ means that all the prescribed doses are correctly taken.

In the following example, this fraction is randomly distributed around 0.8:

```
occ  <- list(time=tdose, name="occ")
fd.param <- c(fd_pop=0.8, gamma_fd=1)
res7 <- simulx(model     = myPKmodel3a,
               parameter = list(pk.param, fd.param),
               treatment = adm1,
               varlevel  = occ,
               output    = list(out, list(name="fd")),
               group     = list(size=N, level="individual"),
               settings  = list(seed=12345))

head(res7$parameter.iov)
```

```
##   id time occ        fd
## 1  1    0   1 0.8547146
## 2  1   24   2 0.7223659
## 3  1   48   3 0.7384445
## 4  1   72   4 0.3859724
## 5  1   96   5 0.7956514
## 6  1  120   6 0.8886648
```

```
pl7 <- ggplot(res7$Cc) + geom_line(aes(time,Cc,colour=id)) + theme(legend.position = "none")
print(pl7)
```



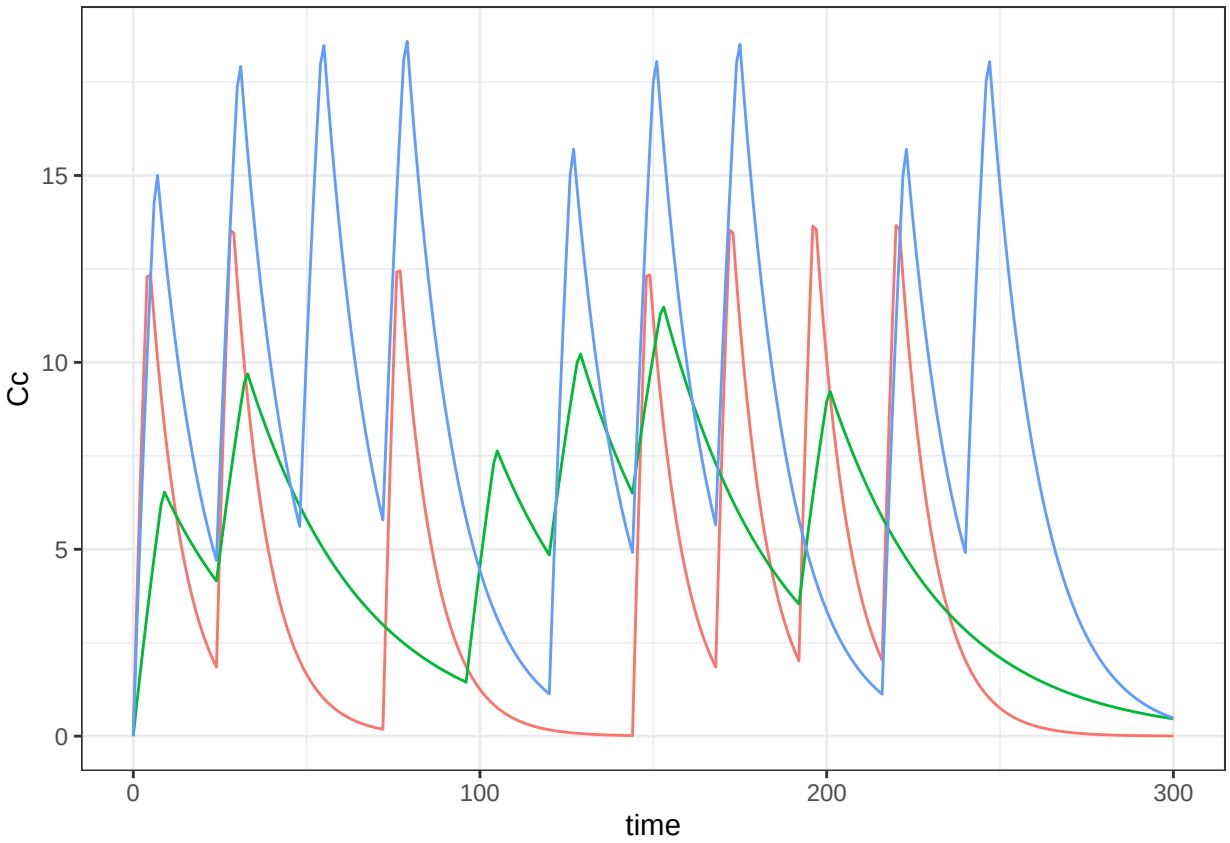
Considering that some doses are not taken at all requires to convert a continuous random variable into a

Bernoulli one:

```
myPKmodel3b <- inlineModel("
[LONGITUDINAL]
input={Tk0,V,Cl,z,zlim}

EQUATION:
if z<zlim
  p = 1
else
  p =0
end
Cc = pkmodel(Tk0, V, Cl, p)

[INDIVIDUAL]
input={Tk0_pop,V_pop,Cl_pop,omega_Tk0,omega_V,omega_Cl}

DEFINITION:
Tk0   = {distribution=lognormal,   prediction=Tk0_pop,  sd=omega_Tk0}
V     = {distribution=lognormal,   prediction=V_pop,    sd=omega_V}
Cl    = {distribution=lognormal,   prediction=Cl_pop,   sd=omega_Cl}
z     = {distribution=normal,      prediction=0, varlevel={id, id*occ}, sd={0,1}}
")

adh.param <- c(zlim = qnorm(adherence.rate))
out.p <- list(name=c("p"), time=occ$time)
res8 <- simulx(model     = myPKmodel3b,
               parameter = list(pk.param, adh.param),
               treatment = adm1,
               varlevel  = occ,
               output    = list(out, out.p),
               group     = list(size=N, level="individual"),
               settings  = list(seed=1234))

head(res8$p)
```

```
##   id time p
## 1  1    0 1
## 2  1   24 1
## 3  1   48 0
## 4  1   72 1
## 5  1   96 0
## 6  1  120 0
```

```
pl8 <- ggplot(res8$Cc) + geom_line(aes(time,Cc,colour=id)) + theme(legend.position = "none")
print(pl8)
```


Cc
15
10
5
0
0
100
200
300
time

Filling, emptying and resetting compartments

R script: reset.R

Introduction

Macros `depot` and `empty` can be used for respectively fill and empty compartments defined as components of an ODE system. On the other hand, `reset` is used for resetting components of the system to their initial values.

These macros can be combined in a same model.

Filling compartments

Filling a single compartment

Consider a one compartment model for oral administration:

```
pk.model1 <- inlineModel("
[LONGITUDINAL]
input = {ka, k}

PK:
depot(target=Ad)

EQUATION:
ddt_Ad = -ka*Ad
ddt_Ac =  ka*Ad - k*Ac
")
```

Here, we are interested in the amount in the central compartment

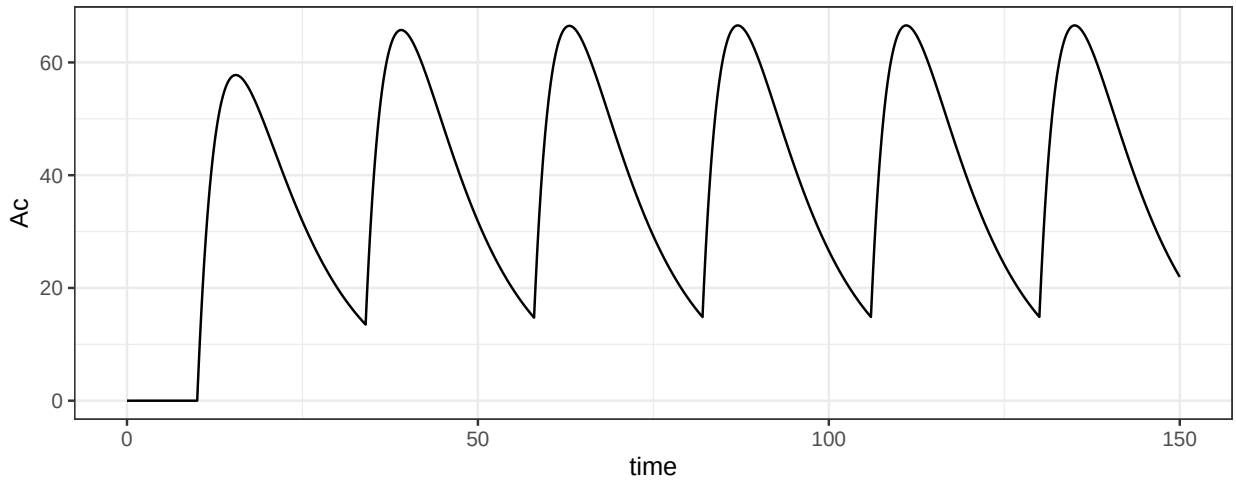
```
p <- c(ka=0.3,  k=0.1)
out <- list(name= "Ac", time = seq(0,150,0.1))
```

By default, the target is Ad, which means that all the doses go to the depot compartment

```
trt <- list(amt=100, time=seq(10,136,by=24))
```

Let us compute and plot the amount in the central compartment defined by this model and this dosing regimen:

```
r <- simulx(model=pk.model1, parameter = p, treatment = trt, output = out)
ggplot(r$Ac, aes(time,Ac)) + geom_line()
```



Filling multiple compartments

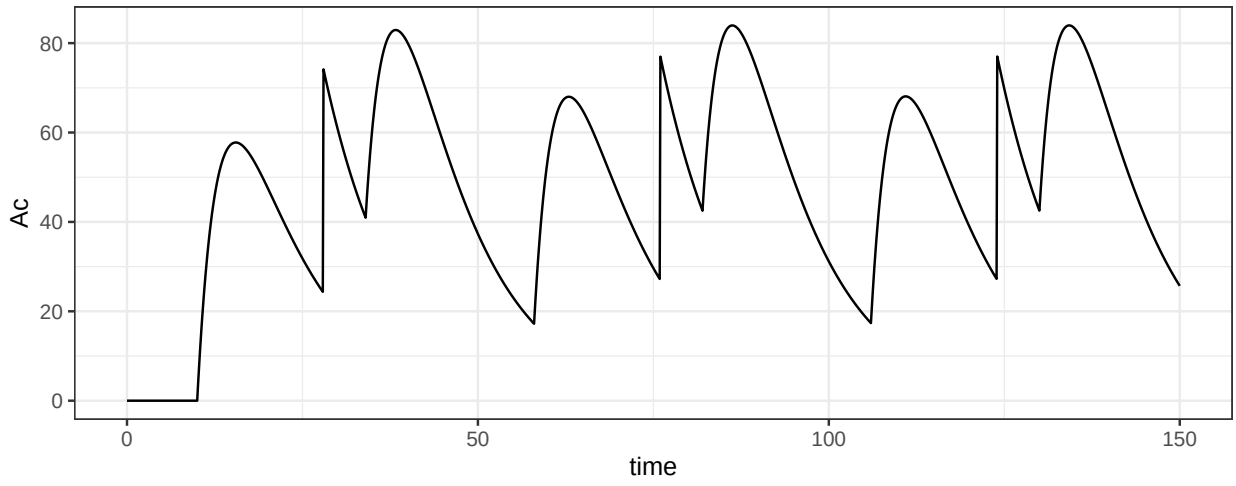
We can now consider both oral (target=Ad) and iv (target=Ac) administrations for this PK model:

```
pk.model2 <- inlineModel("
[LONGITUDINAL]
input = {ka, k}

PK:
depot(target=Ad,  type=1)
depot(target=Ac,  type=2)

EQUATION:
ddt_Ad = -ka*Ad
ddt_Ac =  ka*Ad - k*Ac
")
```

We can then define and combine oral (type=1) and iv (type=2) of administrations:

```
trt1 <- list(amt=100, time=seq(10,136,by=24), type=1) ## put doses in the depot compartment
trt2 <- list(amt=50,  time=seq(28,136,by=48), type=2) ## put doses in the central compartment
trt <- list(trt1, trt2)
```

```
r <- simulx(model=pk.model2, parameter = p, treatment = trt, output = out)
ggplot(r$Ac, aes(time,Ac)) + geom_line()
```



Emptying compartments

Emptying a single compartment

Any component of the system can be set to 0 at any time. In this example, Ac can be set to 0, which means that the central compartment cen be empty:

```
pk.model1 <- inlineModel("
[LONGITUDINAL]
input = {ka, k}

PK:
depot(target=Ad, type=1)
empty(target=Ac, type=2)

EQUATION:
ddt_Ad = -ka*Ad
ddt_Ac =  ka*Ad - k*Ac
")

p <- c(ka=0.3,  k=0.1)
out <- list(name= "Ac", time = seq(0,150,0.1))
```

Here, trt1 defines the doses which are administered

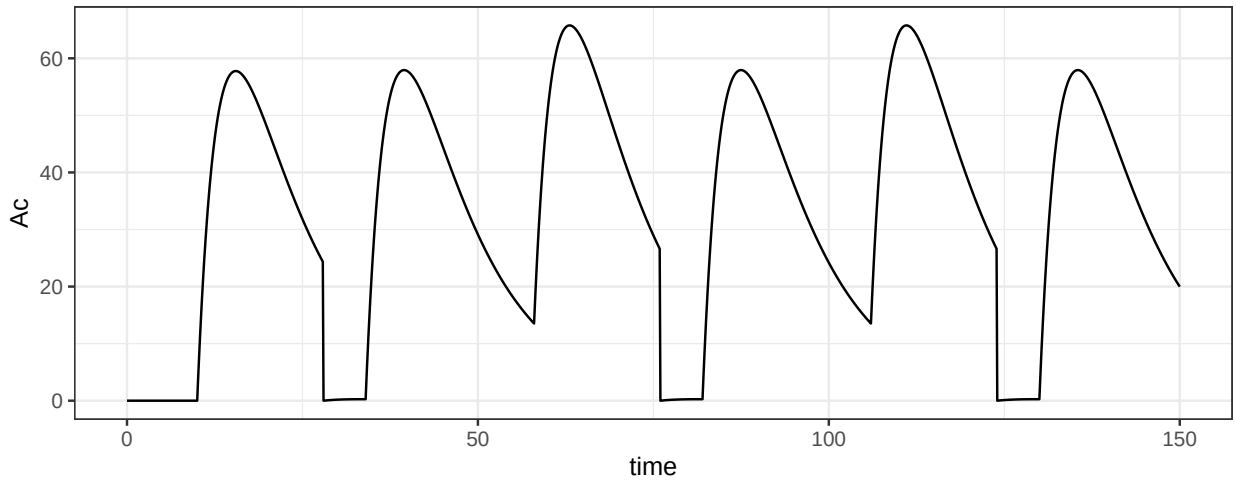
```
trt1 <- list(amt=100, time=seq(10,136,by=24), type=1)
```

while trt2 defines the time at which the central compartment is emptied

```
trt2 <- list(amt=0, time=seq(28,136,by=48), type=2)
```

Both actions can then be combined:

```
trt <- list(trt1, trt2)
r <- simulx(model=pk.model1, parameter = p, treatment = trt, output = out)
ggplot(r$Ac, aes(time,Ac)) + geom_line()
```



Remark: With mlxR <= 3.3.3, it is mandatory to define the amount for trt2 even if it is ignored (use amt=0 for instance). It is not necessary with mlxR >= 3.3.4:

```
## mlxR >= 3.3.4 allows the following definition of emptying actions:
trt2 <- list(time=seq(28,136,by=48), type=2)
```

Emptying multiple compartments

It is possible to set several, or even all, components of the system to 0

```
pk.model2 <- inlineModel("
[LONGITUDINAL]
input = {ka, k}

PK:
depot(target=Ad,  type=1)
empty(target=Ad,  type=2)
empty(target=Ac,  type=3)
empty(target=all, type=4)

EQUATION:
ddt_Ad = -ka*Ad
ddt_Ac =  ka*Ad - k*Ac
")

p <- c(ka=0.3,  k=0.1)
out <- list(name= c("Ad", "Ac"), time = seq(0,150,0.1))
```

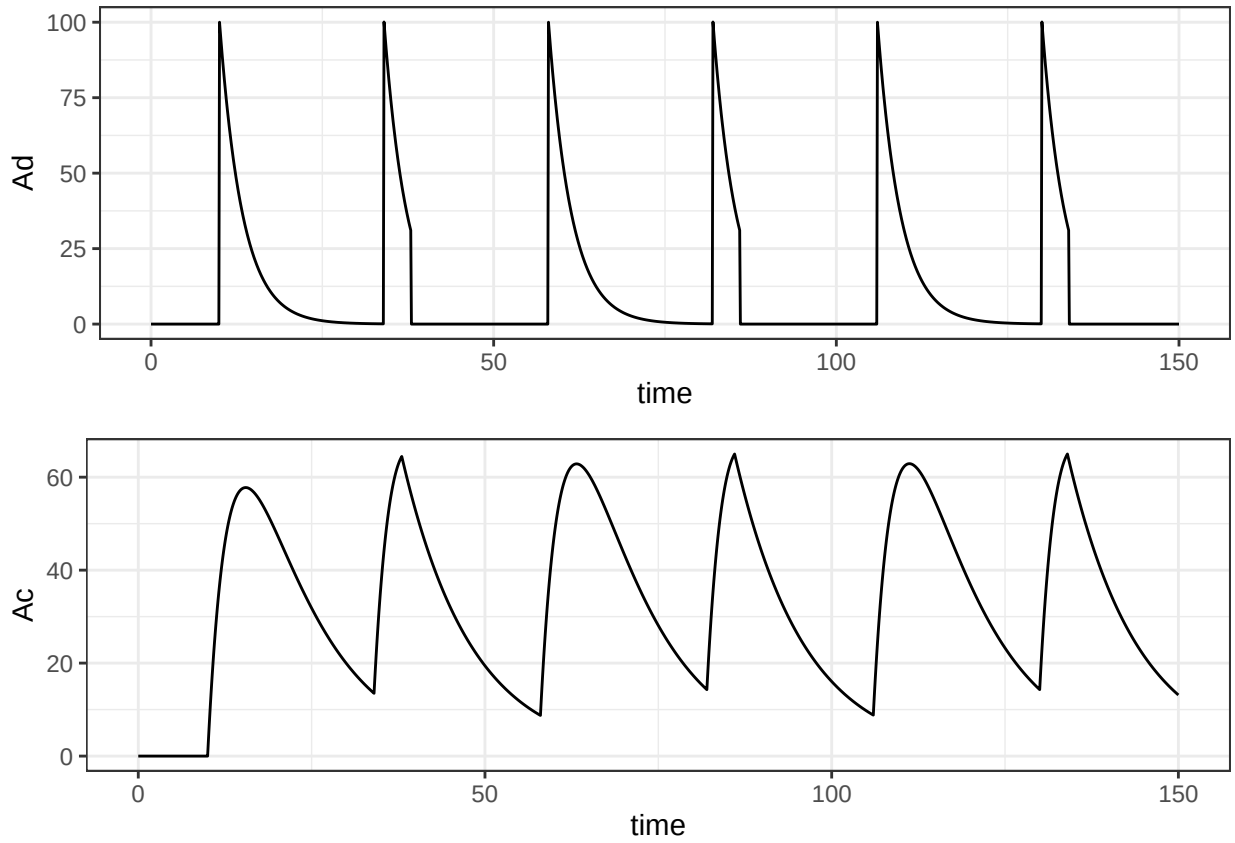
Here, we define several sequences of actions:

```
trt1 <- list(amt=100, time=seq(10,136,by=24), type=1) ## put doses in the depot compartment
trt2 <- list(amt=0,   time=seq(38,136,by=48), type=2) ## empty the depot compartment
trt3 <- list(amt=0,   time=seq(38,136,by=48), type=3) ## empty the central compartment
trt4 <- list(amt=0,   time=seq(38,136,by=48), type=4) ## empty all the compartments
```

In this first example, only the depot compartment is emptied at various times:

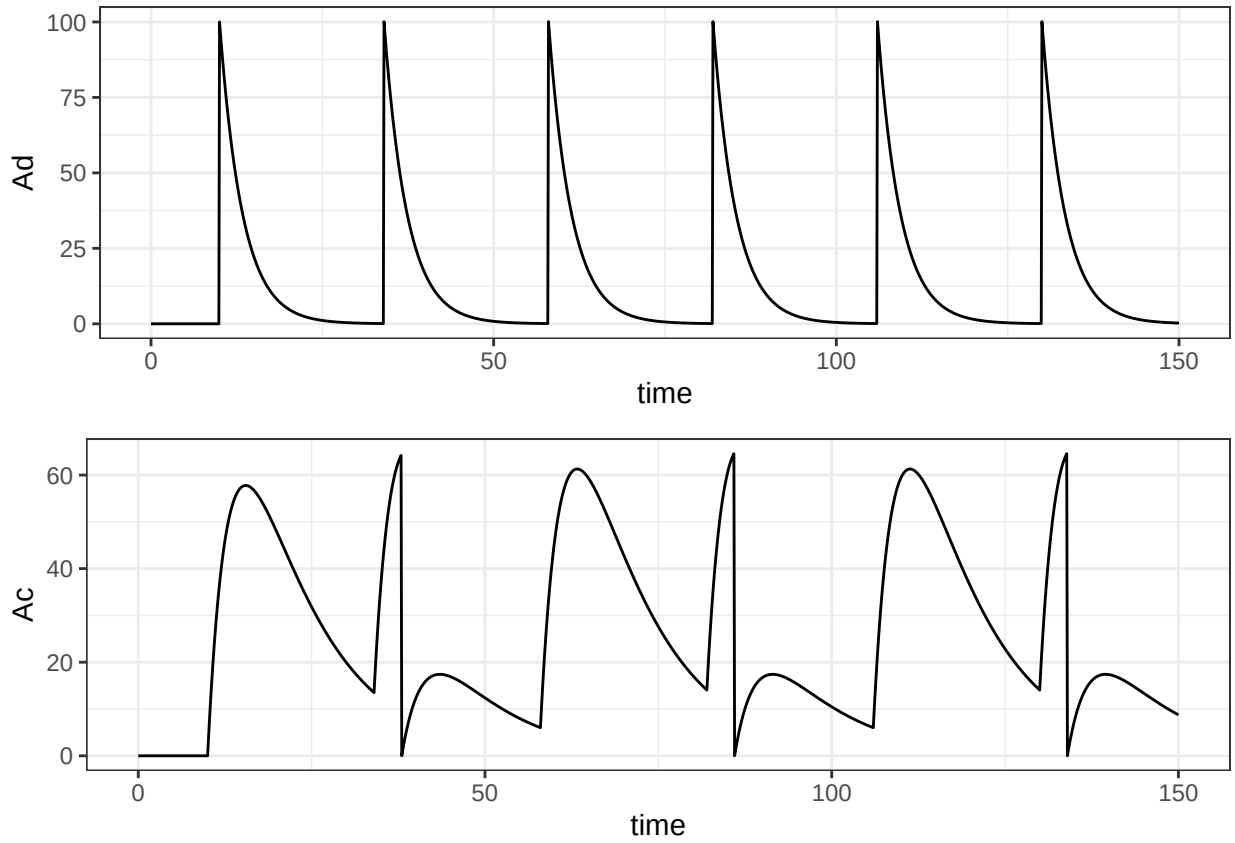
```
library(gridExtra)
trt <- list(trt1, trt2)
r <- simulx(model=pk.model2, parameter = p, treatment = trt, output = out)
```

```
pl1 <- ggplot(r$Ad, aes(time,Ad)) + geom_line()
pl2 <- ggplot(r$Ac, aes(time,Ac)) + geom_line()
grid.arrange(pl1, pl2)
```



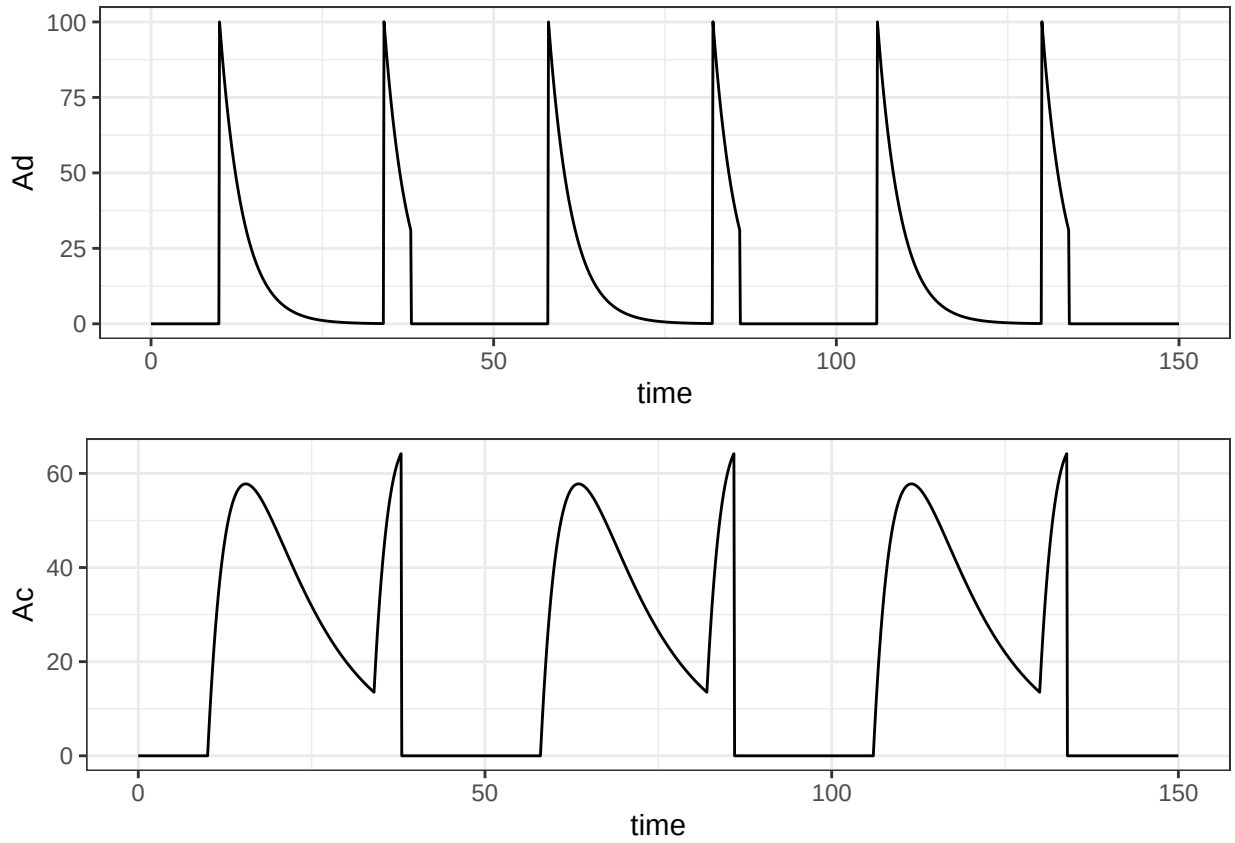
In this second example, only the central compartment is emptied at various times:

```
trt <- list(trt1, trt3)
r <- simulx(model=pk.model2, parameter = p, treatment = trt, output = out)
pl1 <- ggplot(r$Ad, aes(time,Ad)) + geom_line()
pl2 <- ggplot(r$Ac, aes(time,Ac)) + geom_line()
grid.arrange(pl1, pl2)
```



In this last example, all the compartments are emptied simultaneously at various times:

```
trt <- list(trt1, trt4)
r <- simulx(model=pk.model2, parameter = p, treatment = trt, output = out)
pl1 <- ggplot(r$Ad, aes(time,Ad)) + geom_line()
pl2 <- ggplot(r$Ac, aes(time,Ac)) + geom_line()
grid.arrange(pl1, pl2)
```



Resetting compartments

Resetting a single compartment

Instead of setting a component to 0, it is possible to reset it to its nitial values using the `reset` macro. Of course, emptying a compartment of resetting it to its initial state are equivalent actions for compartment which are initially empty. This may be not the case for PD models with non-zero baseline

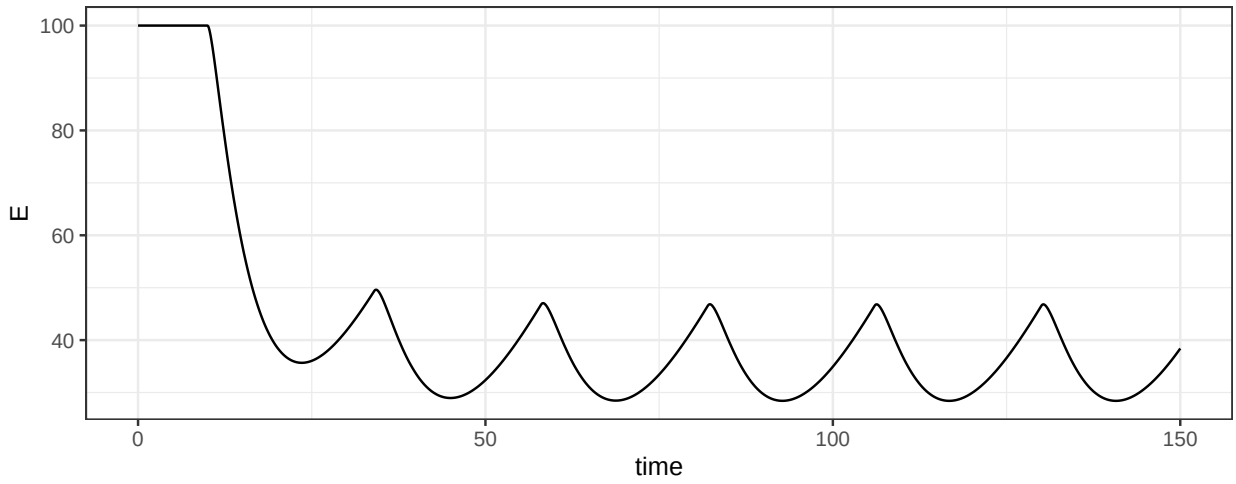
```
pkpd.model1 <- inlineModel("
[LONGITUDINAL]
input = {ka, k, E0, IC50, kout}

PK:
depot(target=Ad,  type=1)
reset(target=E,   type=2)

EQUATION:
E_0 = E0
kin = E0*kout

ddt_Ad = -ka*Ad
ddt_Ac =  ka*Ad - k*Ac
ddt_E  =  kin*(1 - Ac/(Ac+IC50)) - kout*E
")

p <- c(ka=0.3, k=0.1, E0=100, IC50=20, kout=0.2)
out <- list(name="E", time = seq(0,150,0.1))
```

Let us define the dosing regimen

```
trt1 <- list(amt=100, time=seq(10,136,by=24), type=1)
```

and compute the predicted effect

```
r <- simulx(model=pkpd.model1, parameter = p, treatment = trt1, output = out)
ggplot(r$E, aes(time,E)) + geom_line()
```

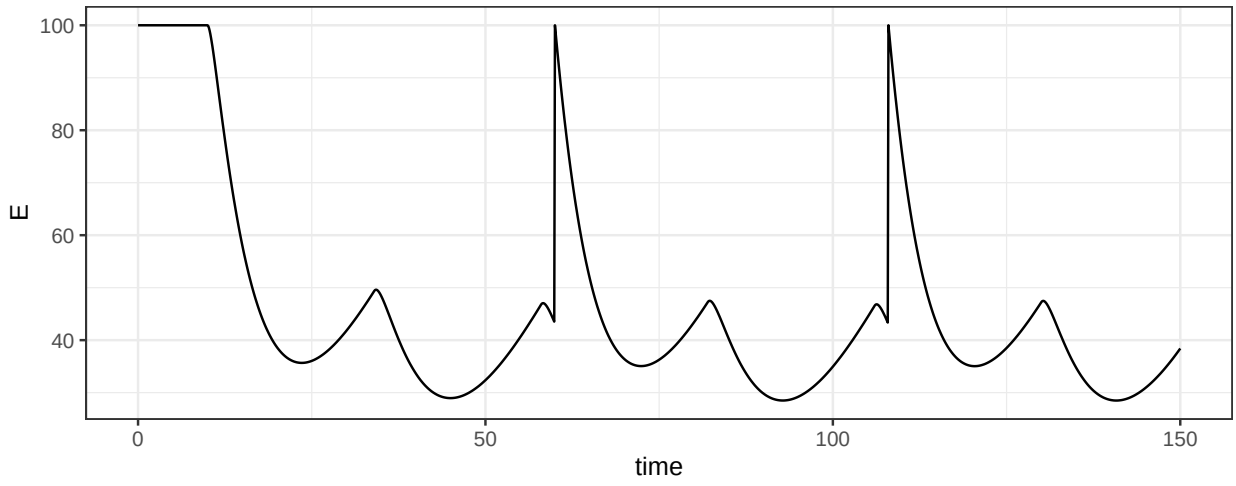


Imagine now that component E is reset to its initial value E_0 at various times:

```
trt2 <- list(amt=0,   time=seq(60,136,by=48), type=2)
```

We can then combine these two types of action:

```
trt <- list(trt1, trt2)
r <- simulx(model=pkpd.model1, parameter = p, treatment = trt, output = out)
ggplot(r$E, aes(time,E)) + geom_line()
```



Resetting multiple compartments

Several, or all, components of the system can be reset to their initial values:

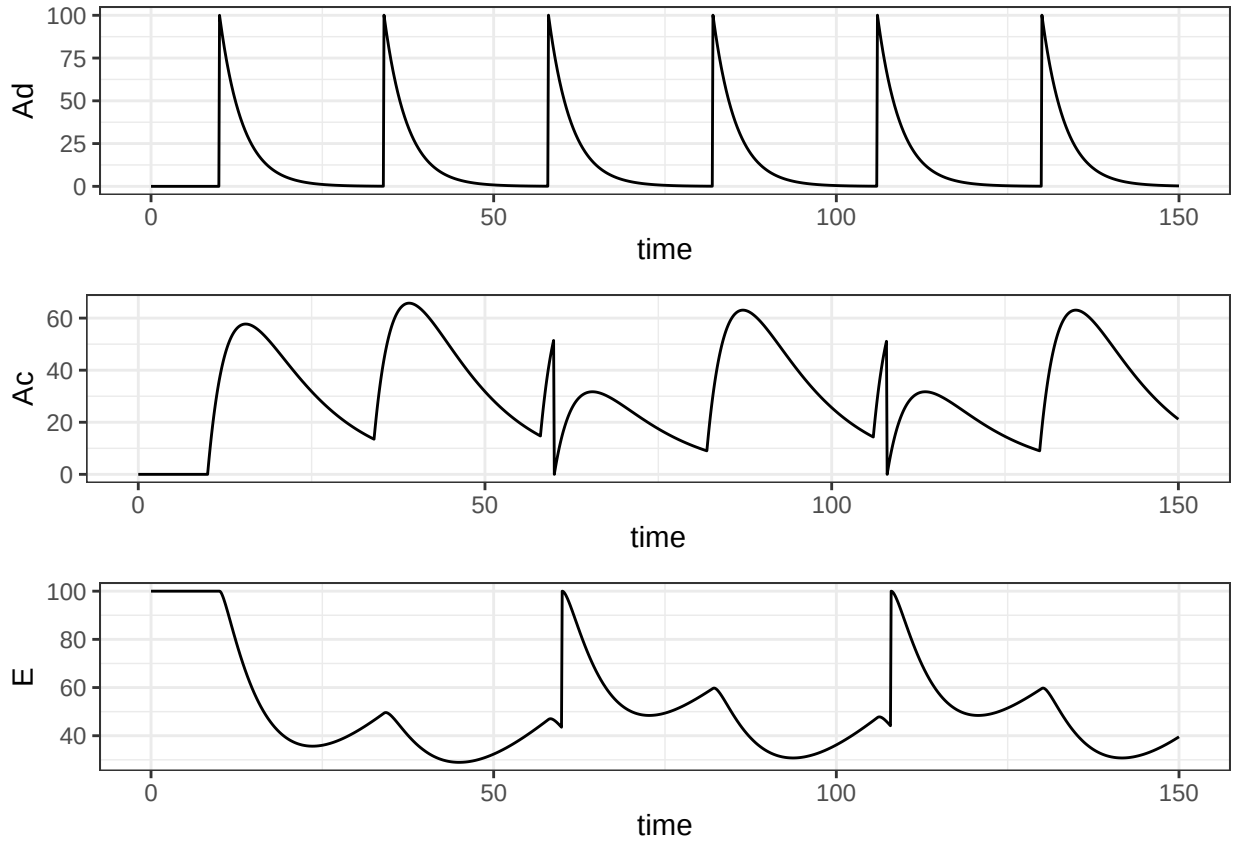
```
pkpd.model2 <- inlineModel("
[LONGITUDINAL]
input = {ka, k, E0, IC50, kout}

PK:
depot(target=Ad,  type=1)
reset(target=E,   type=2)
reset(target=Ac,  type=3)
reset(target=all, type=4)

EQUATION:
E_0 = E0
kin = E0*kout

ddt_Ad = -ka*Ad
ddt_Ac =  ka*Ad - k*Ac
ddt_E  =  kin*(1 - Ac/(Ac+IC50)) - kout*E
")

p <- c(ka=0.3, k=0.1, E0=100, IC50=20, kout=0.2)
out <- list(name=c("Ad", "Ac", "E"), time = seq(0,150,0.1))
```

We can define several sequences of actions

```
trt1 <- list(amt=100, time=seq(10,136,by=24), type=1)
trt2 <- list(amt=0,   time=seq(60,136,by=48), type=2)
trt3 <- list(amt=0,   time=seq(60,136,by=48), type=3)
trt4 <- list(amt=0,   time=seq(60,136,by=48), type=4)
```

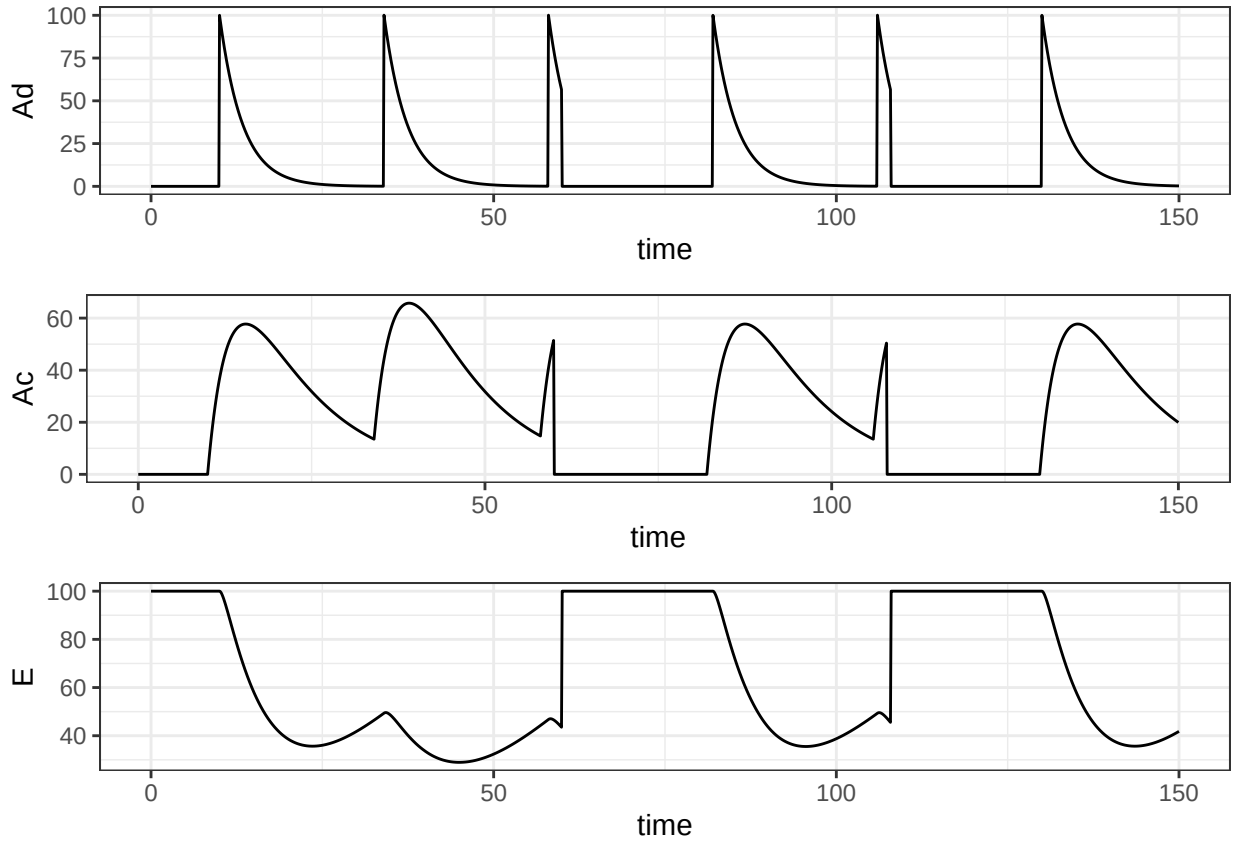
and combine them, by resetting, for instance, A_c and E to their initial values.

```
trt <- list(trt1, trt2, trt3)
r <- simulx(model=pkpd.model2, parameter = p, treatment = trt, output = out)
pl1 <- ggplot(r$Ad, aes(time,Ad)) + geom_line()
pl2 <- ggplot(r$Ac, aes(time,Ac)) + geom_line()
pl3 <- ggplot(r$E, aes(time,E)) + geom_line()
grid.arrange(pl1, pl2, pl3)
```



Here, all the components of the system are reset to their initial values:

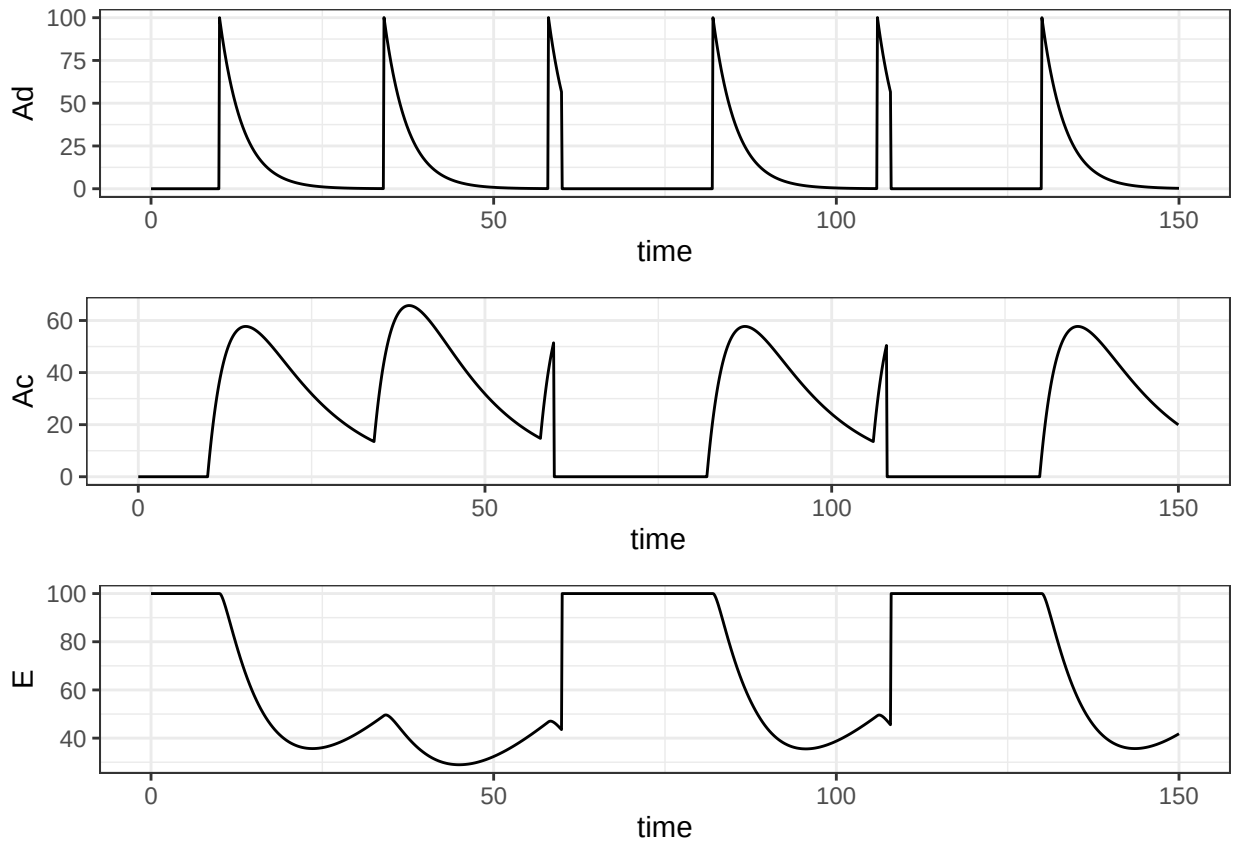
```
trt <- list(trt1, trt4)
r <- simulx(model=pkpd.model2, parameter = p, treatment = trt, output = out)
pl1 <- ggplot(r$Ad, aes(time,Ad)) + geom_line()
pl2 <- ggplot(r$Ac, aes(time,Ac)) + geom_line()
pl3 <- ggplot(r$E, aes(time,E)) + geom_line()
grid.arrange(pl1, pl2, pl3)
```



Remark: resetting and emptying the depot and central compartments are equivalent actions. Then, we could equivalently define the following actions:

```
pkpd.model2b <- inlineModel("
[LONGITUDINAL]
input = {ka, k, E0, IC50, kout}

PK:
depot(target=Ad,  type=1)
reset(target=E,   type=2)
empty(target=Ac,  type=3)
empty(target=Ad,  type=4)
empty(target=Ac,  type=4)
reset(target=E,   type=4)

EQUATION:
E_0 = E0
kin = E0*kout

ddt_Ad = -ka*Ad
ddt_Ac =  ka*Ad - k*Ac
ddt_E  =  kin*(1 - Ac/(Ac+IC50)) - kout*E
")

r <- simulx(model=pkpd.model2, parameter = p, treatment = trt, output = out)
pl1 <- ggplot(r$Ad, aes(time,Ad)) + geom_line()
pl2 <- ggplot(r$Ac, aes(time,Ac)) + geom_line()
```

```
pl3 <- ggplot(r$E, aes(time,E)) + geom_line()
grid.arrange(pl1, pl2, pl3)
```



Simulation of continuous data

R script: continuous.R

Mlxtran code: model/continuous.txt

Introduction

A model for longitudinal data is implemented in the section `[LONGITUDINAL]` of a script Mlxtran.

The probability distribution of longitudinal data is defined in a block `DEFINITION`, using functions of time defined in the `EQUATION` block.

Possible distributions for continuous data are: normal, log-normal, probit-normal and logit-normal.

Thus, if h is one of these transformation (identity, log, probit or logit), the observations (y_j) are defined as:

$$h(y_j) = h(f(t_j;\psi)) + g(t_j;\psi)\varepsilon_j$$

where f and g are defined in the `EQUATION` block and where $\varepsilon_j \sim_{\text{i.i.d.}} \mathcal{N}(0,1)$.

Here, $f(t_j;\psi)$ is the *prediction* of y_j.

Example

The distributions of two types of continuous data are defined in the following example: $(y_j^{(1)})$ are normally distributed while $(y_j^{(2)})$ are log-normally distributed:

$$y_j^{(1)} = f_1(t_{1j}) \ + \ (a + b f_1(t_{1j}))\varepsilon_j^{(1)} \tag{23}$$

$$\log(y_j^{(2)}) = \log(f_2(t_{2j})) \ + \ b\,\varepsilon_j^{(1)} \tag{24}$$

where

$$f_1(t) = \frac{D}{V}e^{-k\,t} \tag{25}$$

$$f_2(t) = \frac{D\,k_a}{V(k_a-k)}(e^{-k\,t} - e^{-k_a\,t}) \tag{26}$$

and where $D = 100$ is given.

Note that the observation times $(t_{1j}, 1 \leq j \leq n_1)$ for $y^{(1)}$ and $(t_{2j}, 1 \leq j \leq n_2)$ for $y^{(2)}$ can be different.

This model is implemented in the file `model/continuous.txt`:

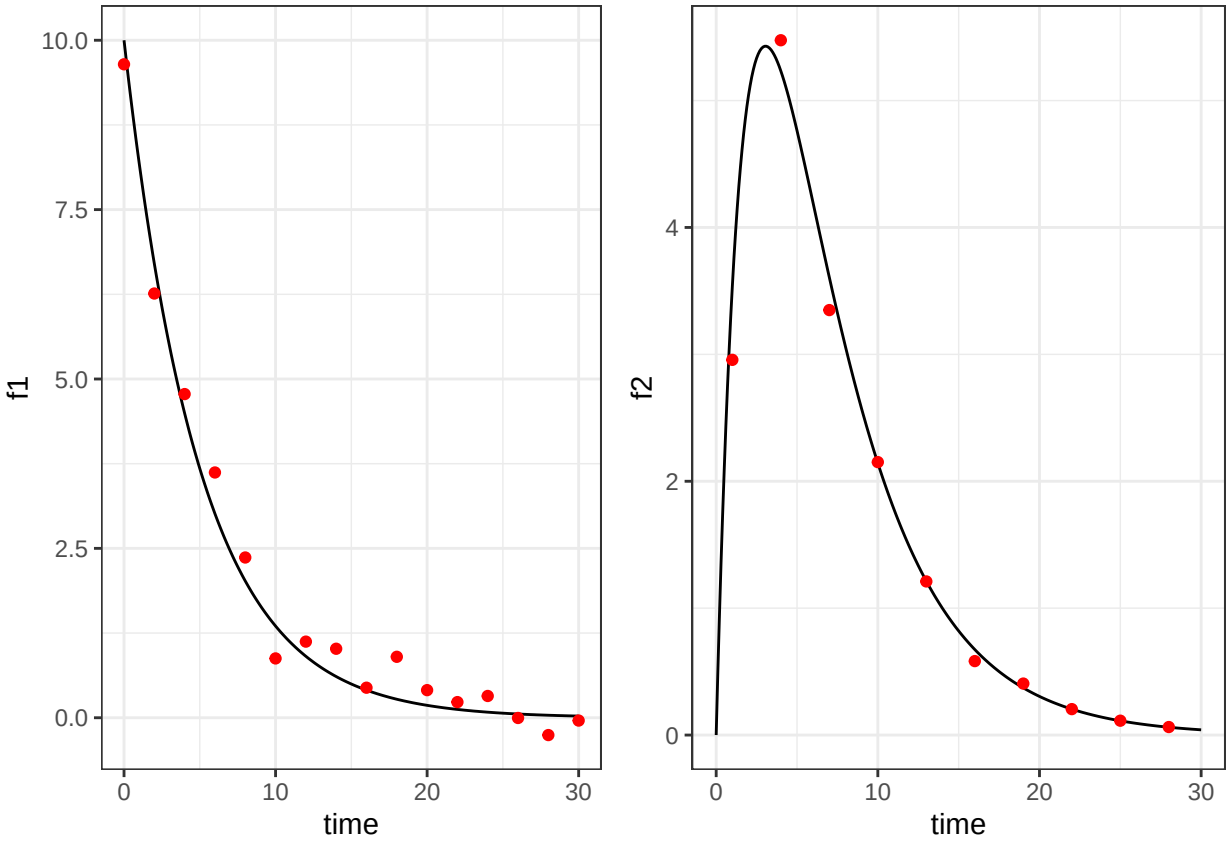
We will use simulx for computing both f_1 and f_2 every 0.1h between 0h and 30h, for generating observations $y^{(1)}$ every 2 hours between 0h and 30h and for generating observations $y^{(2)}$ every 3 hours between 1h and 30h

```
p <- c(ka=0.5, V=10, k=0.2, a=0.3, b=0.1)

f <- list(name=c('f1','f2'), time=seq(0, 30, by=0.1))
```

```
y1 <- list(name='y1', time=seq(0, 30, by=2))
y2 <- list(name='y2', time=seq(1, 30, by=3))

res <- simulx(model     = 'model/continuous.txt',
              parameter = p,
              output    = list(f, y1, y2))
```

Here, `res` is a list with 4 elements: `f1`, `f2`, `y1` and `y2` that we can display on 2 separate plots:

```
library(gridExtra)
plot1=ggplot(data=res$f1, aes(x=time, y=f1)) +
  geom_line(size=0.5) +
  geom_point(data=res$y1, aes(x=time, y=y1), colour="red")

plot2=ggplot(data=res$f2, aes(x=time, y=f2)) +
  geom_line(size=0.5) +
  geom_point(data=res$y2, aes(x=time, y=y2), colour="red")

grid.arrange(plot1, plot2, ncol=2)
```



Simulation of categorical data

R script: categorical.R

Mlxtran code: model/categoricalA.txt ; model/categoricalB.txt

---

Introduction

Assume now that the observed data takes its values in a fixed and finite set of nominal categories $\{c_1, c_2, \ldots, c_K\}$

Considering the observations $(y_j,\, 1 \leq j \leq n)$ as a sequence of conditionally independent random variables, the model is completely defined by the probability mass functions

$$\mathbb{P}(y_j = c_k) \quad \text{for } k = 1, \ldots, K \text{ and } 1 \leq j \leq n$$

For a given j, the sum of the K probabilities is 1, so in fact only $K-1$ of them need to be defined.

Ordinal data further assume that the categories are ordered:

$$c_1 \leq c_2, \leq \ldots \leq c_K.$$

Instead of defining the probabilities of each category, it may be convenient to define the *cumulative probabilities*

$$\mathbb{P}(y_j \leq c_k), \quad \text{for } k = 1, \ldots, K-1$$

or the *cumulative logits*

$$\text{logit}\left(\mathbb{P}(y_j \leq c_k)\right), \quad \text{for } k = 1, \ldots, K-1$$

where $\text{logit}(p) = \log(p/(1-p))$.

The distribution of ordered categorical data can be defined in the block `DEFINITION`of the Section `[LONGITUDINAL]`using either the probability mass functions, the cumulative probabilities or the cumulative logits.

Example using the probabilities

y_j takes its values in $\{1, 2, 3\}$. We use cumulative logits to define the distribution of y_j:

$$\begin{aligned}
\text{logit}\left(\mathbb{P}(y_j \leq 1)\right) &= a - b\,t_j \\
\text{logit}\left(\mathbb{P}(y_j \leq 2)\right) &= a - b\,t_j/2
\end{aligned}$$

We can then derive the probabilities for each category:

$$\begin{aligned}
\mathbb{P}(y_j = 1) &= \frac{1}{1 + e^{-(a - b\,t_j)}} \\
\mathbb{P}(y_j = 2) &= \frac{1}{1 + e^{-(a - b\,t_j/2)}} - \mathbb{P}(y_j = 1) \\
\mathbb{P}(y_j = 3) &= 1 - \mathbb{P}(y_j = 1) - \mathbb{P}(y_j = 2)
\end{aligned}$$

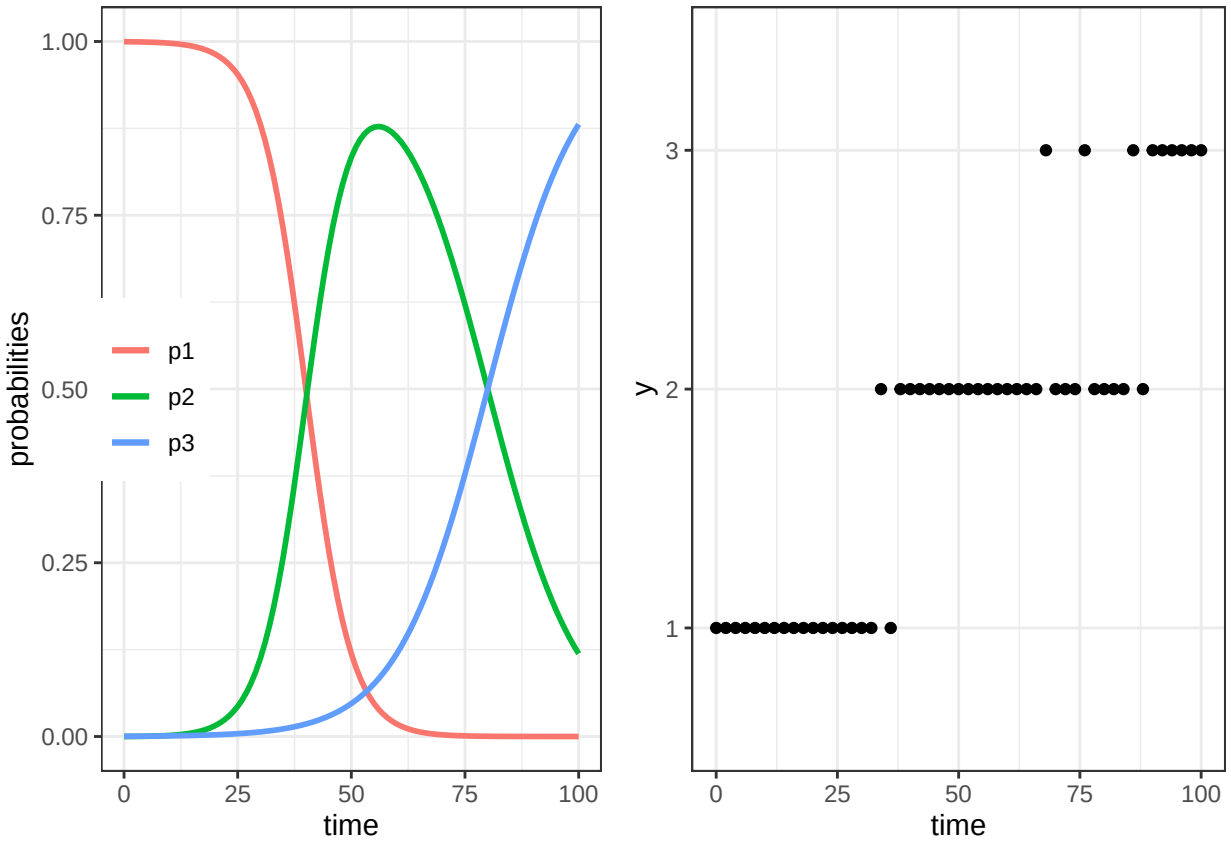
This model is implemented in the file `categoricalA.txt`:

We will use simulx for computing p_1, p_2 and p_3 every hour between 0h and 100h and for generating observations (y_j) every 2 hours between 0h and 100h, with $a = 8$ and $b = 0.2$.

```
seed <- 12345
p    <- c(a=8,b=0.2)
pr   <- list(name=c('p1','p2','p3'), time=0:100)
y    <- list(name='y', time=seq(0, 100, by=2))
res  <- simulx(model     = 'model/categoricalA.txt',
               parameter = p,
               output    = list(pr, y),
               settings  = list(seed=seed))
```

we can now plot the 3 probabilities and the simulated data:

```
library(gridExtra)
plot1=ggplot() + ylab("probabilities") +
  geom_line(data=res$p1, aes(x=time, y=p1, colour="p1"),size=1) +
  geom_line(data=res$p2, aes(x=time, y=p2, colour="p2"),size=1) +
  geom_line(data=res$p3, aes(x=time, y=p3, colour="p3"),size=1) +
  theme(legend.position=c(0.1,0.5),legend.title=element_blank())
plot2=ggplot() + geom_point(aes(x=time, y=y), data=res$y)
grid.arrange(plot1, plot2, ncol=2)
```



Example using the cumulative logits

We can equivalently use `categoricalB.txt` where the cumulative logits define the distribution of the y_j's:

We therefore use simulx for computing lp_1 and lp_2 and for generating observations (y_j)

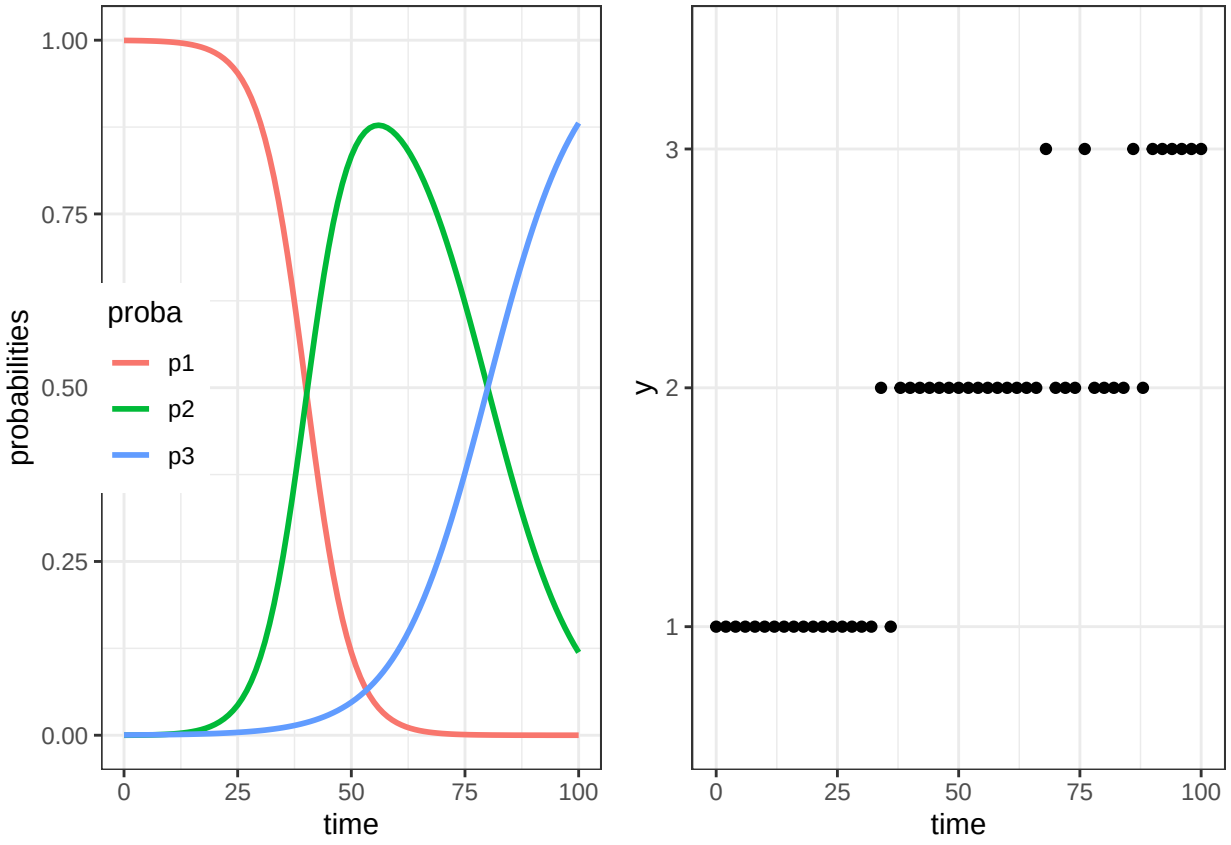
```
lpr  <- list(name=c('lp1','lp2'), time=0:100)
res  <- simulx(model     = 'model/categoricalB.txt',
               parameter = p,
               output    = list(y,lpr),
               settings  = list(seed=seed))
```

We can then derive the probability mass function from the cumulative logits

```
p1 <- 1/(1+exp(-res$lp1$lp1))
p2 <- 1/(1+exp(-res$lp2$lp2))-p1
p3 <- 1-p1-p2
```

and create a single data frame with these probabilities

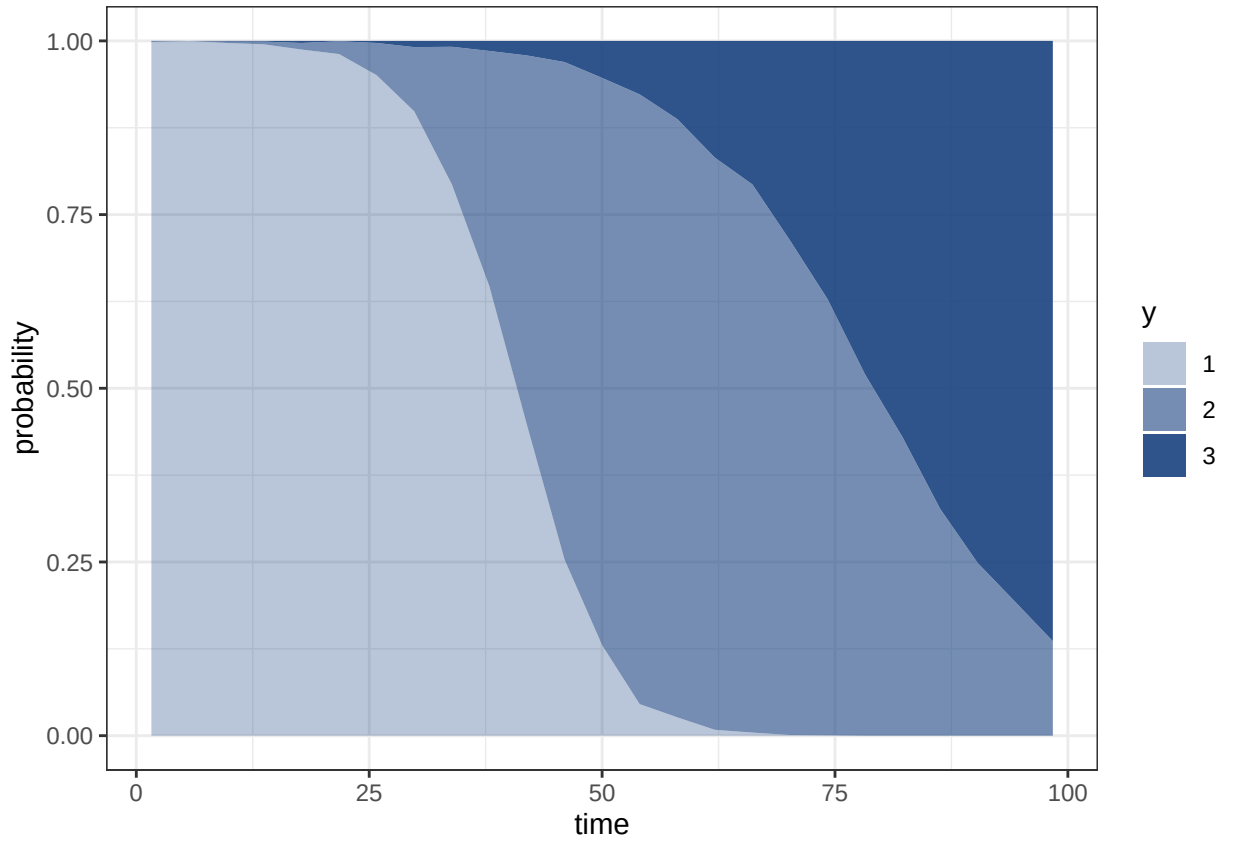
```
library(reshape2)
pr <- data.frame(time=0:100,p1,p2,p3)
pr <- melt(pr ,  id = 'time', variable.name = 'proba')
## pr is a data frame with columns "id", "proba" and "value"
plot1=ggplot(pr, aes(time,value)) + geom_line(aes(colour = proba),size=1) +
  ylab('probabilities') + theme(legend.position=c(.1, .5))
plot2=ggplot() + geom_point(aes(x=time, y=y), data=res$y)
grid.arrange(plot1, plot2, ncol=2)
```



When N replicates of the data are simulated, catplotmlx can be used to display the empirical distribution of the simulated data:

```
g <- list(size=1000)
res <- simulx(model     = 'model/categoricalA.txt',
```

```
                parameter = p,
                output    = y,
                group     = g)
plot3 <- catplotmlx(res$y, breaks=25)
print(plot3)
```



Simulation of categorical data with Markovian dependence

R script: markovian.R

Mlxtran code: model/markovianA.txt ; model/markovianB.txt ; model/markovianC.txt

---

Introduction

For the sake of simplicity, we will assume here that the observations (y_j) take their values in $\{1, 2, \ldots, K\}$.

We have so far assumed that the categorical observations $(y_j,\, j = 1, 2, \ldots, n)$ are independent. It is however possible to introduce dependence between observations from the same individual by assuming that $(y_j,\, j = 1, 2, \ldots, n)$ forms a Markov chain. For instance, a Markov chain with memory 1 assumes that all that is required from the past to determine the distribution of y_j is the value of the previous observation y_{j-1}, i.e., for all $k = 1, 2, \ldots, K$,

$$\mathbb{P}(y_j = k \,|\, y_{j-1}, y_{j-2}, y_{j-3}, \ldots) = \mathbb{P}(y_j = k | y_{j-1}).$$

Discrete-time Markov chain

If observation times are regularly spaced (constant length of time between successive observations), we can consider the observations $(y_j,\, j = 1, 2, \ldots, n)$ to be a discrete-time Markov chain. Here, the probability distribution of the sequence (y_j) is defined by

- The distribution $\pi = (\pi_k, k = 1, 2, \ldots, K)$ of the first observation y_1:

$$\pi_k = \mathbb{P}(y_1 = k).$$

- The sequence of *transition matrices* $(Q_j,\, j = 2, 3, \ldots)$ where for each j, $Q_j = (q_{j,\ell,k}, 1 \leq \ell, k \leq K)$ is a matrix of size $K \times K$ such that

$$q_{j,\ell,k} = \mathbb{P}(y_j = k | \, y_{j-1} = \ell) \quad \text{for all } \; 1 \leq \ell, k \leq K, \tag{27}$$

$$\sum_{k=1}^{K} q_{j,\ell,k} = 1 \quad \text{for all } \; 1 \leq \ell \leq K. \tag{28}$$

Continuous-time Markov chain

The previous situation can be extended to the case where time intervals between observations are irregular by modeling the sequence of states as a *continuous-time Markov process*. The difference is that rather than transitioning to a new (possibly the same) state at each time step, the system remains in the current state for some random amount of time before transitioning.

This process is now characterized by *transition rates* instead of transition probabilities:

$$\mathbb{P}(y(t+h) = k \,|\, y(t) = \ell) = h\, \rho_{\ell k}(t) + o(h), \qquad k \neq \ell.$$

The probability that no transition happens between t and $t + h$ is

$$\mathbb{P}(y(s) = \ell, \forall s \in (t, t+h) \mid y(t) = \ell) = e^{h\, \rho_{\ell\ell}(t)}.$$

Furthermore, for any time t, the transition rates $(\rho_{\ell,k}(t))$ satisfy for any $1 \leq \ell \leq K$,

$$\sum_{k=1}^{K} \rho_{\ell k}(t) = 0.$$

Examples

Example 1: Homogeneous discrete-time Markov chain

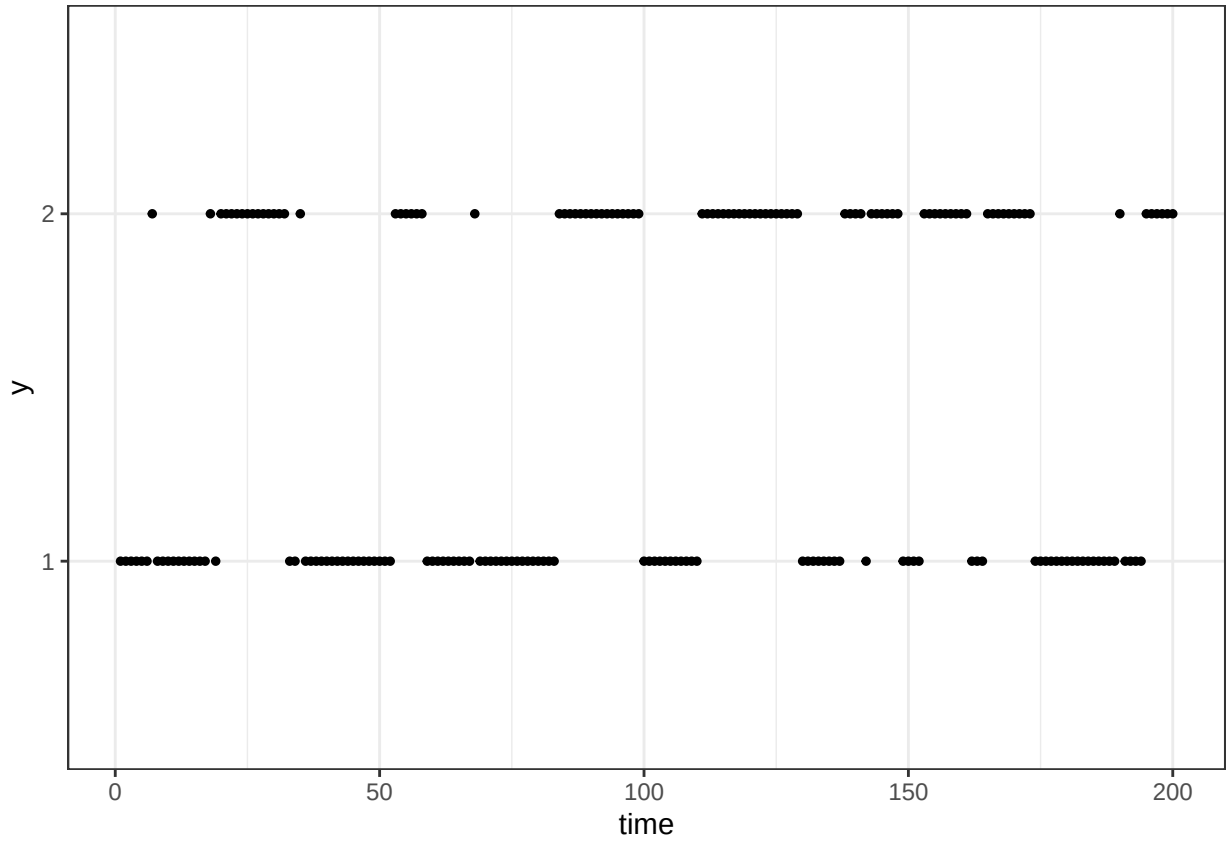
We consider a very simple Markov model with 2 states and where the transition probabilities remain constant over time:

$$\mathbb{P}(y_j = 2|y_{j-1} = 1) = 1 - \mathbb{P}(y_j = 1|y_{j-1} = 1) = p_{12}$$
$$\mathbb{P}(y_j = 1|y_{j-1} = 2) = 1 - \mathbb{P}(y_j = 2|y_{j-1} = 2) = p_{21}$$

This model is implemented in the file `categoricalA.txt`:

We can now generate a sequence of $n = 200$ observations with this model using `simulx`.

```
seed <- 12345
p    <- c(p12=0.2, p21=0.15)
y    <- list(name='y', time=seq(1, 200))
res1  <- simulx(model     = 'model/markovianA.txt',
                parameter = p,
                output    = y,
                settings  = list(seed=seed))

ggplot(data=res1$y) + geom_point(aes(x=time, y=y),size=1)
```



Example 2: Non homogeneous discrete-time Markov chain

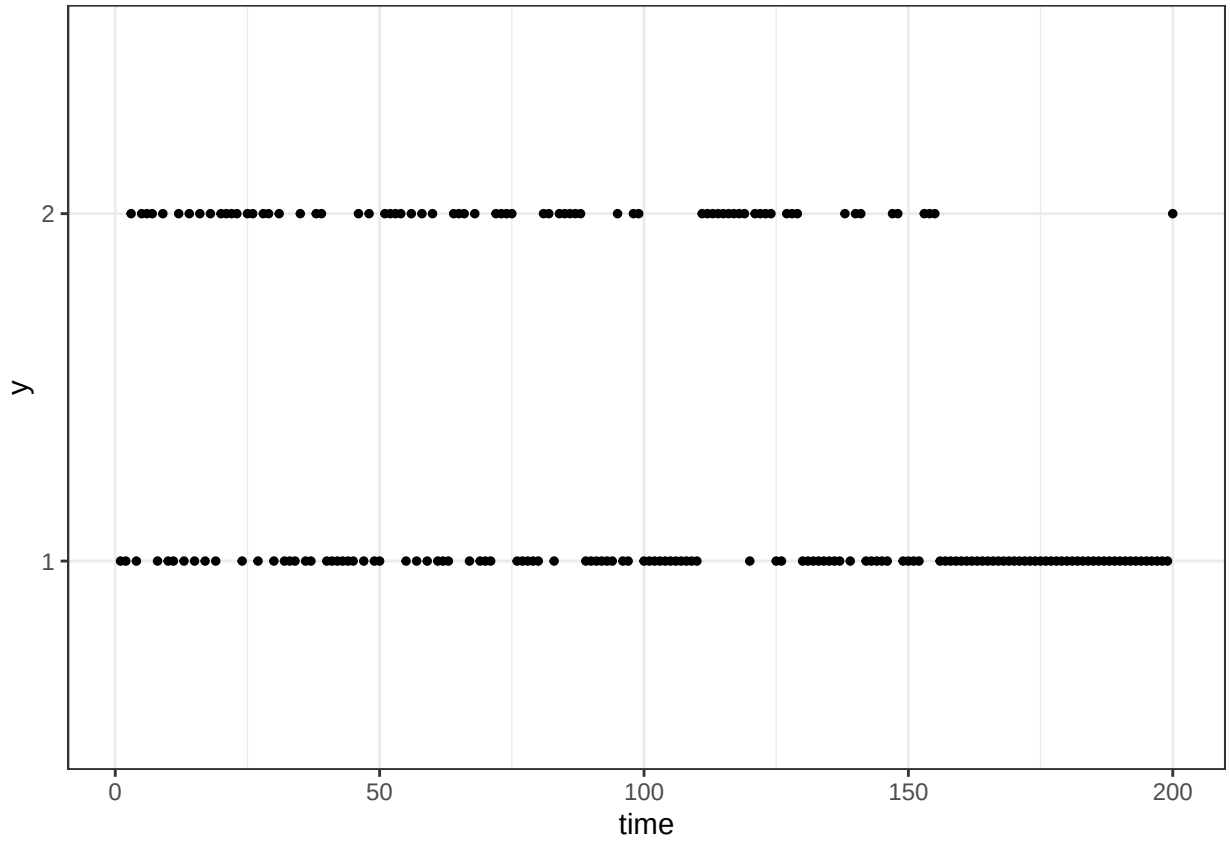
The transition probabilities change with time in this second example. These *cumulative logits* are defined as follow:

$$\text{logit}(\mathbb{P}(y_j = 2|y_{j-1} = 1)) = a + b\,t_j$$
$$\text{logit}(\mathbb{P}(y_j = 1|y_{j-1} = 2)) = c + d\,t_j$$

This model is implemented in the file `categoricalB.txt`:

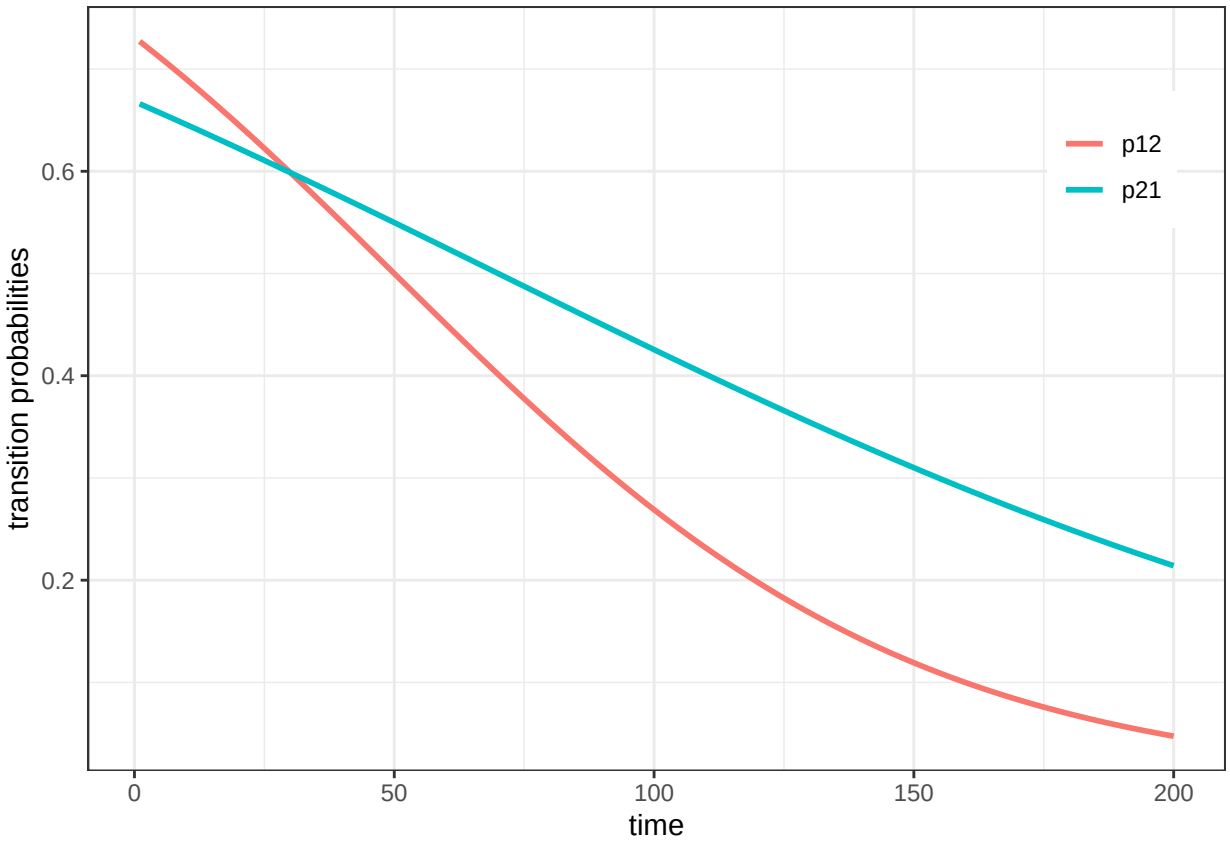
We can generate a new sequence of $n = 200$ observations with this new model.

```
p    <- c(a=1,b=-0.02,c=0.7,d=-0.01)
f    <- list(name=c('p12','p21'), time=seq(1, 200))
res2 <- simulx(model     = 'model/markovianB.txt',
               parameter = p,
               output    = list(y,f),
               settings  = list(seed=seed))

ggplot(res2$y) + geom_point(aes(x=time, y=y),size=1)
```



and we can plot the transition probabilities $p_{12}(t)$ and $p_{21}(t)$ defined in the model.

```
library(reshape2)
r <- melt(merge(res2$p12,res2$p21) ,  id = 'time', variable.name = 'f')
print(ggplot(r, aes(time,value)) + geom_line(aes(colour = f),size=1) +
        ylab('transition probabilities') + guides(colour=guide_legend(title=NULL)) +
        theme(legend.position=c(.9, .8)))
```



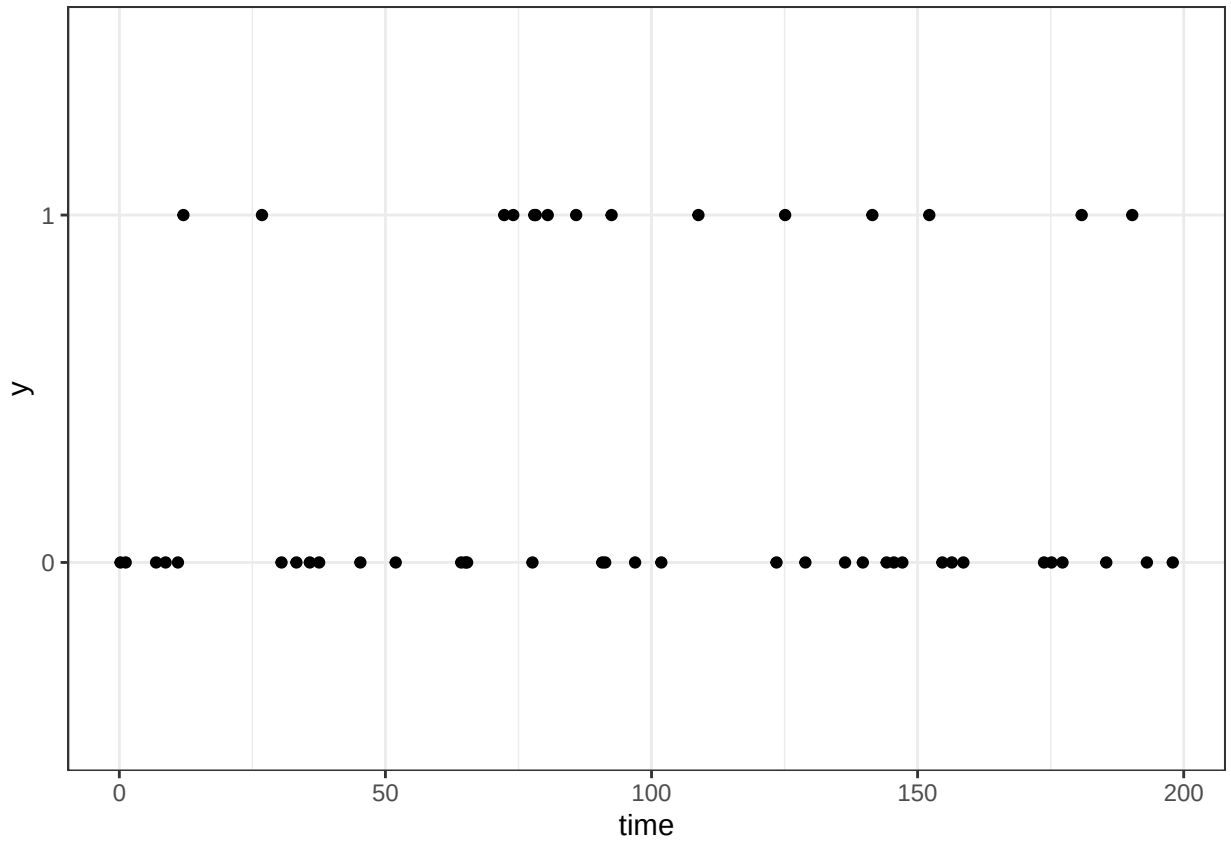
Example 3: Continuous-time Markov chain

We now consider a continuous-time Markov chain: the observation times are irregularly distributed between 0 and 200.

The transition rates are constant over time in the model implemented in the file `categoricalC.txt`:

50 time points are randomly distributed between 0 and 200 in this example.

```
q    <- c(q01=0.8, q10=1)
y    <- list(name='y', time=sort(runif(50,min=0,max=200)))
res3  <- simulx(model     = 'model/markovianC.txt',
                parameter = q,
                output    = y,
                settings  = list(seed=12345))

plot(ggplot(data=res3$y) + geom_point(aes(x=time, y=y)))
```


1
0
y
0
50
100
150
200
time

Simulation of count data

R script: count.R

Mlxtran code: model/count.txt

Introduction

Longitudinal count data is a special type of longitudinal data that can take only nonnegative integer values $\{0, 1, 2, \ldots\}$ that come from counting something, e.g., the number of seizures, hemorrhages or lesions in each given time period.

Considering the observations $(y_j,\, 1 \leq j \leq n)$ as a sequence of conditionally independent random variables, the model is completely defined by the probability mass functions

$$\mathbb{P}(y_j = k) \quad \text{for } k = 0, 1, 2, \ldots \text{ and } 1 \leq j \leq n$$

The distribution of count data can be defined in the block `DEFINITION`of the Section `[LONGITUDINAL]`using the probability mass functions.

Examples

Example 1: Poisson distribution

In this example, the Poisson distribution is used for defining the distribution of y_j:

$$y_j \sim \text{Poisson}(\lambda_j)$$

where the Poisson intensity λ_j is function of time:

$$\lambda_j = a + b\, t_j$$

This model is first implemented using the Mlxtran library of distributions:

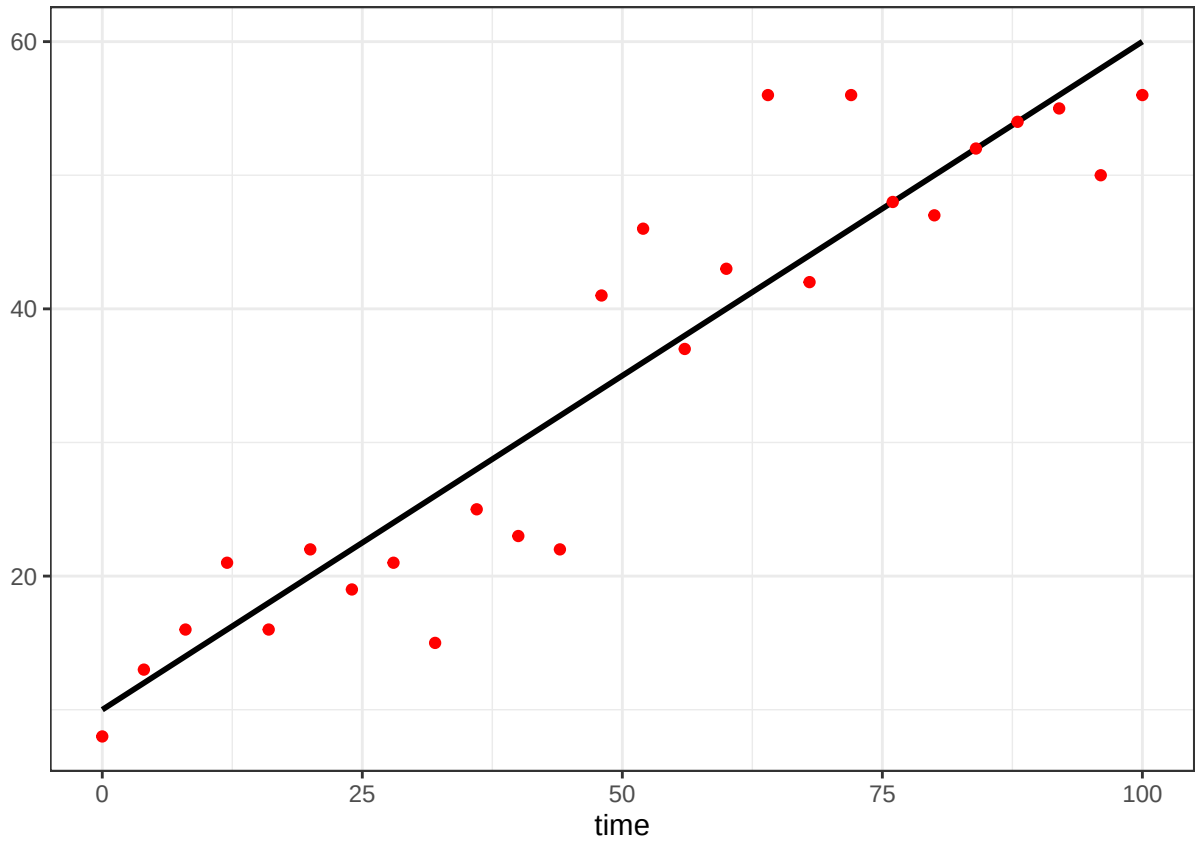
```
poissonModela <- inlineModel("
[LONGITUDINAL]
input =  {a,b}

EQUATION:
lambda = a +b*t

DEFINITION:
y = {distribution=poisson, lambda=lambda}
")
```

We will use simulx for computing λ every hour between 0h and 100h and for generating observations (y_j) every 2 hours between 0h and 100h, with $a = 10$ and $b = 0.5$.

```
p <- c(a=10, b=0.5)
l <- list(name='lambda', time=seq(0, 100, by=1))
y <- list(name='y', time=seq(0, 100, by=4))

res1a <- simulx(model=poissonModela,
                parameter=p,
                output=list(l, y))
```

we can now plot the Poisson intensity and the simulated data:

```
print(ggplot(aes(x=time, y=lambda), data=res1a$lambda) + geom_line(size=1) +
        geom_point(aes(x=time, y=y), data=res1a$y, color="red") + ylab("") )
```



Remark: instead of using the library of distributions it is possible to define the probability mass function in the Mlxtran code:

```
poissonModelb <- inlineModel("
[LONGITUDINAL]
input =  {a,b}

EQUATION:
lambda=a +b*t

DEFINITION:
y = {type=count, P(y=k)=exp(-lambda)*(lambda^k)/factorial(k)}
")
```

Both implementations are equivalent.

Example 2: Negative binomial distribution

Suppose there is a sequence of independent Bernoulli trials, each trial having two potential outcomes called "success" and "failure". In each trial the probability of success is p and of failure is $(1-p)$. We are observing this sequence until a predefined number n of failures has occurred. Then the random number of successes, y, we have seen will have the negative binomial distribution.

For $k = 0, 1, 2, \ldots$,

$$\mathbb{P}(y=k) = \left(\begin{array}{c} k+n-1 \\ k \end{array} \right) p^k (1-p)^n$$

Let us use `simulx` for drawing a random sample of size $N = 10000$ from a negative binomial distribution:

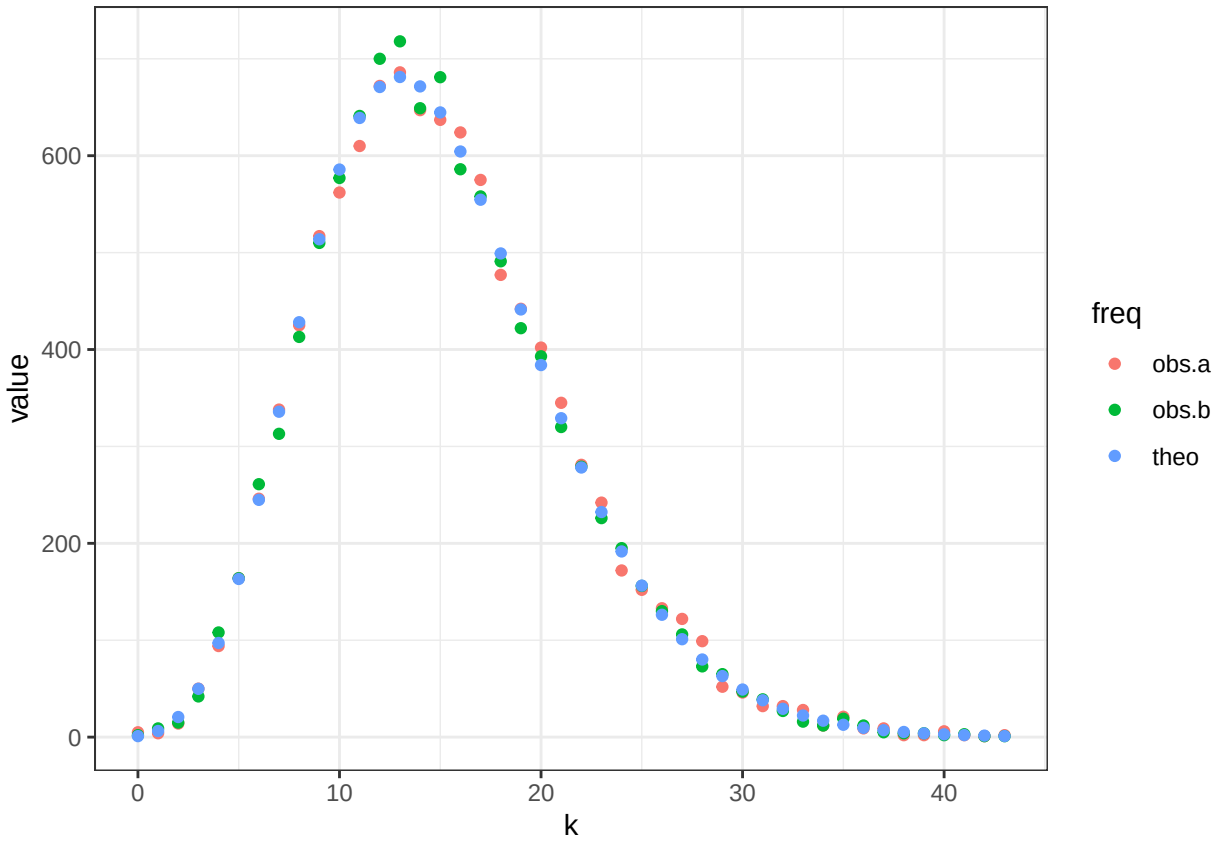
```
negbinModela <- inlineModel("
[LONGITUDINAL]
input =  {n,p}
DEFINITION:
y = {distribution=negativeBinomial, size=n, prob=p}
")

param <- c(n=10,p=0.4)
N <- 10000
y <- list(name='y', time=seq(1, N, by=1))

res2a <- simulx(model=negbinModela,
                parameter=param,
                output=y)
```

We can use equivalently define the probability mass function in the Mlxtran code:

```
negbinModelb <- inlineModel("
[LONGITUDINAL]
input =  {n,p}
DEFINITION:
y = {type   = count,
     P(y=k) = factorial(k+n-1)/factorial(k)/factorial(n-1)*((1-p)^k)*(p^n)
}
")

res2b <- simulx(model=negbinModelb,
                parameter=param,
                output=y)
```

We can now compare the empirical distribution of these 2 samples with the theoretical negative binomial distribution:

```
library(plyr)
library(reshape2)

ca=count(res2a$y$y)
cb=count(res2b$y$y)
names(ca) <- c('k','obs.a')
names(cb) <- c('k','obs.b')
c <- merge(ca, cb)
c$theo <- dnbinom(c$k,size=param[1],p=param[2])*N
r <- melt(c,  id = 'k', variable.name = 'freq')
print(ggplot(r, aes(k,value, colour=freq)) + geom_point())
```


value
600
400
200
0
0
10
20
30
40
k
freq
obs.a
obs.b
theo

Simulation of time-to-event data

R script: survival.R

Mlxtran code: model/survival1.txt ; model/survival2.txt

Introduction

Here, observations are the "times at which events occur.'' An event may be one-off (e.g., death, hardware failure) or repeated (e.g., epileptic seizures, mechanical incidents, strikes).

To begin with, we will consider a one-off event. The *survival function* $S(t)$ gives the probability that the event happens after time $t > t_{\text{start}}$:

$$S(t) \overset{\text{def}}{=} \mathbb{P}(T > t).$$

The *hazard function* $h(t)$ is defined for individual i as the instantaneous rate of the event at time t, given that the event has not already occurred:

$$h(t) \overset{\text{def}}{=} \lim_{dt \to 0} \frac{S(t) - S(t+dt)}{S(t)\,dt}.$$

This is equivalent to

$$h(t) = -\frac{d}{dt} \log S(t).$$

Depending on the application, the length of time to this event may be called the *survival* time or the *failure* time. In general, we simply say *time-to-event*.

The random variable representing the time-to-event is typically written T. There are then several ways to define observations, depending on the situation:

- The event time is exactly observed at time t. Here, observation is "$T = t$''.

The distribution of time to event data is defined in the block `DEFINITION`of the Section `[LONGITUDINAL]` by the hazard function h previously defined in the block `EQUATION`. This is the Mlxtran code for a single exactly observed event:

- We may know the event has happened in an interval I of length L but not know the exact time. This is *interval censoring* and observation is the event: "$T \in I$''.
- If we assume that the trial ends at time t_{stop}, the event may happen after the end. This is *right censoring*. Here, observation is the event: "$T > t_{\text{stop}}$''.

Remark: it is possible to combine interval censoring and right censoring.

Let us now consider repeated events. For any given hazard function h, the survival function S now represents the survival since the previous event at t_{j-1}, given here in terms of the cumulative hazard from t_{j-1} to t_j:

$$S(t_j|t_{j-1}) = \mathbb{P}(T_j > t_j \,|\, T_{j-1} = t_{j-1}) \tag{29}$$

$$= \exp\left(-\int_{t_{j-1}}^{t_j} h(u)\,du\right) \tag{30}$$

Taking into account censoring for repeated events is slightly different from one-off events: let $(b_0, b_1], (b_1, b_2], \ldots, (b_{K-1}, b_K]$ be a sequence of successive intervals with

$$t_{\text{start}} = b_0 < b_1 < b_2 < \ldots < b_K = t_{\text{stop}}.$$

We do not know the exact event times, but a sequence $(m_k;\, 1 \leq k \leq K)\}$ is observed where m_k is the number of events that occurred in interval $(b_{k-1}, b_k]$.

The Mlxtran code for repeated events is the same as for the single case, without the statement maxEventNumber = 1 if the number of events is not bounded, or with maxEventNumber = M if the maximum number of possible events is M.

Remark: if the number of events is not bounded, it is mandatory to define a right censoring time for simulation.

Simulation of a single event

We consider a Weibull model for a single event in this first example:

$$h(t) = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1}, \tag{31}$$

$$S(t) = e^{-(t/\lambda)^k} \tag{32}$$

The study stops at time 60: there is therefore a right censoring time equal to 60.

This model is implemented in the file `survival1.txt`:

We will use simulx for computing h every hour between 0h and 60h and for generating a single event starting at time 0, with $\beta = 1.5$ and $\lambda = 20$. It is mandatory to state explicitly when the experiment starts (at time $t = 0$ in this example), defining `time=0` for the output `e`.

```
p <- c(beta = 1.5, lambda=20)
h <- list(name='h', time=seq(0, 60, by=0.2))
e <- list(name='e', time=0)

res <- simulx(model     = 'model/survival1.txt',
              settings  = list(seed=123),
              parameter = p,
              output    = list(h, e))
```

Information about the simulated event is stored in the data frame `res$e`

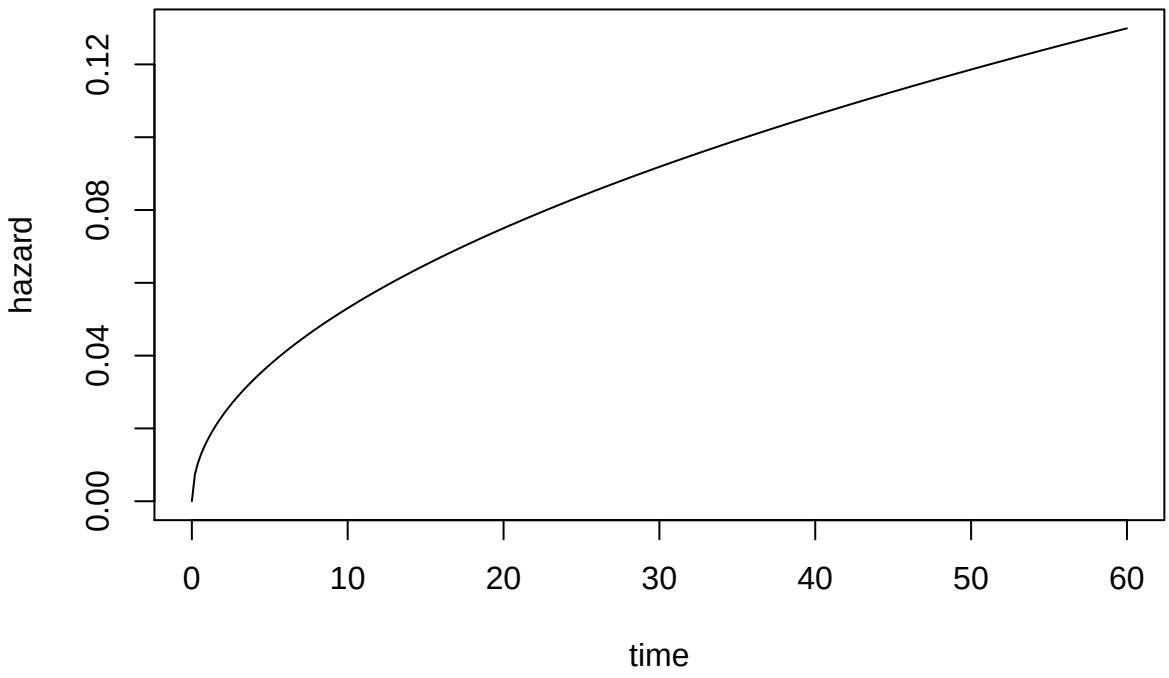
```
print(res$e)
```

```
##       time e
## 1  0.00000 0
## 2 28.81942 1
```

The first line means that the experiment started at time 0 (i.e., the event happened after $t = 0$), the second line means that the event was observed at time $t = 28.82$.

The hazard function is stored in the data frame `res$h`

```
hazard  <- res$h
plot(x=hazard$time, y=hazard$h, type='l', xlab="time", ylab="hazard")
```



We can use the same model for generating 100 single events. An additional argument `group` is then necessary, with the number of subjects defined in the field `size`.

```
N <- 100
res <- simulx(model     = 'model/survival1.txt',
              settings  = list(seed=1234),
              parameter = p,
              output    = list(h, e),
              group     = list(size = N))
```

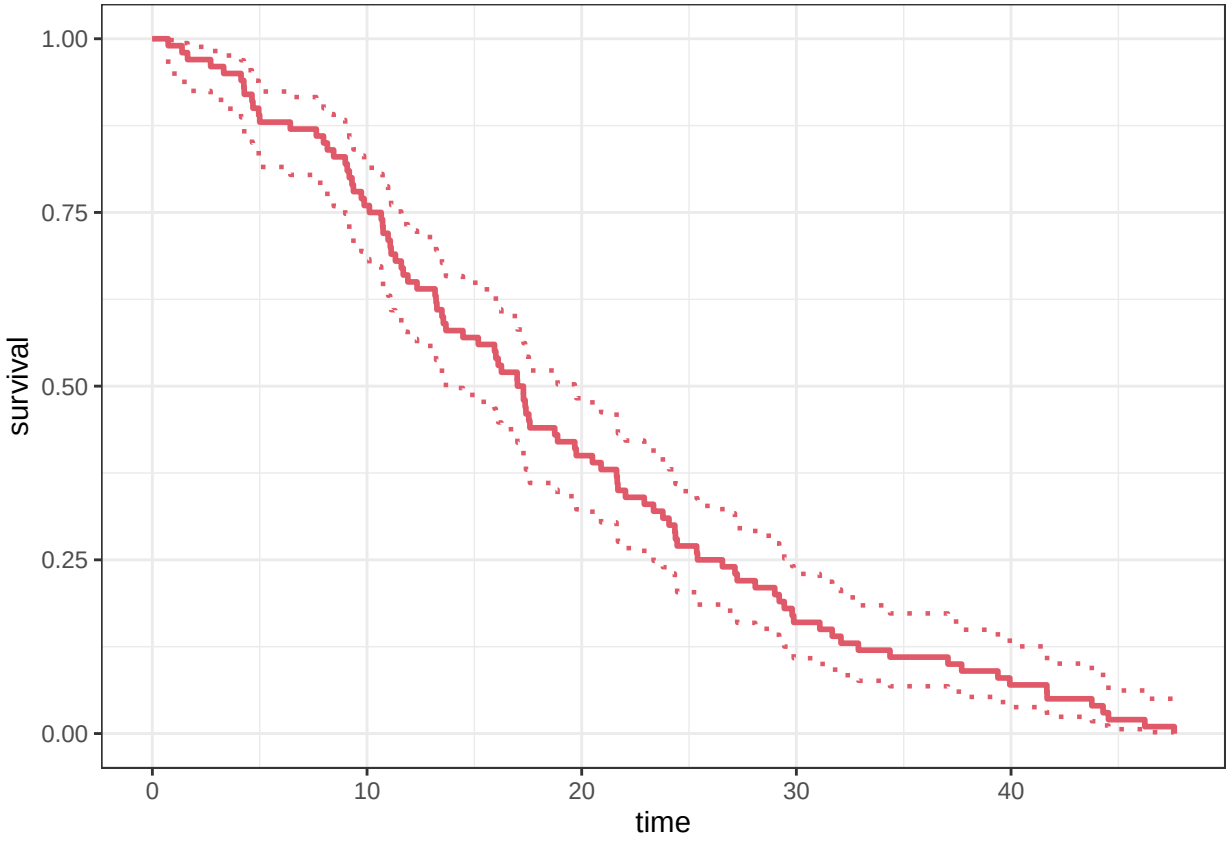
Some events are exactly observed (e=1) some others are right censored (time=60 and e=0)

```
print(res$e[1:10,])
```

```
##    id      time e
## 1   1  0.000000 0
## 2   1 29.435794 1
## 3   2  0.000000 0
## 4   2 10.723412 1
## 5   3  0.000000 0
## 6   3  8.983183 1
## 7   4  0.000000 0
## 8   4  4.954004 1
## 9   5  0.000000 0
## 10  5  4.683973 1
```

We can now use kmplotmlx for computing and plotting the Kaplan Meier estimate of the survival function with a 90% confidence interval:

```
pl1  <- kmplotmlx(res$e, level=0.9)
print(pl1)
```



Simulation of repeated events

Assume now that we have repeated events which are interval censored. The length of the intervals is 5.

This new model is implemented in the file `survival2.txt`:

We can use simulx with this new model:

```
res <- simulx(model     = 'model/survival2.txt',
              settings  = list(seed=123),
              parameter = p,
              output    = list(e))
```

`res$e` contains now the number events in each interval of length 5 between time 0 and time 60.

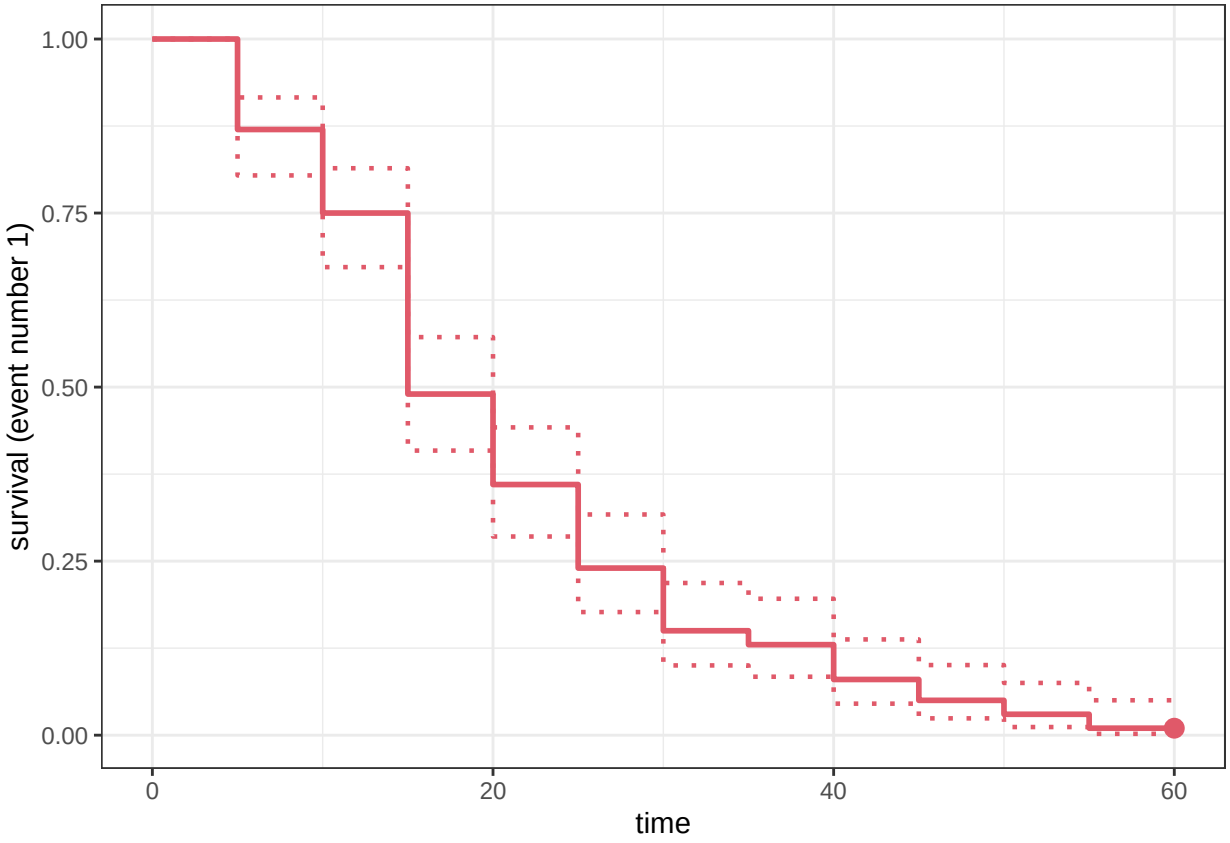
```
print(res$e)
```

```
##    time e
## 1     0 0
## 2     5 0
## 3    10 0
## 4    15 0
## 5    20 0
## 6    25 0
## 7    30 1
```

```
## 8    35 0
## 9    40 0
## 10   45 0
## 11   50 0
## 12   55 0
## 13   60 1
```

We can, as previously, simulate 100 replicates of the same experiment and display the Kaplan Meier plot:

```
res <- simulx(model     = 'model/survival2.txt',
              settings  = list(seed=123),
              parameter = p,
              output    = list(e),
              group     = list(size=N))


pl2  <- kmplotmlx(res$e, level=0.9)
print(pl2)
```



Defining individual designs

Individual right censoring times

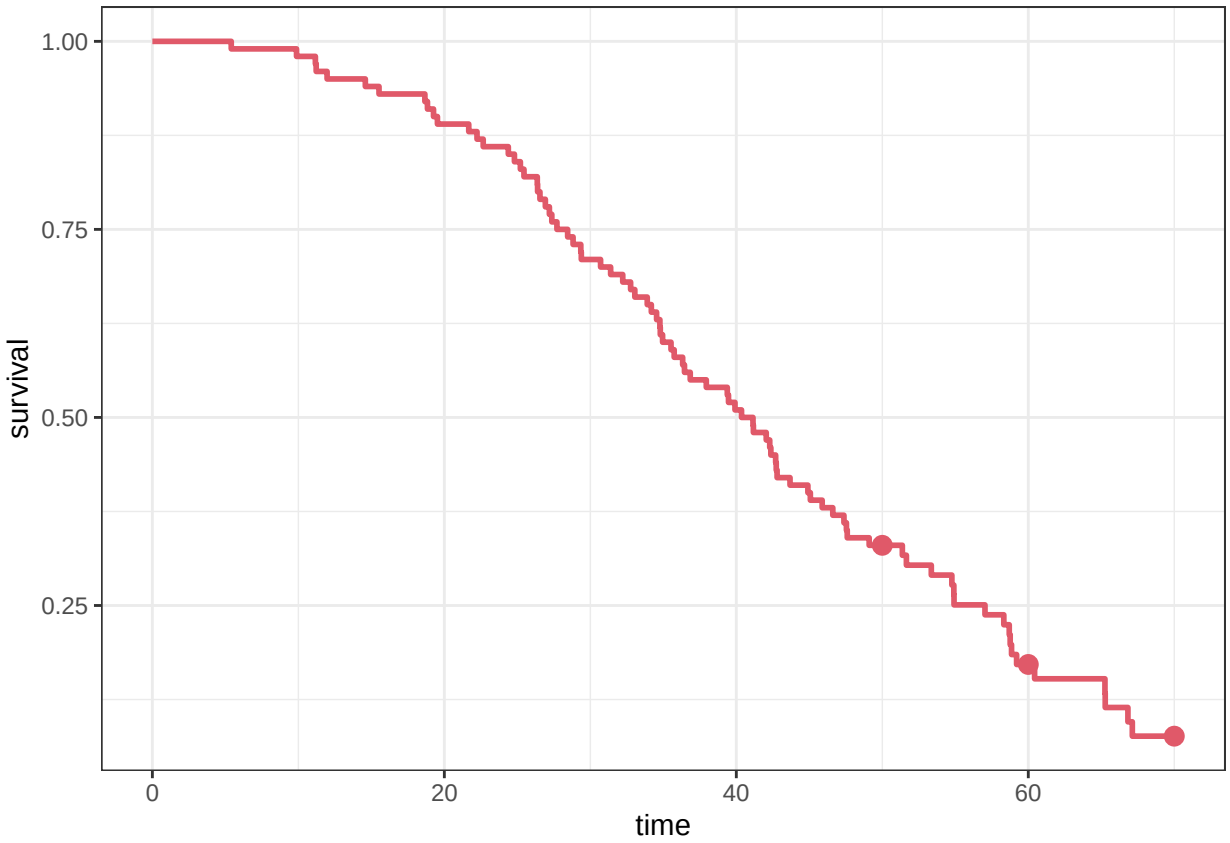
The right censoring time can be defined per individual

```
tteModel1 <- inlineModel("
[LONGITUDINAL]
input = {beta, lambda, rct}
```

```
EQUATION:
h=(beta/lambda)*(t/lambda)^(beta-1)

DEFINITION:
e = {type=event, maxEventNumber=1, rightCensoringTime=rct, hazard=h}
")

N <- 100
p1   <- c(beta=2.5,lambda=50)
rct  <- data.frame(id=1:N, rct=c(rep(50,N/4),rep(60,N/4),rep(70,N/2)))
e    <- list(name='e', time=0)
res1a <- simulx(model     = tteModel1,
                parameter = list(p1, rct),
                output    = e,
                settings  = list(seed=12345))
print(res1a$e[c(1:4,69:72,173:176),])
```

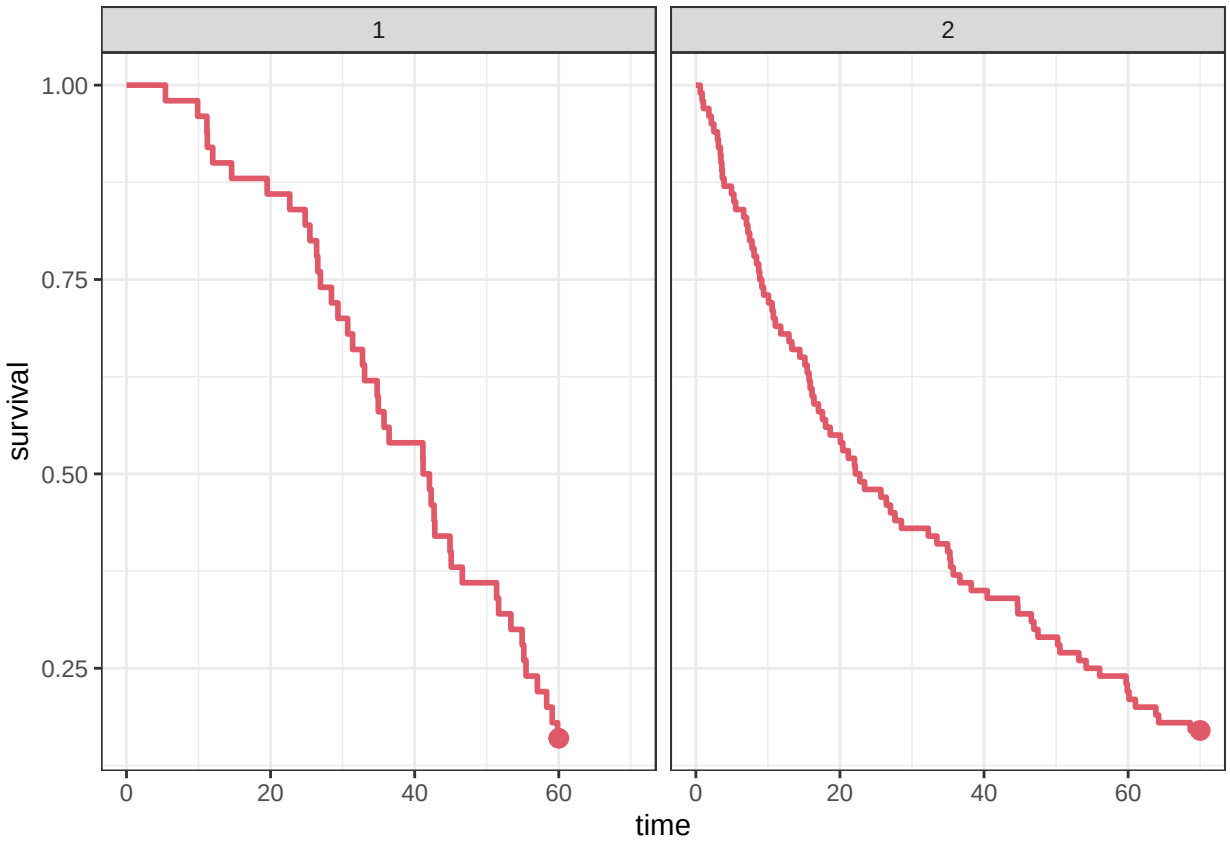
```
##     id      time e
## 1    1  0.000000 0
## 2    1 50.000000 0
## 3    2  0.000000 0
## 4    2  9.873987 1
## 69  35  0.000000 0
## 70  35 36.448000 1
## 71  36  0.000000 0
## 72  36 60.000000 0
## 173 87  0.000000 0
## 174 87 70.000000 0
## 175 88  0.000000 0
## 176 88 39.908235 1
```

```
kmplotmlx(res1a$e)
```



or per group

```
g1   <- list(size=50,  parameter=c(beta=2.5,lambda=50, rct=60))
g2   <- list(size=100, parameter=c(beta=1,lambda=40, rct=70))
res1b <- simulx(model     = tteModel1,
                output    = e,
                group     = list(g1,g2),
                settings = list(seed=12345))
kmplotmlx(res1b$e)
```



Maximum number of events

```
tteModel1b <- inlineModel("
[LONGITUDINAL]
input = {beta, lambda, men}

EQUATION:
h=(beta/lambda)*(t/lambda)^(beta-1)

DEFINITION:
e = {type=event, maxEventNumber=men, rightCensoringTime=100, hazard=h}
")

N <- 3
p1   <- c(beta=2.5,lambda=50)
men  <- data.frame(id=1:3, men=c(1,2,3)) ## individual maximum number of events
e    <- list(name='e', time=0)
res1a <- simulx(model     = tteModel1b,
                parameter = list(p1, men),
                output    = e,
                settings  = list(seed=12345))
print(res1a$e)
```

```
##   id       time e
## 1  1   0.000000 0
## 2  1  59.090342 1
```

```
## 3  2   0.000000 0
## 4  2   9.873994 1
## 5  2  53.763458 1
## 6  3   0.000000 0
## 7  3  55.145929 1
## 8  3  99.157858 1
## 9  3 100.000000 0
```

Simulation of a joint model

R scripts: joint1.R ; joint2.R

Introduction

A *joint model* is one that allows us to simultaneously describe the distribution of different types of observations made on the same individual.

Suppose that we have L different types of observation: $y^{(1)} = (y_j^{(1)}, 1 \leq j \leq n_1)$, $y^{(2)} = (y_j^{(2)}, 1 \leq j \leq n_2)$, $\ldots$, $y^{(L)} = (y_j^{(L)}, 1 \leq j \leq n_L)$, where n_ℓ is the number of observations of type ℓ.

Note that the numbers of observations (n_ℓ) and the observation times $(t_j^{(\ell)})$ may be different.

The joint model is the joint distribution of $y = (y^{(1)}, y^{(2)}, \ldots, y^{(L)})$.

This joint distribution is defined in the block `DEFINITION`of the Section `[LONGITUDINAL]`.

Examples

Joint continuous PKPD model

We consider in this example a joint model for continuous PK and PD data. We assume here that the predicted effect E is function of the predicted concentration C:

$$E(t) = k_{\text{in}}/k_{\text{out}} \quad \text{for } t \leq 0 \tag{33}$$

$$\dot{E}(t) = k_{\text{in}}\left(1 - \frac{C(t)}{IC_{50} + C(t)}\right) - k_{\text{out}}E(t) \quad \text{for } t \geq 0 \tag{34}$$

We then assume that the measured concentrations $(y_j^{(1)})$ are log-normally distributed and the measured effects $(y_j^{(2)})$ normally distributed:

$$\log(y_j^{(1)}) = \log(C(t_j^{(1)})) + a_1\varepsilon_j^{(1)}, \quad ; \quad 1 \leq j \leq n_1 \tag{35}$$

$$y_j^{(2)} = f_2(t_j^{(2)}) + a_2\varepsilon_j^{(2)}, \quad ; \quad 1 \leq j \leq n_2 \tag{36}$$

This model is implemented in `joint1.R`:

```
joint.model1 <- inlineModel("
[LONGITUDINAL]
input = {ka, V, Cl, IC50, kin, kout, a1, a2}

EQUATION:
C     = pkmodel(ka, V, Cl)
t0    = 0
E_0   = kin/kout
ddt_E = kin*(1 - C/(IC50+C)) - kout*E

DEFINITION:
Concentration = {distribution = lognormal,
```

```
                prediction   = C,
                sd           = a1}

Effect = {distribution = normal,
          prediction   = E,
          sd           = a2}
")
```

We use simulx for computing C and E every hour between 0h and 100h and for generating measured concentrations every 12 hours between time 4h and 100h and measured effects every 10 hours between time 5h and 100h.

A dose of 10 mg is administrated orally every 12 hours, starting at time 2h.

```
p <- c(ka=0.5, V=8, Cl=1.5, IC50=0.5, kin=10, kout=0.1, a1=0.1, a2=5)

a <- list(amount = 10, time = seq(2,120,by=12))

f <- list(name = c('C', 'E'),     time = seq(0,100,by=1))
c <- list(name = 'Concentration', time = seq(4,100,by=12))
e <- list(name = 'Effect',        time = seq(5,100,by=10))

res <- simulx(model     = 'model/joint1.txt',
              treatment = a,
              parameter = p,
              output    = list(f, c, e),
              settings  = list(seed = 1234))
```

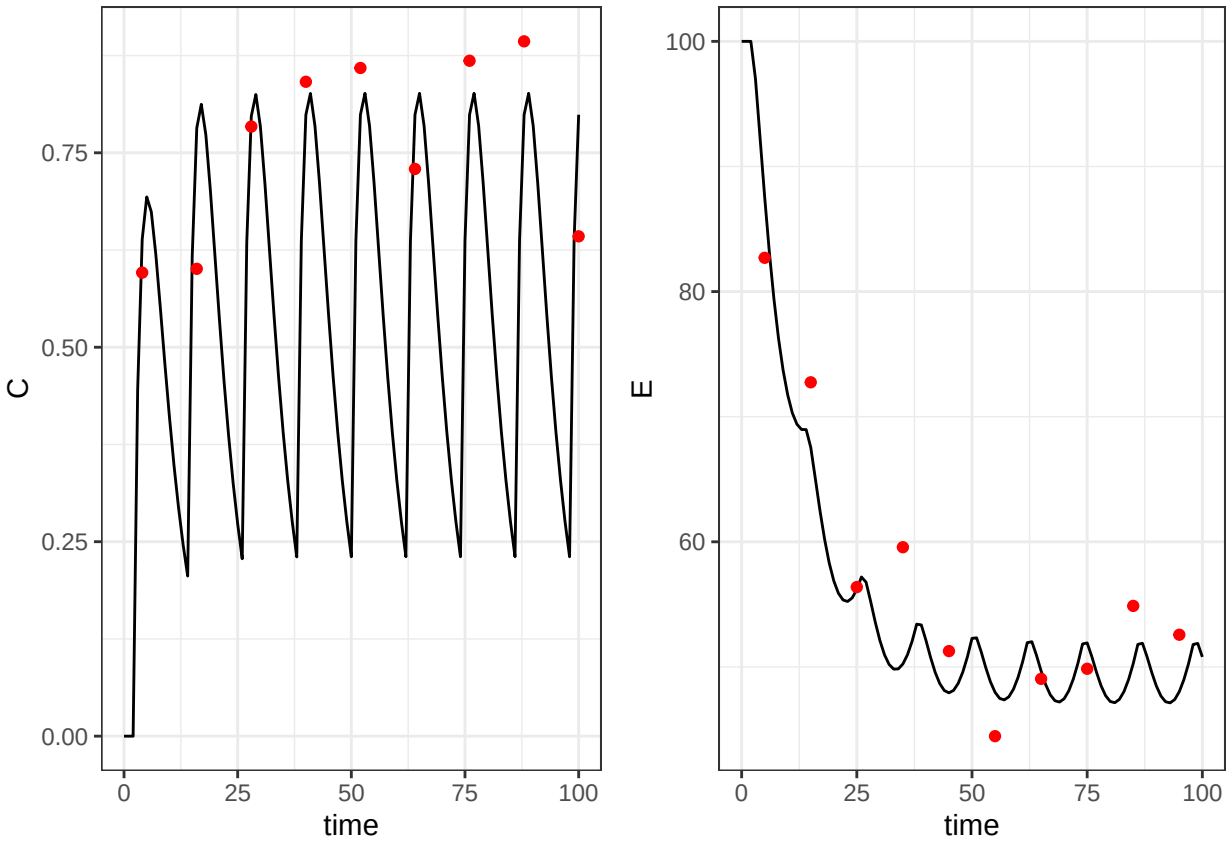
Here, res is a list with four elements:

```
names(res)
```

```
## [1] "C"             "E"             "Concentration" "Effect"
## [5] "treatment"
```

Let us plot this data:

```
library("gridExtra")
plot1 = ggplot() + geom_line(data=res$C, aes(x=time, y=C)) +
        geom_point(data=res$Concentration, aes(x=time, y=Concentration),colour="red")
plot2 = ggplot() + geom_line(data=res$E, aes(x=time, y=E)) +
        geom_point(data=res$Effect, aes(x=time, y=Effect),colour="red")
grid.arrange(plot1, plot2, ncol=2)
```



Joint PK and time-to-event data model

We consider now a joint model for continuous PK data and repeated events. The hazard h is function of the predicted concentration C:

$$h(t) = u\, e^{v\, C(t)}$$

We also assume that the events are exactly observed until time 100h (right censoring time) This model is implemented in `joint2.R`:

```
joint.model2 <- inlineModel("
[LONGITUDINAL]
input = {ka, V, Cl, u, v, a1}

EQUATION:
C = pkmodel(ka, V, Cl)
h = u*exp(v*C)

DEFINITION:
Concentration = {distribution = lognormal,
                 prediction   = C,
                 sd           = a1}

Hemorrhaging  = {type               = event,
                 rightCensoringTime = 100,
                 hazard             = h}
")
```

We use `simulx` for computing C and h every hour between 0h and 100h and for generating measured concentrations every 12 hours between time 4h and 100h and repeated events, starting at time 0.

```
p <- c(ka=0.5, V=8, Cl=1.5, u=0.003, v=3, a1=0.05)

a <- list(amount = 10, time = seq(2,120,by=12))

f <- list(name = c('C', 'h'),     time = seq(0,100,by=1))
c <- list(name = 'Concentration', time = seq(4,100,by=12))
e <- list(name = 'Hemorrhaging',  time = 0)

res1 <- simulx(model     = joint.model2,
               treatment = a,
               parameter = p,
               output    = list(f, c, e),
               settings  = list(seed = 12345))
```

Let us see the simulated events:

```
print(res1$Hemorrhaging)
```

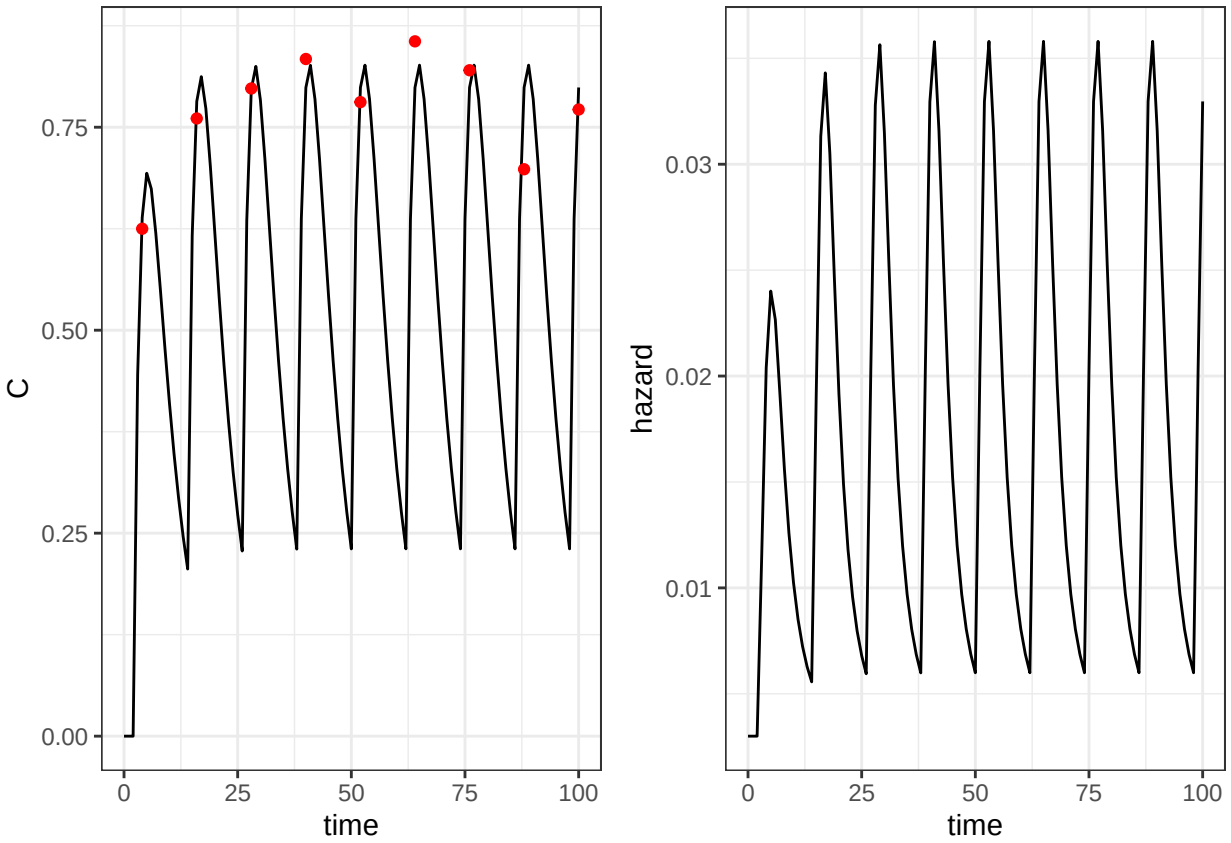
```
##        time Hemorrhaging
## 1   0.00000            0
## 2  86.95924            1
## 3  99.83272            1
## 4 100.00000            0
```

Indeed, in this example,

- the experiment starts at time 0, which means that the first event will occur after time 0
- there is one observed event at time 86.96h
- there is one observed event at time 99.83h
- the next event will occur after time 100

We can also plot the predicted and observed concentrations and the hazard function

```
plot1 = ggplot() + geom_line(data=res1$C, aes(x=time, y=C)) +
        geom_point(data=res1$Concentration, aes(x=time, y=Concentration),colour="red")
plot2 = ggplot() + geom_line(data=res1$h, aes(x=time, y=h)) +
        ylab("hazard")  + theme(legend.position="none")
grid.arrange(plot1, plot2, ncol=2)
```

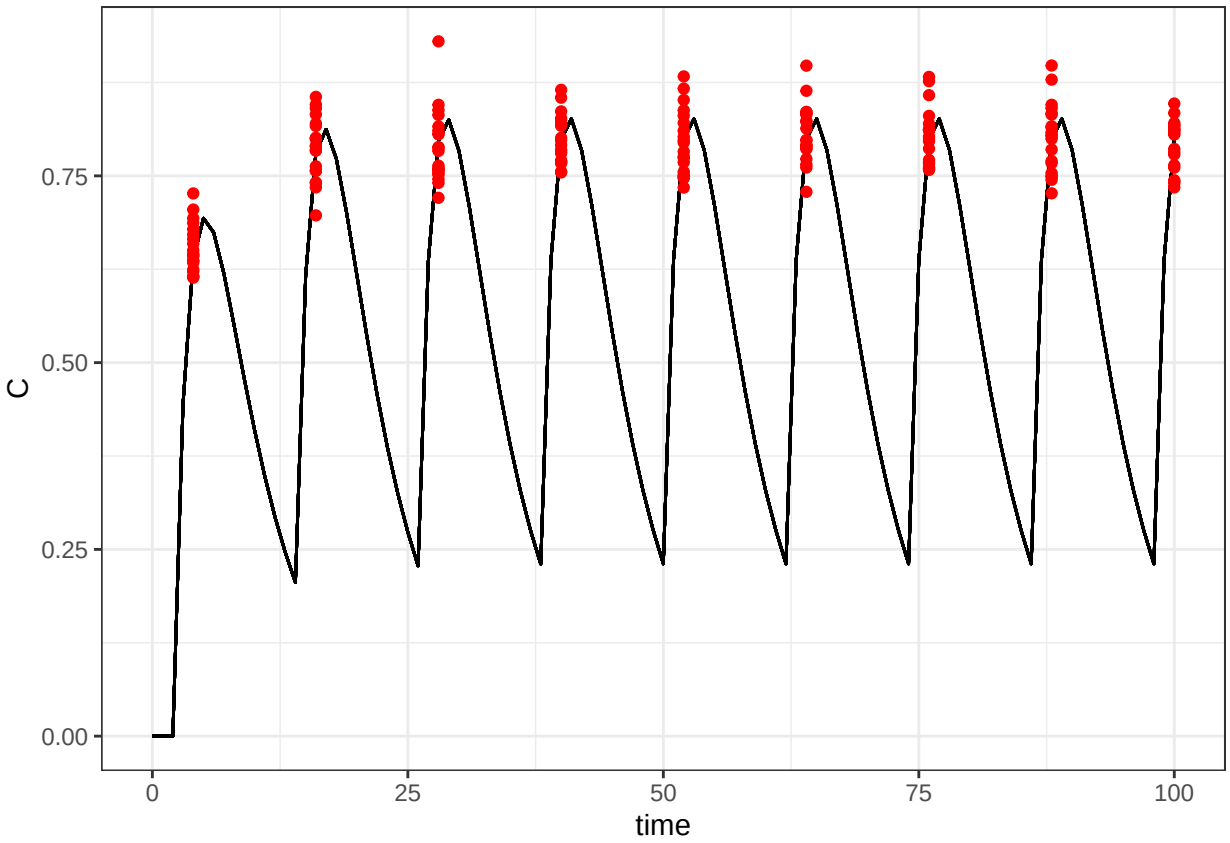


Instead of simulating data for a single individual, we can simulate data for $N = 20$ individuals with the same model, using the input argument `group`

```
N   <- 20
res2 <- simulx(model     = joint.model2,
               treatment = a,
               parameter = p,
               output    = list(f, c, e),
               group     = list(size = N, level='longitudinal'),
               settings  = list(seed = 121212))
```

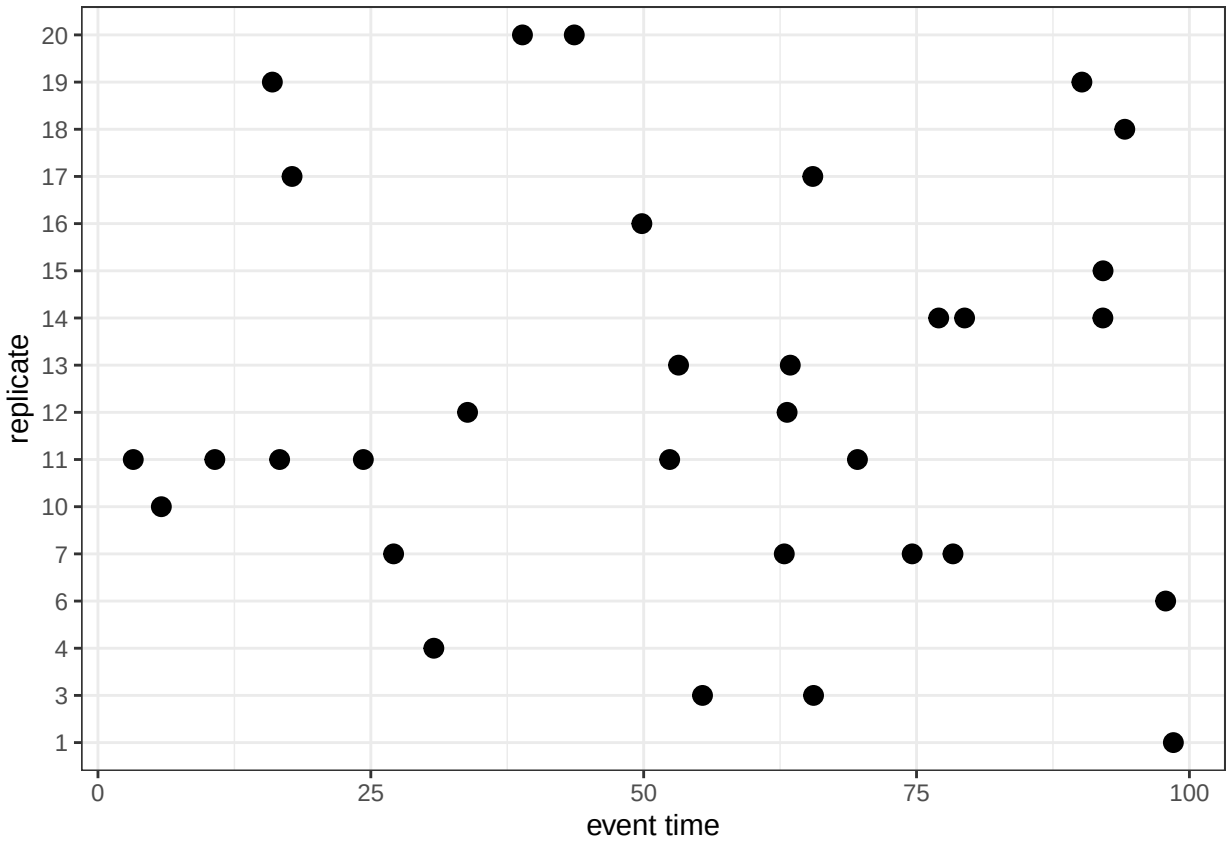
We can plot the predicted and observed concentrations

```
plot3 = ggplot() + geom_line(data=res2$C, aes(x=time, y=C, group=id)) +
  geom_point(data=res2$Concentration, aes(x=time, y=Concentration),colour="red")
plot(plot3)
```



We can also plot the observed events for the 20 individuals:

```
h1    = res2$Hemorrhaging[res2$Hemorrhaging[,3]==1,]
plot4 = ggplot()+geom_point(data=h1, aes(x=time,y=id), size=3) +
        xlab("event time") + ylab("replicate")  + theme(legend.position="none")
plot(plot4)
```


replicate
20
19
18
17
16
15
14
13
12
11
10
7
6
4
3
1
0
25
50
75
100
event time

Censored data (data below/above a limit of quantification)

R script: censored.R

Introduction

Censoring occurs when the value of a measurement or observation is only partially known. In the longitudinal context, censoring refers to the values of the measurements, not the times at which they were taken.

For example, the lower limit of quantification (LLOQ) is the lowest quantity of a substance that can be reliably measured. Therefore, any time the quantity is below the LLOQ, the "observation'' is not a measurement but the information that the measured quantity is less than the LLOQ.

A measuring device can also have an upper limit of quantification (ULOQ) such that any value above this limit cannot be measured and reported.

As hinted above, censored values are not typically reported as a number, but their existence is known, as well as the type of censoring. Thus, the observation $y_{ij}^{(r)}$ (i.e., what is reported) is the measurement y_{ij} if not censored, and the type of censoring otherwise. We usually distinguish between three types of censoring: left, right and interval.

Example

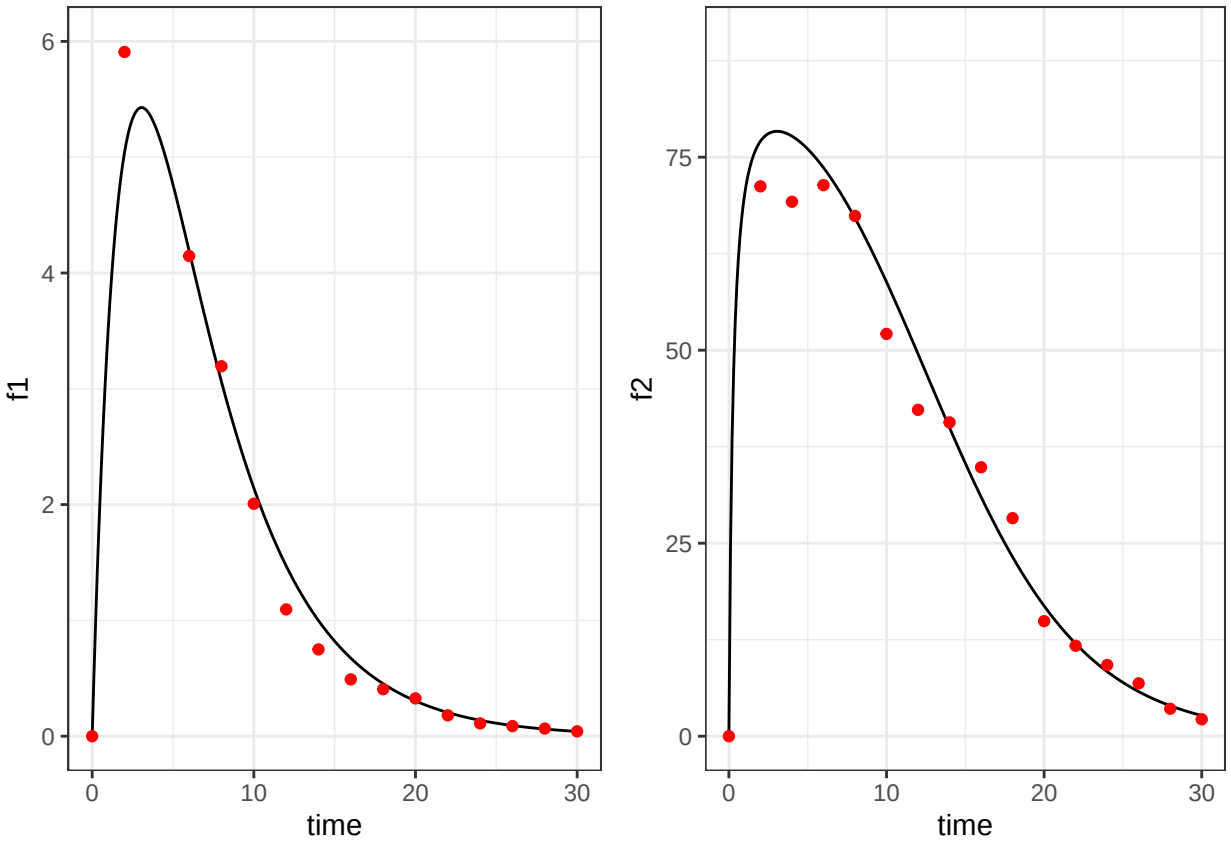
We use here a basic model for continuous PKPD data:

```
myModel <- inlineModel("
[LONGITUDINAL]
input = {ka, V, k, Emax, EC50, b1, b2}

EQUATION:
D = 100
f1 = D*ka/(V*(ka-k))*(exp(-k*t) - exp(-ka*t))
f2 = Emax*f1/(f1 + EC50)
g1 = b1*f1
g2 = b2*f2

DEFINITION:
y1 = {distribution=normal, prediction=f1, sd=g1}
y2 = {distribution=normal, prediction=f2, sd=g2}
")
```

We will start evaluating the predictions f_1 and f_2 and simulating 2 sequences of observations y_1 and y_2:

```
f <- list(name= c('f1','f2'), time=seq(0, 30, by=0.1))
y <- list(name=c('y1','y2'), time=seq(0, 30, by=2))
p <- c(ka=0.5, V=10, k=0.2, Emax=100, EC50=1.5, b1=0.2, b2=0.15)
s <- 123456

res1a <- simulx(model    = myModel,
                parameter = p,
                output    = list(f, y),
                settings  = list(seed=s) )

library(gridExtra)
```

```
pla1 <- ggplot() + geom_line(data=res1a$f1, aes(x=time, y=f1), size=0.5) +
  geom_point(data=res1a$y1, aes(x=time, y=y1), colour="red") + ylim(c(0, 6))
pla2 <- ggplot() + geom_line(data=res1a$f2, aes(x=time, y=f2), size=0.5) +
  geom_point(data=res1a$y2, aes(x=time, y=y2), colour="red") + ylim(c(0, 90))
grid.arrange(pla1, pla2, ncol=2)
```



We now introduce a lower limit of quantification LLOQ=1.8 for the PK data and an upper limit for the PD data ULOQ=60:

```
y1 <- list(name='y1', time=seq(0, 30, by=2), lloq=0.8)
y2 <- list(name='y2', time=seq(0, 30, by=2), uloq=60)

res1b <- simulx(model    = myModel,
                parameter = p,
                output    = list(y1, y2),
                settings  = list(seed=s) )
```

`simulx` creates a column CENS that takes the value 0 when the data is not censored, 1 when the data is left censored (BLQ data) and -1 if the data is right censored (ALQ data).

```
head(res1b$y1)
```

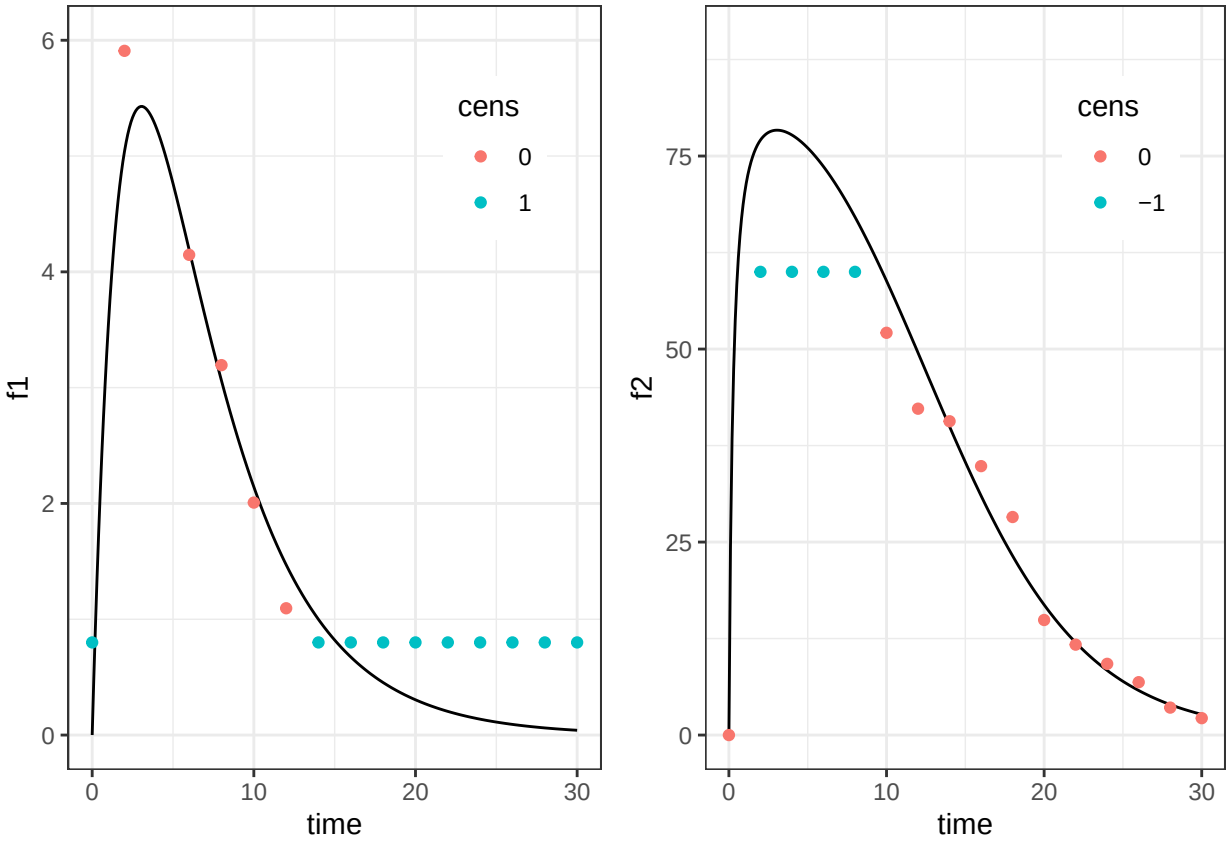
```
##   time       y1 cens
## 1    0 0.800000    1
## 2    2 5.908515    0
## 3    4 6.610188    0
## 4    6 4.146683    0
```

```
## 5    8 3.194614    0
## 6   10 2.007575    0
```

```
head(res1b$y2)
```

```
##   time       y2 cens
## 1    0  0.00000    0
## 2    2 60.00000   -1
## 3    4 60.00000   -1
## 4    6 60.00000   -1
## 5    8 60.00000   -1
## 6   10 52.10554    0
```

```
plb1 <- ggplot() + geom_line(data=res1a$f1, aes(x=time, y=f1), size=0.5) +
  geom_point(data=res1b$y1, aes(x=time, y=y1, colour=cens)) +
  theme(legend.position=c(.8, .8)) + ylim(c(0, 6))
plb2 <- ggplot() + geom_line(data=res1a$f2, aes(x=time, y=f2), size=0.5) +
  geom_point(data=res1b$y2, aes(x=time, y=y2, colour=cens))+
  theme(legend.position=c(.8, .8)) + ylim(c(0, 90))
grid.arrange(plb1, plb2, ncol=2)
```



In addition, we can take into account the fact that a concentration only takes positive values. In other word, a concentration below the limit of quantification (BLQ data) is known to be between 0 and LLOQ. The lower limit of the censoring interval is then defined as LIMIT=0.

```
y1$limit=0

res1b <- simulx(model    = myModel,
```

```
                parameter = p,
                output    = list(y1, y2),
                settings  = list(seed=s) )
```

An additional column LIMIT is then created:

```
head(res1b$y1)
```

```
##   time       y1 cens limit
## 1    0 0.800000    1     0
## 2    2 5.908515    0     0
## 3    4 6.610188    0     0
## 4    6 4.146683    0     0
## 5    8 3.194614    0     0
## 6   10 2.007575    0     0
```

The simulated data can be saved in a single file using the `Monolix` format (see the Monolix documentation for more information).

```
writeDatamlx(res1b, result.file = "res1b.csv")
head(read.table("res1b.csv", header=T, sep=","), n=20)
```

```
##    time        y cens limit ytype
## 1     0  0.80000    1     0     1
## 2     0  0.00000    0     .     2
## 3     2  5.90852    0     .     1
## 4     2 60.00000   -1     .     2
## 5     4  6.61019    0     .     1
## 6     4 60.00000   -1     .     2
## 7     6  4.14668    0     .     1
## 8     6 60.00000   -1     .     2
## 9     8  3.19461    0     .     1
## 10    8 60.00000   -1     .     2
## 11   10  2.00757    0     .     1
## 12   10 52.10554    0     .     2
## 13   12  1.09479    0     .     1
## 14   12 42.27568    0     .     2
## 15   14  0.80000    1     0     1
## 16   14 40.64667    0     .     2
## 17   16  0.80000    1     0     1
## 18   16 34.83516    0     .     2
## 19   18  0.80000    1     0     1
## 20   18 28.24314    0     .     2
```

The file can also be created directly from `simulx`:

```
res1c <- simulx(model     = myModel,
                parameter = p,
                result.file   ="res1c.csv",
                output    = list(y1, y2),
                settings  = list(seed=s))

head(read.table("res1c.csv", header=T, sep=","))
```

```
##   time       y cens limit ytype
## 1    0  0.8000    1     0     1
## 2    0  0.0000    0     .     2
```

```
## 3    2  5.9085    0     .     1
## 4    2 60.0000   -1     .     2
## 5    4  6.6102    0     .     1
## 6    4 60.0000   -1     .     2
```

Simulation of hierarchical model: Introduction

The models we are interested with are mixed effects models (see also this web animation), i.e. hierarchical models that involves different types of variables:

- We call $y_i = (y_{ij}, 1 \leq j \leq n_i)$ the set of *longitudinal data* recorded at times $(t_{ij}, 1 \leq j \leq n_i)$ for subject i, and $\boldsymbol{y}$ the combined set of observations for all N individuals: $\boldsymbol{y} = (y_1, \ldots, y_N)$.
- We write ψ_i for the vector of *individual parameters* for individual i and $\boldsymbol{\psi}$ the set of individual parameters for all N individuals: $\boldsymbol{\psi} = (\psi_1, \ldots, \psi_N)$.
- The distribution of the individual parameters ψ_i of subject i may depend on a vector of *individual covariates* c_i. Then, let $\boldsymbol{c} = (c_1, c_2, \ldots, c_N)$.
- In a population approach context, we call θ the vector of *population parameters*.

Considering these variables as random variables, the joint probability distribution of $\boldsymbol{y}$, $\boldsymbol{\psi}$, $\boldsymbol{c}$ and θ can be decomposed into a product of conditional distributions:

$$\mathrm{p}(\boldsymbol{y}, \boldsymbol{\psi}, \boldsymbol{c}, \theta) = \mathrm{p}(\boldsymbol{y}|\boldsymbol{\psi}, \theta)\mathrm{p}(\boldsymbol{\psi}|\boldsymbol{c}, \theta)\mathrm{p}(\boldsymbol{c}|\theta)\mathrm{p}(\theta)$$

`Mlxtran` takes advantage of the hierarchical structure of this joint probability distribution by decomposing the joint model into several submodels. Then, each component of the model is implemented in a different section:

In each section, `Mlxtran` supports flexible equation-based descriptions implemented in a block `EQUATION` and explicit definition-based descriptions of probability distributions in a block `DEFINITION`.

Models for described in the section `[LONGITUDINAL]` include models for continuous data, categorical data, count data and survival data.

We will see how to define a model for the individual parameters in the section `[INDIVIDUAL]`, for the individual covariates in the section `[COVARIATES]` and for the population parameters in the section `[POPULATION]`.

Simulation of a hierarchical model: the individual parameters

---

R scripts: hierarchical1.R

Mlxtran codes: model/hierarchical1a.txt ; model/hierarchical1b.txt

---

Introduction

The population approach means that each individual is represented by the same basic parametric model but not necessarily the exact same parameter values. Thus, individual i has parameters ψ_i. If we consider that individuals are randomly selected from the population, then we can treat ψ_i as a random vector of parameters.

The probability distribution of ψ_i is a parametric distribution that depends on a vector θ of *population parameters* and possibly a set of *individual covariates* c_i. This dependence can be made clear by writing $\mathtt{p}(\psi_i\,;\theta, c_i)$ for the pdf of ψ_i.

In this context, the model for individual i is the joint distribution of the longitudinal data y_i and the individual parameters ψ_i:

$$\mathtt{p}(y_i, \psi_i; \theta, c_i) = \mathtt{p}(y_i|\psi_i)\,\mathtt{p}(\psi_i; \theta, c_i).$$

The model $\mathtt{p}(y_i|\psi_i)$ for the longitudinal data is implemented in the section `[LONGITUDINAL]` and the model $\mathtt{p}(\psi_i; \theta, c_i)$ for the individual parameters is implemented in the section `[INDIVIDUAL]`. Its inputs are the population parameters θ and the individual covariates c_i.

Examples

Example 1

Consider the following model for continuous data:

(1)
$$y_{ij} = \frac{100}{V_i} e^{-k\,t_{ij}} + a\,e_{ij}$$

where $e_{ij} \sim_{\text{i.i.d.}} \mathcal{N}(0,1)$.

Given V_i, each y_{ij} is therefore normally distributed:

(2)
$$y_{ij}|V_i \sim \mathcal{N}\left(\frac{100}{V_i} e^{-k\,t_{ij}}, a^2\right).$$

Here, V_i is an individual parameter which is assumed to be lognormally distributed:

(3)
$$\log(V_i) \underset{\text{i.i.d.}}{\sim} \mathcal{N}\left(\log\left(V_{\text{pop}}\right)\ ,\ \omega_V^2\right).$$

The other parameters of model (8) are $k_i = k$ and $a_i = a$. They are fixed in this example: we don't need to define their probability distributions.

The model therefore consists of the conditional distribution of the longitudinal data $(y_i j)$ defined in (8) and the distribution of the individual parameter V_i defined in (9).

This model is implemented in `hierarchical1a.txt`.

We will use simulx for sampling one individual parameter V_i and one sequence $y_i = (y_{ij})$ for one individual $i = 1$, with some given values of the population parameters V_{pop}, ω_V, k and a

```
p <- c(V_pop=10, omega_V=0.3, k=0.1, a=0.5)
f <- list(name='f', time=seq(0, 30, by=0.1))
y <- list(name='y', time=seq(1, 30, by=3))
V <- list(name='V')

res1 <- simulx(model     = 'model/hierarchical1a.txt',
               parameter = p,
               output    = list(V, f, y),
               settings  = list(seed = 12345))
```

Here, res1 is a list with three elements:

```
names(res1)
```

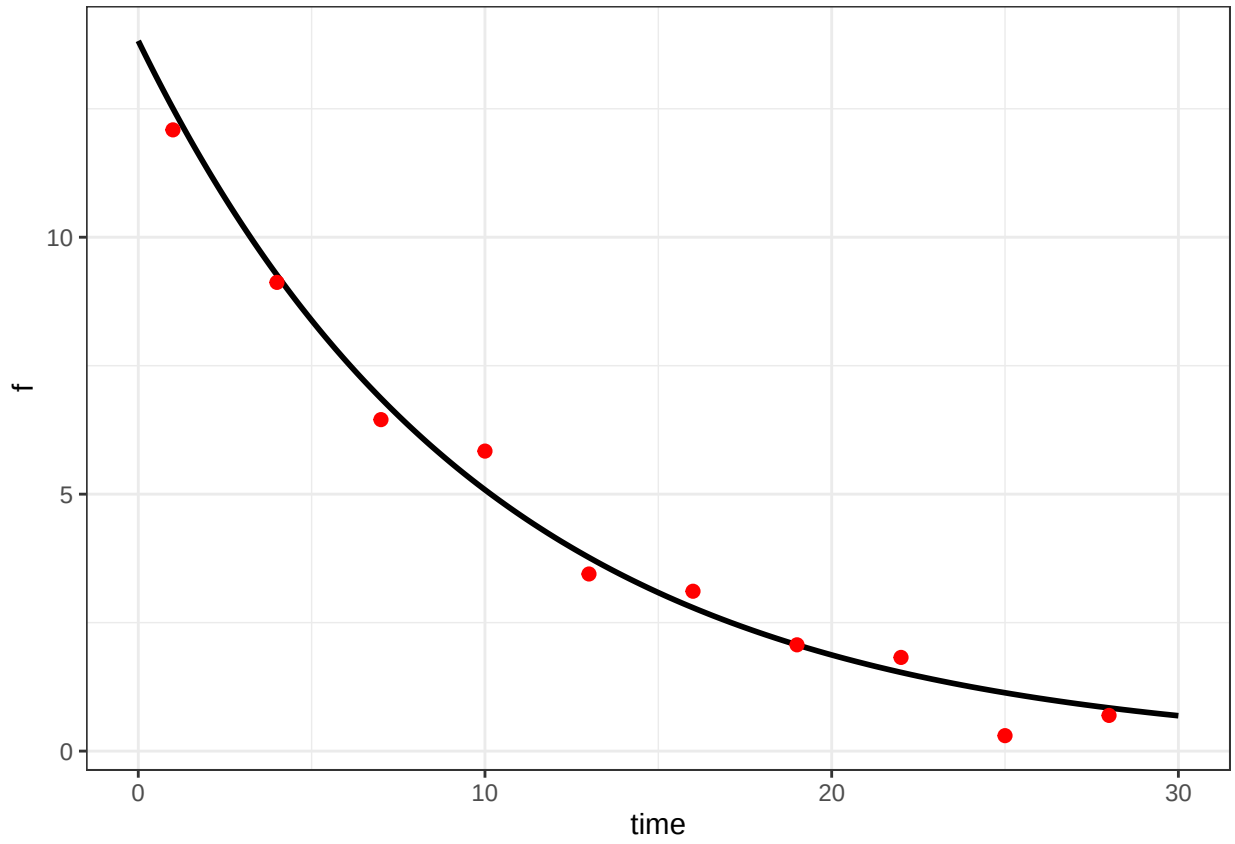
```
## [1] "f"         "y"         "parameter"
```

Let us see the simulated value of V_1 and plot the simulated data

```
print(res1$parameter)
```

```
##          V
## 1 7.234932
```

```
print(ggplot() +
        geom_line(data=res1$f, aes(x=time, y=f), size=1) +
        geom_point(data=res1$y, aes(x=time, y=y), colour='red', size=2))
```



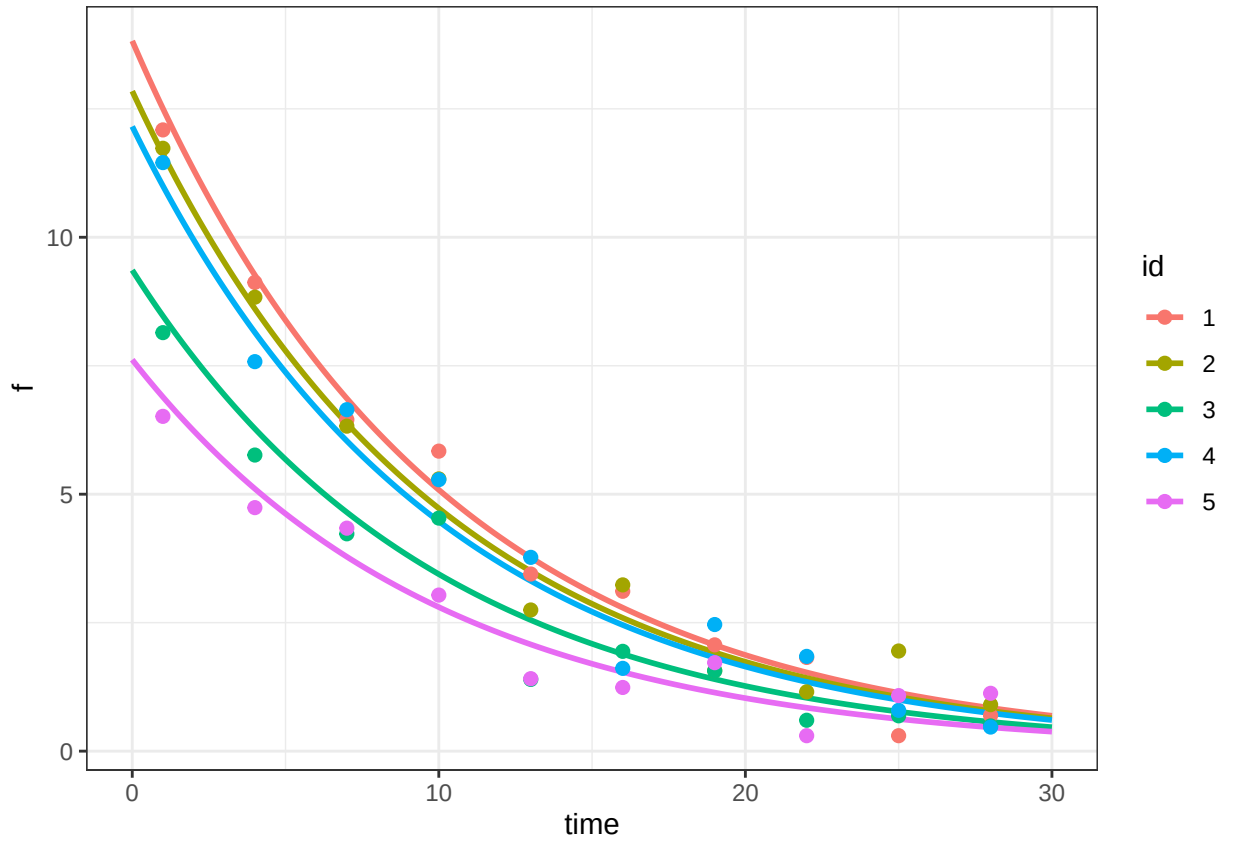
Instead of only one individual, let us now simulate a group of 5 individuals with the same model. An individual parameter V_i is drawn for each individual $i = 1, 2, \ldots, 5$. The level of randomization is therefore

'individual'.

```
g <- list( size  = 5,
           level = 'individual')

res2 <- simulx(model     = 'model/hierarchical1a.txt',
               parameter = p,
               output    = list(V, f, y),
               group     = g,
               settings  = list(seed = 12345))

print(res2$parameter)
```

```
##   id         V
## 1  1  7.234932
## 2  2  7.785818
## 3  3 10.683442
## 4  4  8.226188
## 5  5 13.128576
```

```
print(ggplot() +
        geom_line(data=res2$f, aes(x=time, y=f, colour=id),size=1) +
        geom_point(data=res2$y, aes(x=time, y=y, colour=id), size=2))
```

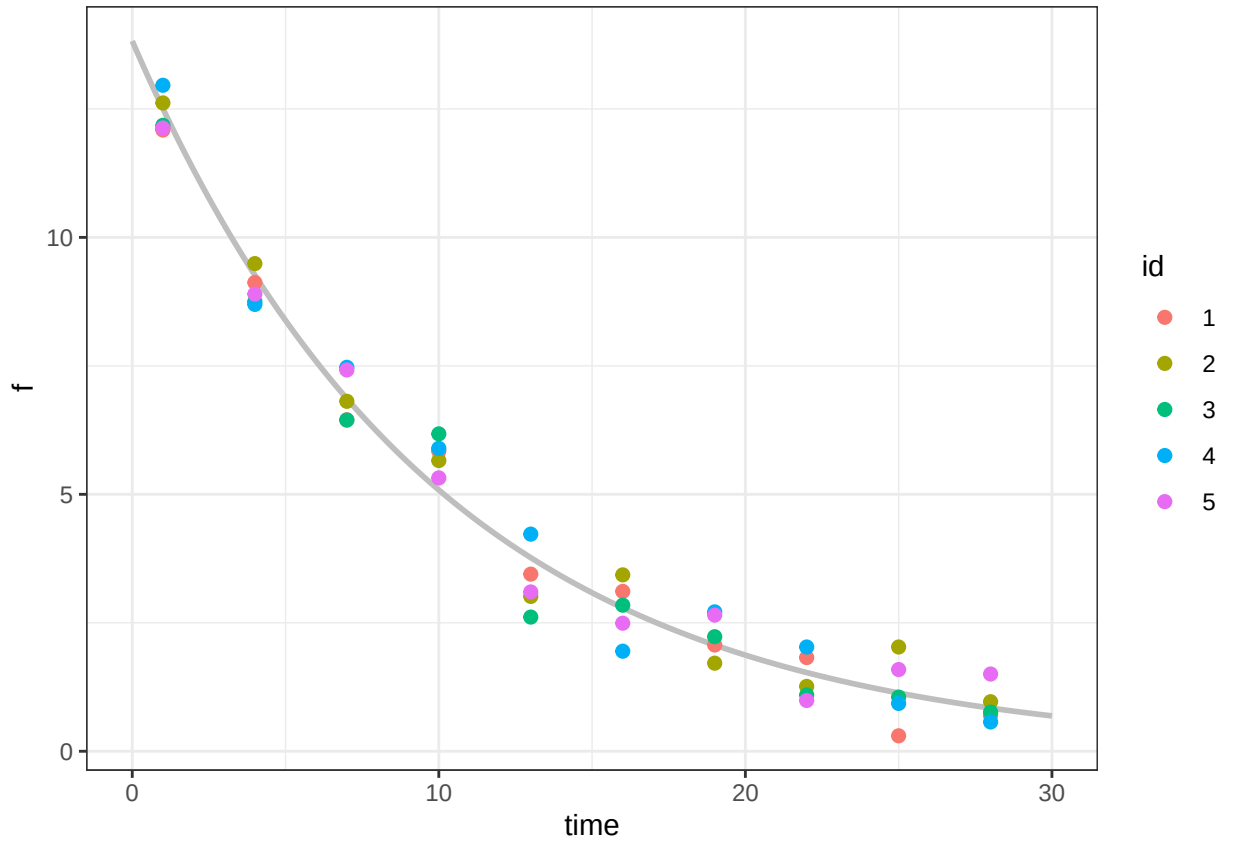


If we now want to sample a single individual parameter V_1 for a single individual $i = 1$, and 5 replicates of the longitudinal data for this individual, we must define the level of randomization as 'longitudinal'.

```
g <- list( size  = 5,
           level = 'longitudinal')

res3 <- simulx(model     = 'model/hierarchical1a.txt',
               parameter = p,
               output    = list(V, f, y),
               group     = g,
               settings  = list(seed = 12345))

print(res3$parameter)
```

```
##   id        V
## 1  1 7.234932
## 2  2 7.234932
## 3  3 7.234932
## 4  4 7.234932
## 5  5 7.234932
```

```
print(ggplot() +
        geom_line(data=res3$f, aes(x=time, y=f), colour="grey", size=1) +
        geom_point(data=res3$y, aes(x=time, y=y, colour=id), size=2))
```



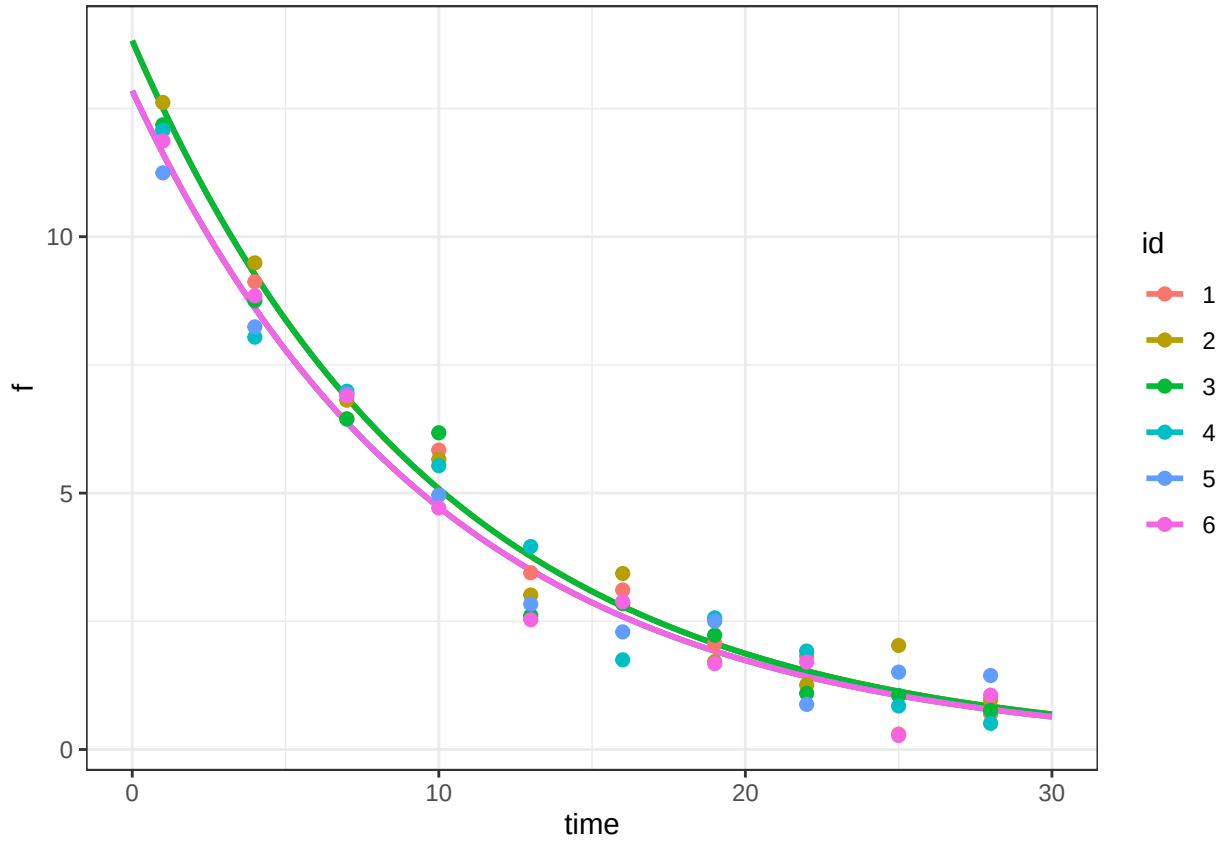
It is possible to combine several levels of randomization. We may want, for example, to sample 2 individual parameters V_1 and V_2, and 3 replicates of the longitudinal data for each individual.

```
g <- list( size  = c(2,3),
           level = c('individual','longitudinal'))
```

```
res4 <- simulx(model     = 'model/hierarchical1a.txt',
               parameter = p,
               output    = list(V, f, y),
               group     = g,
               settings  = list(seed = 12345))

print(res4$parameter)
```

```
##   id        V
## 1  1 7.234932
## 2  2 7.234932
## 3  3 7.234932
## 4  4 7.785818
## 5  5 7.785818
## 6  6 7.785818
```

```
print(ggplot() +
        geom_line(data=res4$f, aes(x=time, y=f, colour=id), size=1) +
        geom_point(data=res4$y, aes(x=time, y=y, colour=id), size=2))
```



Example 2

We still consider model (8) for the longitudinal data but V_i is now function of the weight of individual i:

(4)

$$\log(V_i) \underset{\text{i.i.d.}}{\sim} \mathcal{N}\left(\log\left(V_{\text{pop}}\left(w_i/w_{\text{pop}}\right)^{\beta}\right)\ ,\ \omega_V^2\right).$$

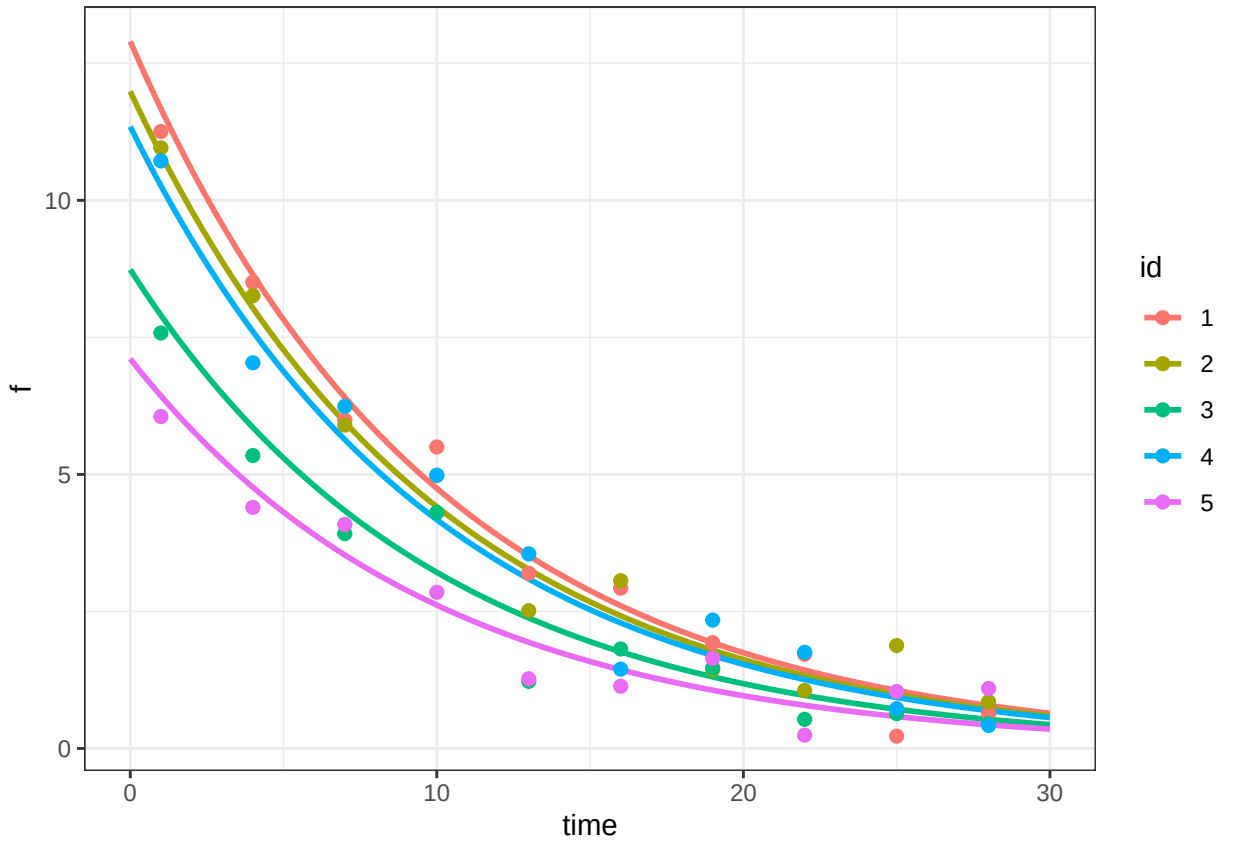
This model is implemented in `hierarchical1b.txt`.

We therefore need to define the individual weight w_i and the population weight w_{pop} as inputs of the model by adding them to the vector parameter.

```
p <- c(V_pop=10, omega_V=0.3, beta=1, w=75, w_pop=70, k=0.1, a=0.5)
f <- list(name='f', time=seq(0, 30, by=0.1))
y <- list(name='y', time=seq(1, 30, by=3))
V <- list(name='V')

g <- list( size  = 5,
           level = 'individual')

res5 <- simulx(model     = 'model/hierarchical1b.txt',
               parameter = p,
               output    = list(V, f, y),
               group     = g,
               settings  = list(seed = 12345))

print(res5$parameter)
```

```
##   id         V
## 1  1  7.751713
## 2  2  8.341948
## 3  3 11.446545
## 4  4  8.813773
## 5  5 14.066331
```

```
print(ggplot() +
        geom_line(data=res5$f, aes(x=time, y=f, colour=id),size=1) +
        geom_point(data=res5$y, aes(x=time, y=y, colour=id), size=2))
```



Remark 1: The model for the individual parameter V_i is a linear Gaussian model. Indeed, an equivalent mathematical representation of model (10) is the following one:

$$\log(V_i) = \log(V_{\text{pop}}) + \beta \log(w_i/w_{\text{pop}}) + \eta_i$$

where $\eta_i \sim_{\text{i.i.d.}} \mathcal{N}(0, \ , \ \omega_V^2)$.

It is then possible to take advantage of this representation and implement the model for the individual parameters as a linear model. We could therefore replace the section `[INDIVIDUAL]` of `hierarchical1b.txt` with the following one:

Remark 2: The individual covariate w_i is fixed and the same for all the individuals ($w_i = 75$) in this examples. We will see in the next article how to simulate individual covariates. Individual covariates can also be provided in a data file.

Simulation of a hierarchical model: the individual covariates

---

R scripts: hierarchical2.R

Mlxtran codes: model/hierarchical2.txt

Introduction

We have seen in the previous article that the vector of *individual parameters* ψ_i is treated as a random vector in a population context.

When the probability distribution of ψ_i depends on a vector of *individual covariates* c_i, we can also consider that c_i is a random vector sampled from a population distribution $\mathrm{p}(\, c_i \,; \theta)$.

In this context, the model for individual i is the joint distribution of the longitudinal data y_i, the individual parameters ψ_i and the individual covariates c_i:

$$\mathrm{p}(y_i, \psi_i, c_i; \theta) = \mathrm{p}(y_i|\psi_i)\, \mathrm{p}(\psi_i c_i; \theta)\, \mathrm{p}(c_i; \theta).$$

The model $\mathrm{p}(c_i; \theta)$ for the individual covariates is implemented in the section `[COVARIATE]`.

Example

We consider the model defined in the previous article:

- Given V_i, each y_{ij} is normally distributed:

(5)
$$y_{ij}|V_i \sim \mathcal{N}\left(\frac{100}{V_i} e^{-k\, t_{ij}}, a^2\right).$$

- V_i is lognormally distributed and depends on the covariate w_i:

(6)
$$\log(V_i) \underset{\text{i.i.d.}}{\sim} \mathcal{N}\left(\log\left(V_{\text{pop}}\, (w_i/w_{\text{pop}})^{\beta}\right)\ ,\ \omega_V^2\right).$$

We assume furthermore that w_i is normally distributed:

(7)
$$w_i \underset{\text{i.i.d.}}{\sim} \mathcal{N}\left(w_{\text{pop}}\ ,\ \omega_w^2\right).$$

This model is implemented in `hierarchical3.txt`.

We will use simulx for sampling one individual covariate w_i, one individual parameter V_i and one sequence $y_i = (y_{ij})$ for one individual $i = 1$, with some given values of the population parameters V_{pop}, ω_V, w_{pop}, ω_w, k and a.

```
p <- c(V_pop=10, omega_V=0.1, beta=1, w_pop=70, omega_w=12, k=0.15, a=0.5)

f   <- list(name='f', time=seq(0, 30, by=0.1))
y   <- list(name='y', time=seq(1, 30, by=3))
ind <- list(name=c('w','V'))
out <- list(ind, f, y)

res1 <- simulx(model     = 'model/hierarchical2.txt',
               parameter = p,
```

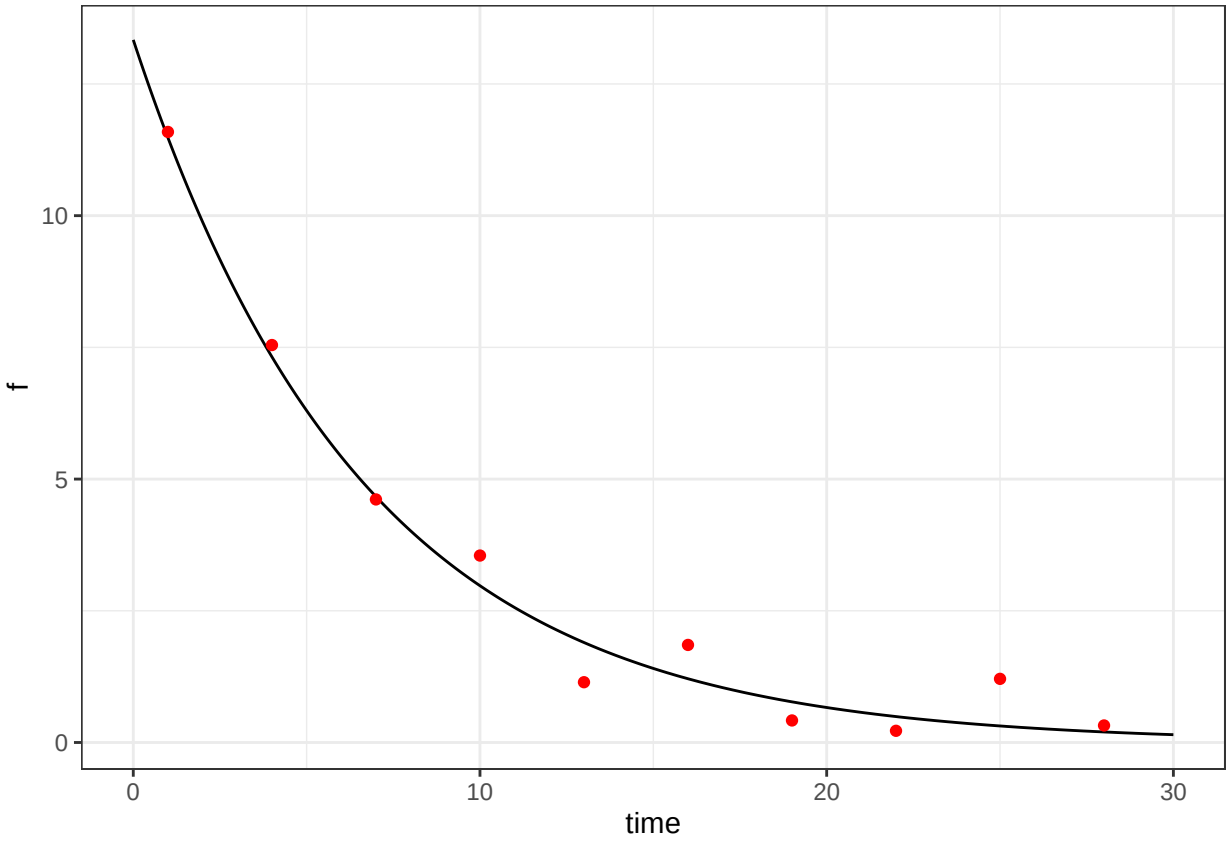
```
              output    = out,
              settings  = list(seed = 12345))
```

When several individual parameters and/or covariates are defined in the list of outputs, simulx also returns them as an additional element parameter.

```
print(res1$parameter)
```

```
##          w       V
## 1 57.05343 7.49811
```

```
plot(ggplot() +
  geom_line( data=res1$f, aes(x=time, y=f), colour="black") +
  geom_point(data=res1$y, aes(x=time, y=y), colour="red"))
```

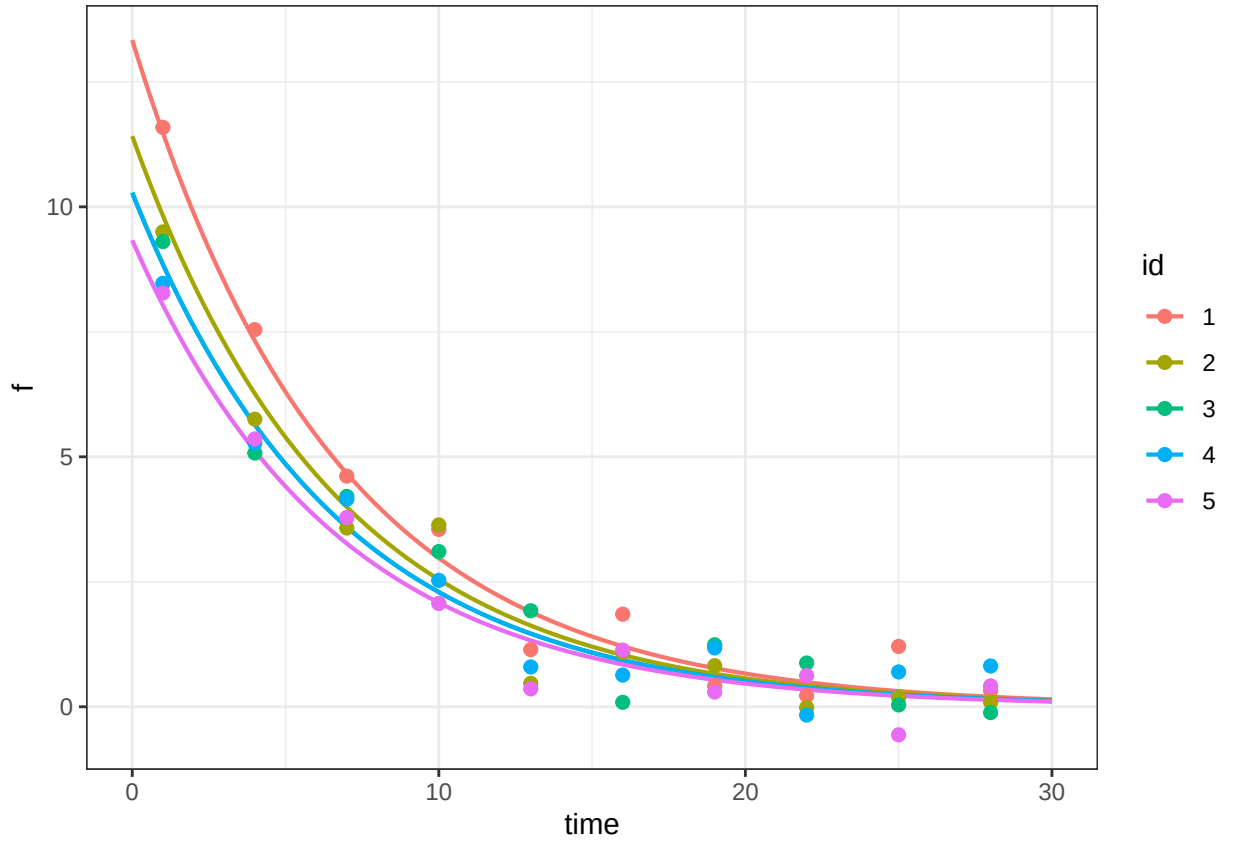


Instead of only one individual, let us now simulate a group of 5 individuals with individual covariates w_i drawn for each individual $i = 1, 2, \ldots, 5$. The level of randomization is therefore 'covariate'.

```
g <- list( size  = 5,
           level = 'covariate')

res2 <- simulx(model     = 'model/hierarchical2.txt',
               parameter = p,
               output    = out,
               group     = g,
               settings  = list(seed = 12345))

print(res2$parameter)
```

```
##   id        w         V
## 1  1 57.05343  7.498110
## 2  2 59.98875  8.760768
## 3  3 72.64440  9.723821
## 4  4 62.18951  9.728031
## 5  5 80.88824 10.715601
```

```
plot(ggplot() + geom_line(data=res2$f, aes(x=time, y=f, colour=id), size=0.75) +
       geom_point(data=res2$y, aes(x=time, y=y, colour=id), size=2))
```



We can combine several levels of randomization, and draw, for instance, 2 individual covariates w_1, w_2 and 3 individual parameters V_i per covariate

```
g <- list( size  = c(2,3),
           level = c('covariate','individual'))

res3 <- simulx(model     = 'model/hierarchical2.txt',
               parameter = p,
               output    = out,
               group     = g,
               settings  = list(seed = 12345))

print(res3$parameter)
```

```
##   id        w        V
## 1  1 57.05343 7.498110
## 2  2 57.05343 8.332094
## 3  3 57.05343 7.636891
```

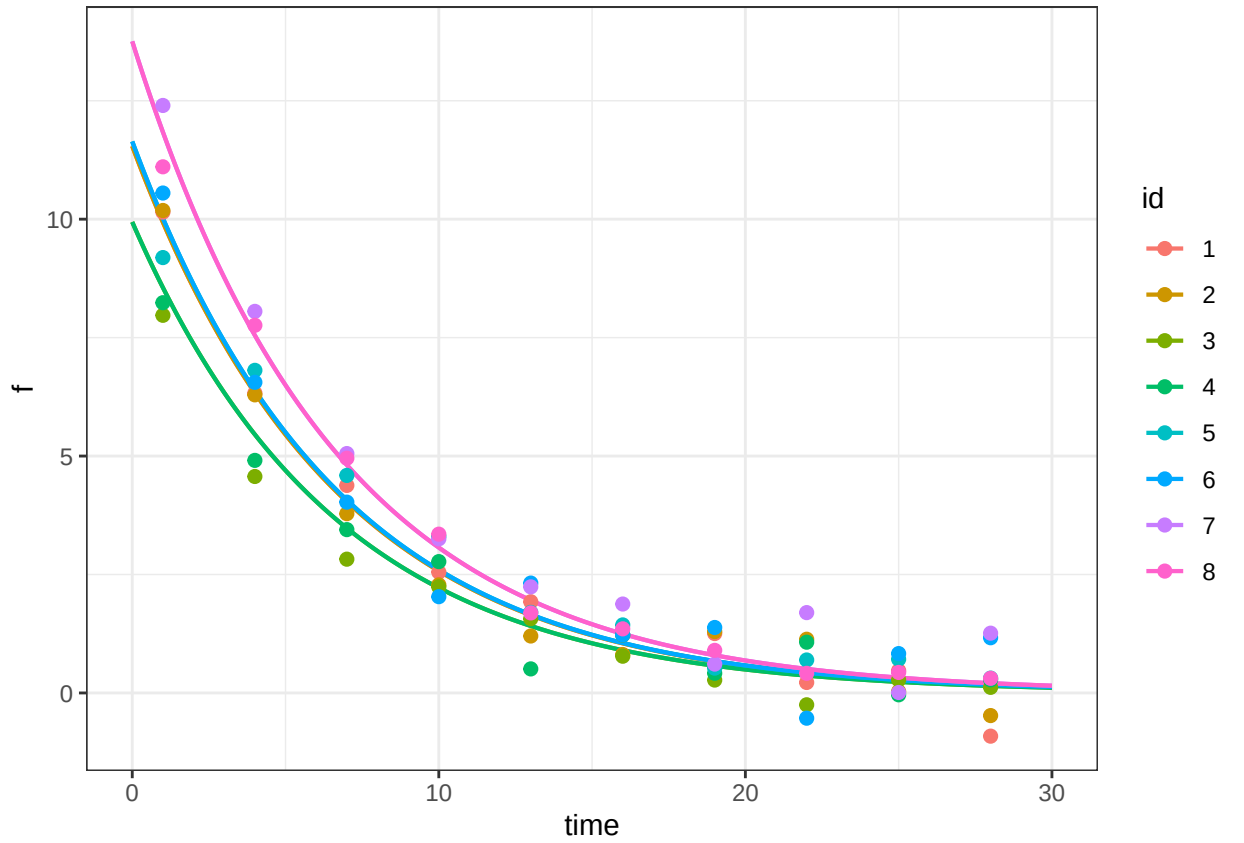
```
## 4  4 59.98875 9.383776
## 5  5 59.98875 7.946958
## 6  6 59.98875 8.999809
```

In the next example, we draw 2 individual covariates, 2 individual parameters per covariate and 2 sequences of longitudinal data per individual parameters.

```
g <- list( size  = c(2,2,2),
           level = c('covariate','individual','longitudinal'))

res4 <- simulx(model     = 'model/hierarchical2.txt',
               parameter = p,
               output    = out,
               group     = g,
               settings  = list(seed = 123123))

print(res4$parameter)
```

```
##   id        w         V
## 1  1 67.46624  8.663878
## 2  2 67.46624  8.663878
## 3  3 67.46624 10.059836
## 4  4 67.46624 10.059836
## 5  5 57.21343  8.590764
## 6  6 57.21343  8.590764
## 7  7 57.21343  7.270871
## 8  8 57.21343  7.270871
```

```
plot(ggplot() + geom_line(data=res4$f, aes(x=time, y=f, colour=id), size=0.75) +
  geom_point(data=res4$y, aes(x=time, y=y, colour=id), size=2))
```




```
pa <- c(V_pop=10, omega_V=0.1, w_pop=70, beta=1, k=0.15, a=0.5)
pw <- data.frame(id=1:4, w=c(60,70,80,90))
res5a <- simulx(model     = 'model/hierarchical1b.txt',
                parameter = list(pa,pw),
                output    = out,
                settings  = list(seed = 123123))

print(res5a$parameter)
```

```
##   id  w         V
## 1  1 60  8.392343
## 2  2 70  8.989259
## 3  3 80 11.928734
## 4  4 90 13.513762
```

```
res5b <- simulx(model     = 'model/hierarchical1b.txt',
                parameter = list(pa,pw),
                output    = out,
                group     = list(size = 7),
                settings  = list(seed = 1231))

print(res5b$parameter)
```

```
##   id  w         V
## 1  1 60  8.942568
## 2  2 70  9.998305
## 3  3 80 12.855733
```

```
## 4  4 90 11.994321
## 5  5 60 11.257617
## 6  6 70 11.498215
## 7  7 90 11.851334
```

```
res5c <- simulx(model     = 'model/hierarchical1b.txt',
                parameter = list(pa,pw),
                output    = out,
                group     = list(size = 2),
                settings  = list(seed = 123123))

print(res5c$parameter)
```

```
##   id  w         V
## 1  1 70  9.791067
## 2  2 80 10.273438
```

Model with categorical random covariates

R scripts: catcov.R

Mlxtran codes: model/catcov1A.txt ; catcov1B.txt ; catcov2A.txt ; catcov2B.txt

---

Introduction

Categorical variables take a finite number of values from a set that is not necessarily numerical or even ordered, e.g., gender, country or ethnicity.

The approach taken for continuous covariates extends easily to categorical ones. For simplicity's sake, let us consider a unique covariate c_i that takes its values in $\{a_1, a_2, \ldots, a_K\}$, and a unique parameter ψ_i.

Here, a reference covariate value c_{pop} is a reference category, i.e., a specific element a_{k^*} of $\{a_1, a_2, \ldots, a_K\}$.

Assume for instance a linear model of covariates. Then, the prediction $\tilde{\psi}_i$ for ψ_i is given by :

$$h(\tilde{\psi}_i) = h(\psi_{\text{pop}}) + \beta_1 \mathbb{1}_{c_i=a_1} + \beta_2 \mathbb{1}_{c_i=a_2} + \ldots + \beta_K \mathbb{1}_{c_i=a_K},$$

with $\beta_{k^*} = 0$ and where $\mathbb{1}$ is the indicator function ($\mathbb{1}_A = 1$ if A is true, 0 otherwise).

This model is equivalent to

$$h(\tilde{\psi}_i) = \begin{cases} h(\psi_{\text{pop}}) & \text{if } c_i = a_{k^*} \\ h(\psi_{\text{pop}}) + \beta_k & \text{if } c_i = a_k \neq a_{k^*}. \end{cases}$$

We see that if the covariate has K categories, $K-1$ coefficients (β_k) are required for defining the covariate model.

Example 1

Consider a model where gender takes the values M and F with

$$\mathbb{P}(\text{gender}_i = F) = p_F$$

Then, gender is used as a categorical covariate for describing the variability of k

$$\log(k_i) = \log(k_{\text{pop}}) + \beta_F \mathbb{1}_{\text{gender}_i=F} + \eta_i$$

That means that the *typical* values of k for a male and a female are, respectively, k_{pop} and $k_{\text{pop}} e^{\beta_F}$.

Function of time is then defined as

$$f(t; k_i) = e^{-k_i\, t}$$

This model is implemented in `catcov1A.txt`:

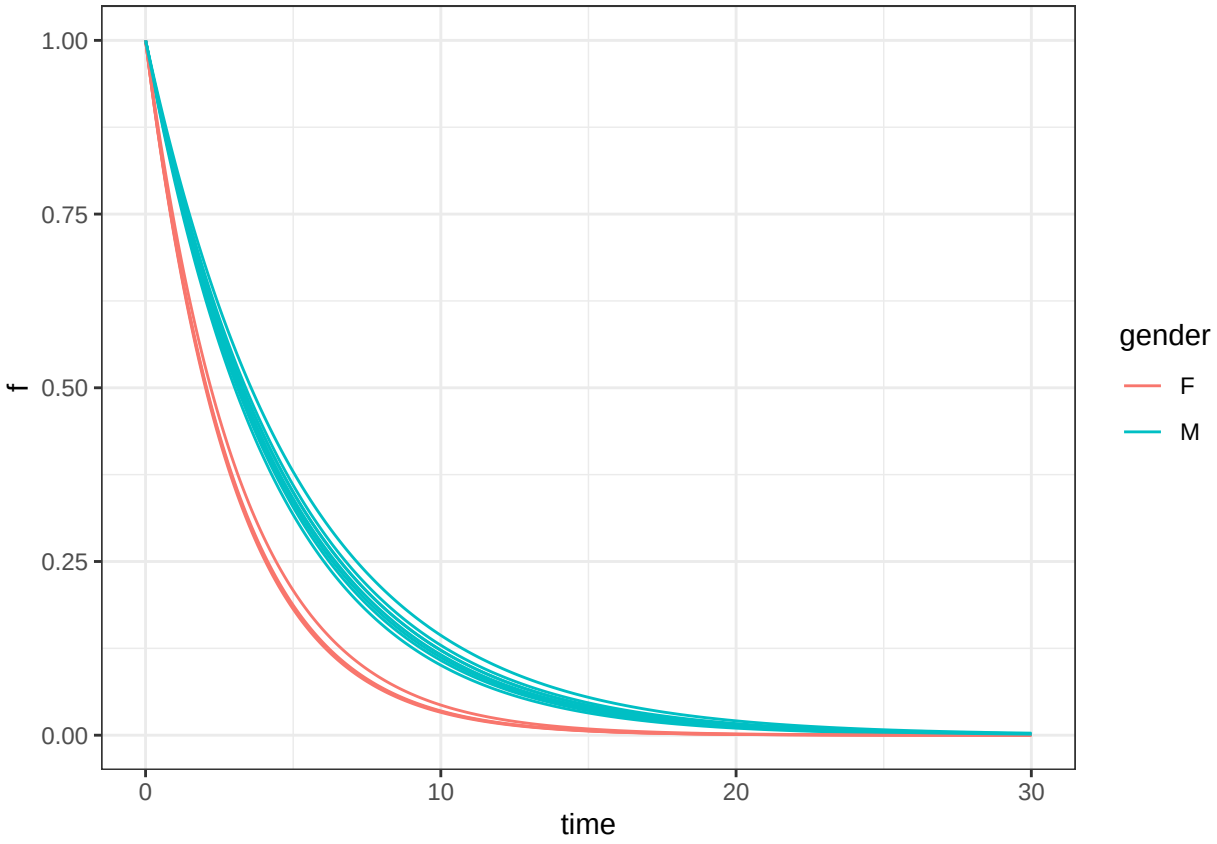
Let us generate 12 simulated genders, 12 individual parameters k_i (one per simulated gender) and 12 functions $f(t; k_i)$ (one per individual parameter) with this model:

```
p <- c(p_F=0.4,k_pop=0.2, omega_k=0.1, beta_F=0.6)
f <- list(name='f', time=seq(0, 30, by=0.1))
ind <- list(name=c("gender","k"))
```

```
res1 <- simulx(model    = "model/catcov1A.txt",
               parameter= p,
               output   = list(ind, f),
               group    = list(size=12, level='covariate'),
               settings = list(seed=12345))

print(head(res1$parameter))
```

```
##   id gender         k
## 1  1      F 0.3352546
## 2  2      M 0.2044563
## 3  3      F 0.3414598
## 4  4      M 0.2189958
## 5  5      F 0.3379371
## 6  6      M 0.2100349
```

```
fg <- merge(res1$f,res1$parameter)
plot(ggplot(data=fg) + geom_line(aes(x=time, y=f, group=id, colour=gender), size=0.5))
```



Remark: `simulx` supports any alphanumeric (alphabetic and numeric) character to define the categories of a categorical variable in a block a block `EQUATION` or in a bloc `DEFINITION`.

The model for k_i can be represented in an equivalent manner as follows:

$$\tilde{k}_i = \begin{cases} k_{\text{pop}} & \text{if} \quad \text{gender}_i = M \\ k_{\text{pop}} e^{\beta_F} & \text{if} \quad \text{gender}_i = F. \end{cases} \tag{37}$$

$$\log(k_i) = \log(\tilde{k}_i)) + \eta_i \tag{38}$$

This model is implemented in `catcov1B.txt`:

Implementations `catcov1A.txt` and `catcov1B.txt` are equivalent.

Example 2

Let us consider now an extension of the previous model, with a continuous covariate w and a categorical covariate trt with three categories TA, TB and TC. The reference category is B in this example.

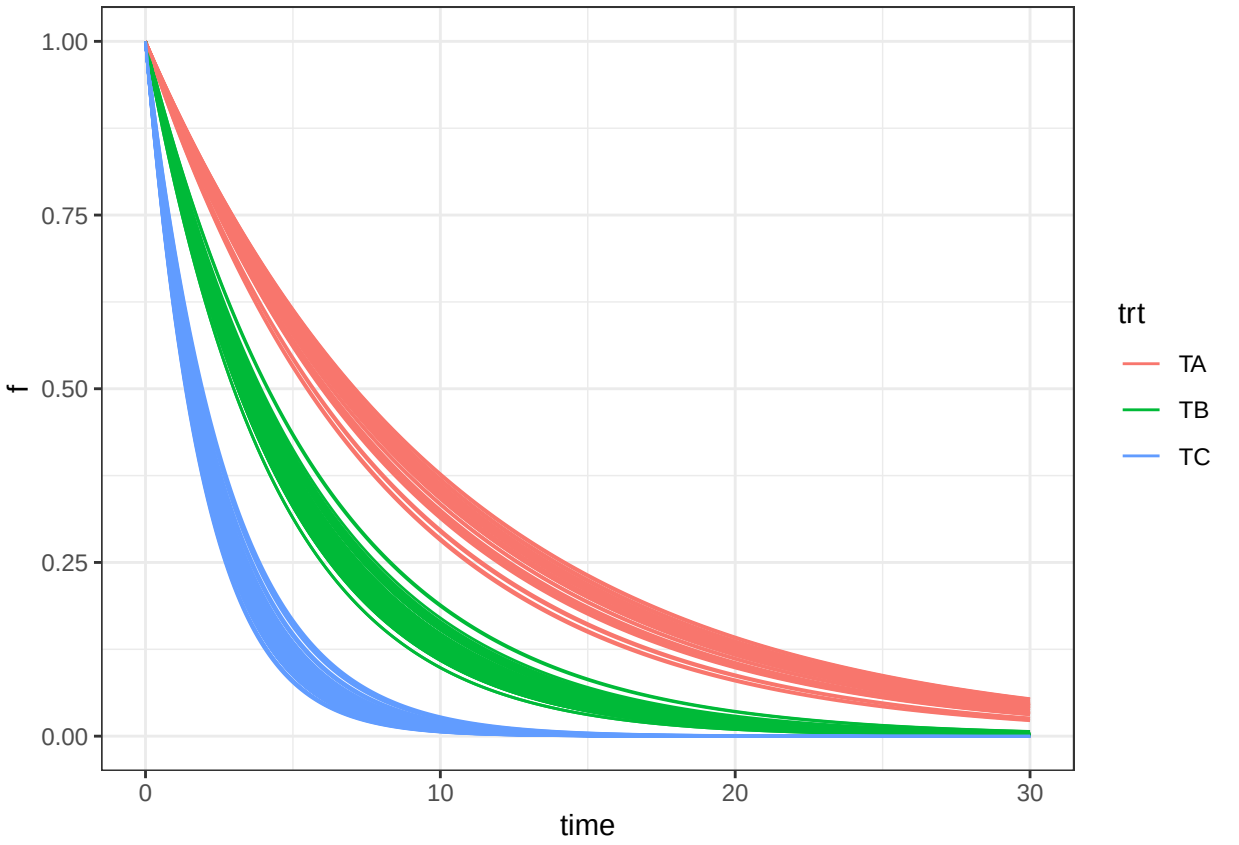
This model is implemented in `catcov2A.txt`:

We will use `simulx` for generating 200 individuals with different simulated covariates:

```
p   <- c(w_pop=70, omega_w=15, p_A=0.2, p_B=0.4, k_pop=0.2, omega_k=0.05)
f   <- list(name='f', time=seq(0, 30, by=0.1))
ind <- list(name=c('w','k','trt'))

res3 <- simulx(model     = "model/catcov2A.txt",
               parameter = p,
               output    = list(ind, f),
               group     = list(size=200, level='covariate'),
               settings  = list(seed=12345))

print(head(res3$parameter))
```

```
##   id        w         k trt
## 1  1 53.81679 0.1820028  TB
## 2  2 57.48594 0.1944052  TB
## 3  3 73.30550 0.4348502  TC
## 4  4 60.23688 0.2030887  TB
## 5  5 83.61031 0.4441325  TC
## 6  6 58.68136 0.1978523  TB
```

```
fg<-merge(res3$f,res3$parameter)
plot(ggplot(data=fg) + geom_line(aes(x=time, y=f, group=id, colour=trt), size=0.5))
```



Until now, the categorical covariates were simulated using section `[COVARIATE]`of the model `Mlxtran`. Categorical covariates can also be used as *inputs* of the model.

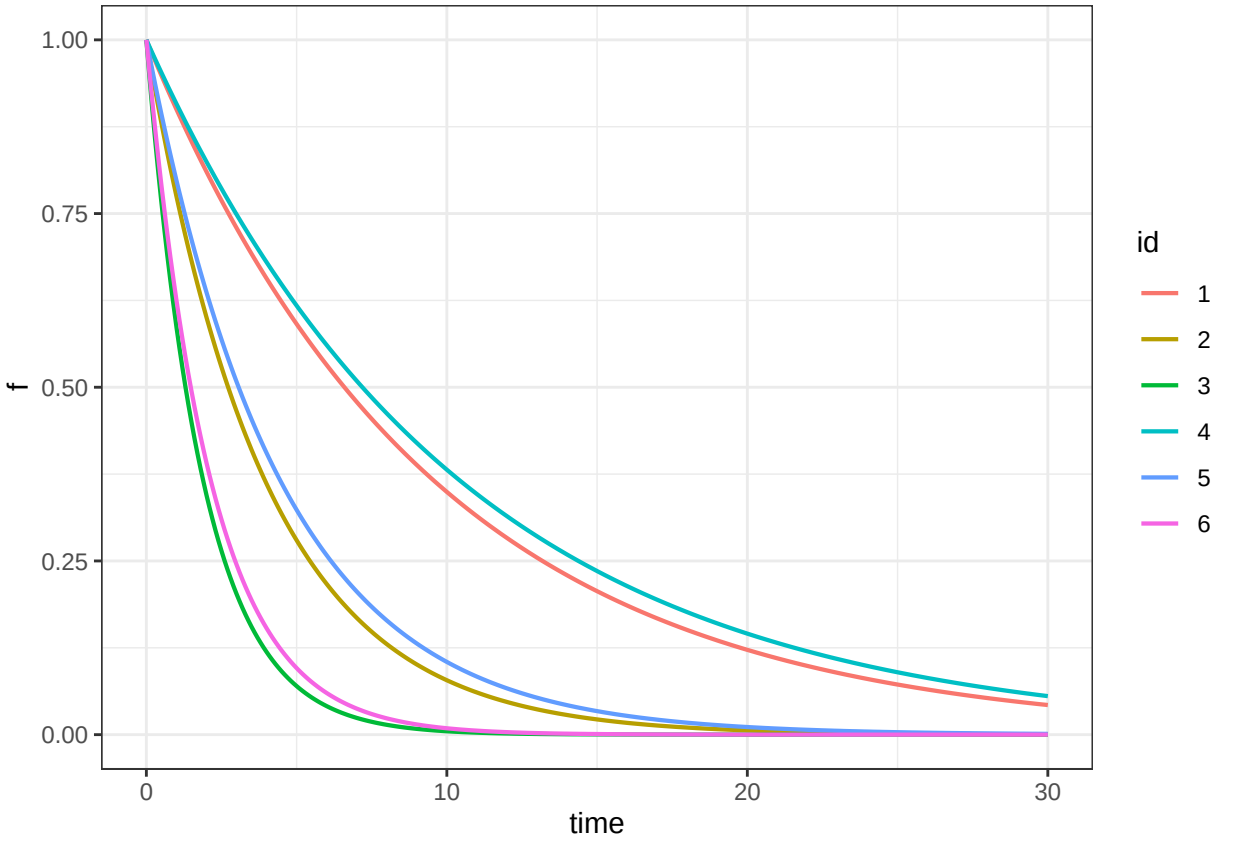
FOr example, model `catcov2B.txt` has only two sections `[INDIVIDUAL]` and `[LONGITUDINAL]`:

The values of the covariates w and trt for 6 individuals are defined in the following `R` script. They are then added to the list of input parameters.

```
p.indiv <- inlineDataFrame("
id   w    trt
 1  80     TA
 2  60     TB
 3  85     TC
 4  55     TA
 5  90     TB
 6  60     TC
")
p.pop <- c(k_pop=0.2, omega_k=0.1)

f <- list(name='f', time=seq(0, 30, by=0.1))
ind <- list(name=c('w','k','trt'))

res4 <- simulx(model     = "model/catcov2B.txt",
               parameter = list(p.indiv, p.pop),
               output    = list(ind, f),
               settings  = list(seed=1234))

print(res4$parameter)
```

```
##   id  w          k trt
## 1  1 80 0.10516783  TA
## 2  2 60 0.25470290  TB
## 3  3 85 0.53205950  TC
## 4  4 55 0.09641153  TA
## 5  5 90 0.22568857  TB
## 6  6 60 0.46898138  TC
```

```
plot(ggplot(data=res4$f, aes(x=time, y=f, colour=id)) + geom_line(size=0.75))
```



Simulation of a hierarchical model: the population parameters

---

R scripts: hierarchical3.R

Mlxtran codes: model/hierarchical3.txt

Introduction

We have seen in the previous article that the vector of *individual parameters* ψ_i and the vector of *individual covariates* c_i can be treated as random vectors in a population context.

The probability distributions of ψ_i and c_i depend usually on a vector of *population parameters* θ, that can also be treated as a random vector with probability distribution $\mathrm{p}(\theta)$. This distribution can be either used as a *prior distribution* in a Bayesian context, as a description of the *uncertainty* about the population parameters, or a a description of some *inter-population variability*.

In this context, the model for individual i is the joint distribution of the longitudinal data y_i, the individual parameters ψ_i, the individual covariates c_i and the population parameters θ:

$$\mathrm{p}(y_i, \psi_i, c_i; \theta) = \mathrm{p}(y_i|\psi_i)\,\mathrm{p}(\psi_i|c_i, \theta)\,\mathrm{p}(c_i|\theta)\,\mathrm{p}(\theta).$$

The model $\mathrm{p}(\theta)$ for the population parameters is implemented in the section `[POPULATION]`.

Example

We consider the model defined in the previous article:

(8)
$$y_{ij}|V_i \sim \mathcal{N}\left(\frac{100}{V_i}e^{-k\,t_{ij}}, a^2\right).$$

(9)
$$\log(V_i) \underset{\text{i.i.d.}}{\sim} \mathcal{N}\left(\log\left(V_{\text{pop}}\,(w_i/w_{\text{pop}})^{\beta}\right)\ ,\ \omega_V^2\right).$$

(10)
$$w_i \underset{\text{i.i.d.}}{\sim} \mathcal{N}\left(w_{\text{pop}}\ ,\ \omega_w^2\right).$$

We furthermore assume that w_{pop} and V_{pop} are, respectively, normally and log-normally distributed:

(11)
$$w_{\text{pop}} \underset{\text{i.i.d.}}{\sim} \mathcal{N}\left(w_s\ ,\ \gamma_w^2\right) \tag{39}$$
$$\log(V_{\text{pop}}) \underset{\text{i.i.d.}}{\sim} \mathcal{N}\left(\log(V_s)\ ,\ \gamma_V^2\right) \tag{40}$$

This model is implemented in `hierarchical3.txt`.

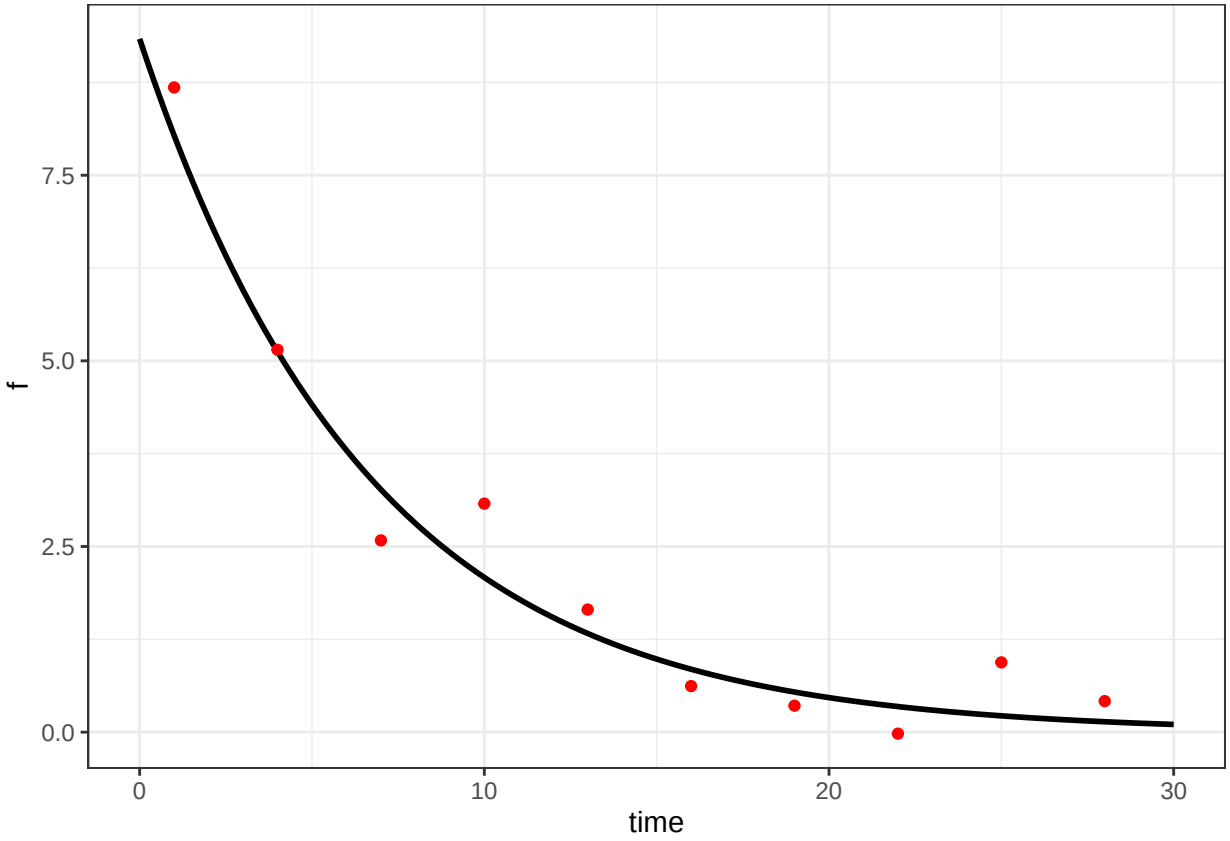
We will use simulx for sampling one set of population parameters (w_{pop}, V_{pop}), one individual covariate w_i, one individual parameter V_i and one sequence $y_i = (y_{ij})$ for one individual i.

```
p <- c(Vs=10, gV=0.1, ws=70, gw=10, omega_w=12, omega_V=0.15, beta=1, k=0.15, a=0.5)
f <- list(name='f', time=seq(0, 30, by=0.1))
y <- list(name='y', time=seq(1, 30, by=3))
ind <- list(name=c('w','V'))
```

```
pop <- list(name=c('w_pop','V_pop'))

res <- simulx(model     = 'model/hierarchical3.txt',
              parameter = p,
              output    = list(pop, ind, f, y),
              settings  = list(seed = 123456))

print(res$parameter)
```

```
##      w_pop    V_pop        w        V
## 1 64.63595 11.17389 71.00768 10.71065
```

```
print(ggplot() + geom_line(data=res$f, aes(x=time, y=f), size=1) +
geom_point(data=res$y, aes(x=time, y=y), colour='red'))
```



Instead of only one individual, we can easily simulate a group of several individuals combining several levels of randomization.

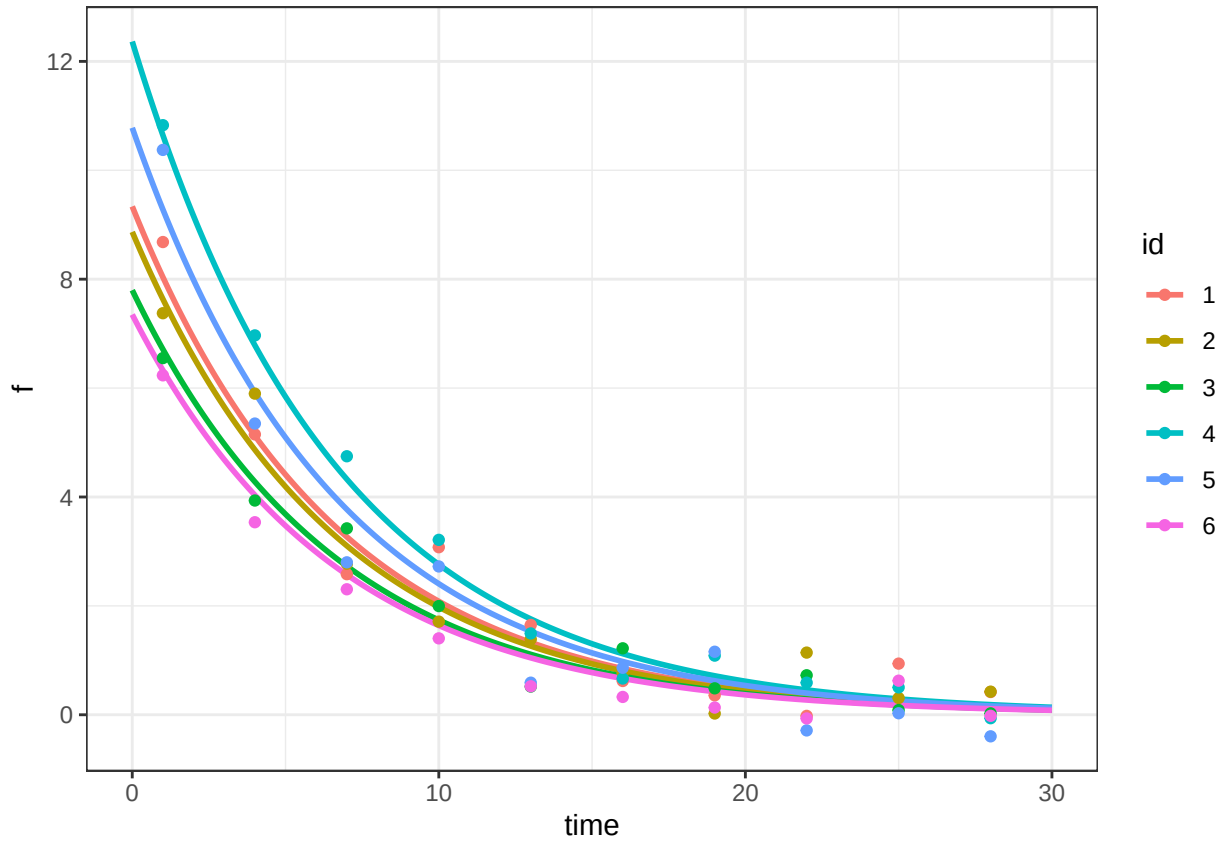
We can draw, for instance, 2 populations and 3 individual covariates per population,

```
g <- list(size=c(2,3),
          level=c('population','covariate'))

res <- simulx(model     = 'model/hierarchical3.txt',
              parameter = p,
              output    = list(pop, ind, f, y),
              group     = g,
              settings  = list(seed = 123456))
```

```
print(res$parameter)
```

```
##   id    w_pop     V_pop        w         V
## 1  1 64.63595 11.173891 71.00768 10.710646
## 2  2 64.63595 11.173891 53.72723 11.275816
## 3  3 64.63595 11.173891 80.15044 12.824081
## 4  4 75.30978  9.247238 69.11859  8.087218
## 5  5 75.30978  9.247238 71.44946  9.275857
## 6  6 75.30978  9.247238 79.76655 13.605292
```

```
print(ggplot() + geom_line(data=res$f, aes(x=time, y=f, colour=id), size=1) +
geom_point(data=res$y, aes(x=time, y=y, colour=id)))
```



or 2 populations and 3 individual parameters per population (in such case, the three individuals of a same population share the same covariates).

```
g <- list(size=c(2,3),
          level=c('population','individual'))

res <- simulx(model     = 'model/hierarchical3.txt',
              parameter = p,
              output    = list(pop, ind, f, y),
              group     = g,
              settings  = list(seed = 123456))

print(res$parameter)
```

```
##   id    w_pop     V_pop        w         V
## 1  1 64.63595 11.173891 71.00768 10.710646
## 2  2 64.63595 11.173891 71.00768 14.902490
## 3  3 64.63595 11.173891 71.00768 11.361238
## 4  4 75.30978  9.247238 64.40106  7.535243
## 5  5 75.30978  9.247238 64.40106  8.360805
## 6  6 75.30978  9.247238 64.40106 10.984493
```

Library of probability distributions

R script: distribution.R

Normal and transformations of normal distributions

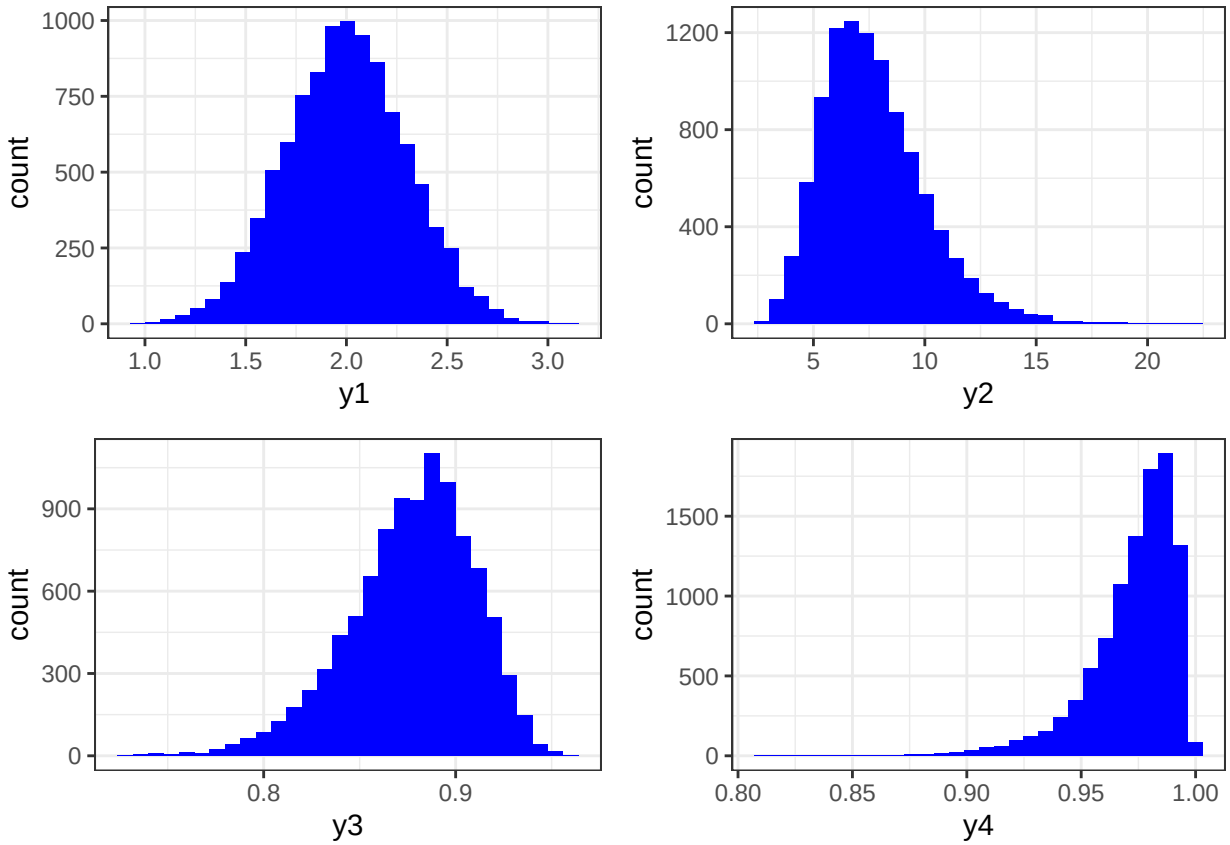
Normal, lognormal, logitnormal and probit normal distributions are available in the Mlxtran library of distributions:

```
normal.dist = inlineModel("
[LONGITUDINAL]
DEFINITION:
y1 = {distribution=normal, mean=2, sd=0.3}
y2 = {distribution=lognormal, mean=2, sd=0.3}
y3 = {distribution=logitnormal, mean=2, sd=0.3}
y4 = {distribution=probitnormal, mean=2, sd=0.3}
")
```

Here, mean=2 and sd=0.3 are the mean and standard deviation of the normal distributions:

$$
\begin{aligned}
y_1 &\sim \mathcal{N}(2, 0.3) \\
\log(y_2) &\sim \mathcal{N}(2, 0.3) \\
\text{logit}(y_3) = \log(y_3/(1-y_3)) &\sim \mathcal{N}(2, 0.3) \\
\text{probit}(y_4) = \Phi^{-1}(y_4) &\sim \mathcal{N}(2, 0.3)
\end{aligned}
$$

```
N <- 10000
y <- list(name=c('y1','y2','y3','y4'),time=(1:N))
normal.res <- simulx(model = normal.dist, output=y)

update_geom_defaults("bar",   list(fill = "blue"))
pl1 <- ggplot() + geom_histogram(data=normal.res$y1,aes(y1), bins=30)
pl2 <- ggplot() + geom_histogram(data=normal.res$y2,aes(y2), bins=30)
pl3 <- ggplot() + geom_histogram(data=normal.res$y3,aes(y3), bins=30)
pl4 <- ggplot() + geom_histogram(data=normal.res$y4,aes(y4), bins=30)

library(gridExtra)
grid.arrange(pl1, pl2, pl3, pl4)
```




```
z1 <- normal.res$y1$y1
print(c(mean(z1), sd(z1)))
```

```
## [1] 1.9980808 0.3011621
```

```
z2 <- log(normal.res$y2$y2)
print(c(mean(z2), sd(z2)))
```

```
## [1] 1.9996600 0.2976984
```

```
z3 <- log(normal.res$y3$y3/(1-normal.res$y3$y3))
print(c(mean(z3), sd(z3)))
```

```
## [1] 1.9966620 0.3016822
```

```
z4 <- qnorm(normal.res$y4$y4)
print(c(mean(z4), sd(z4)))
```

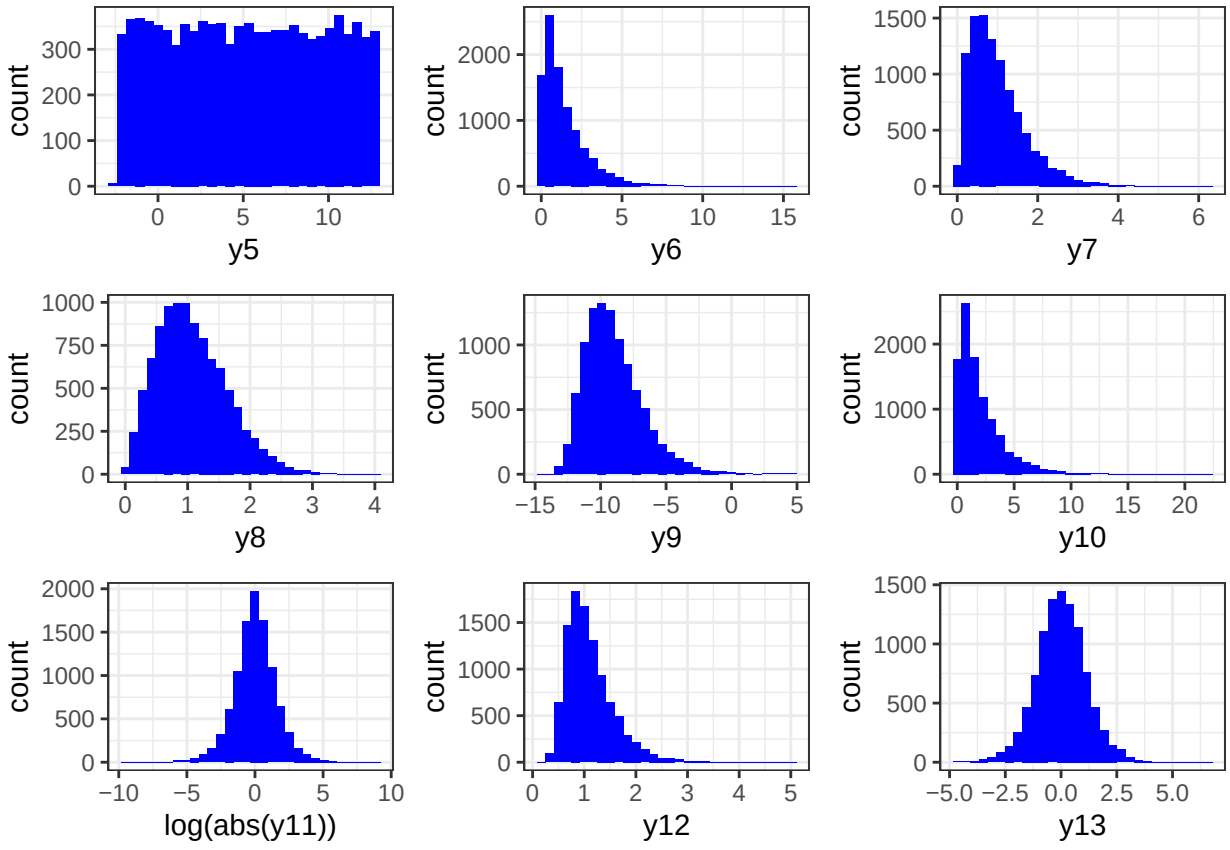
```
## [1] 2.0040576 0.2989264
```

Continuous probability distributions

Several continuous and discrete distributions are also available. The names and the parametrizations of these distributions are those used in R:

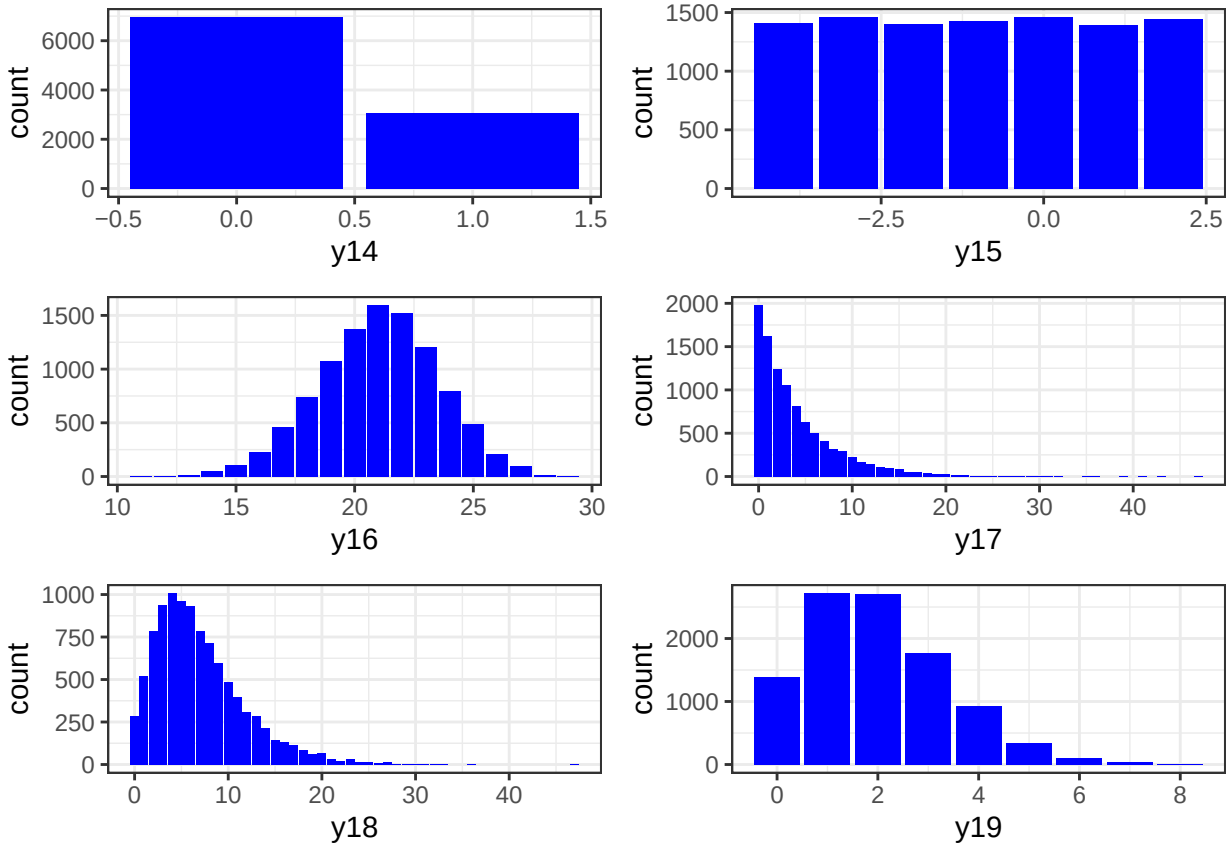
```
continuous.dist = inlineModel("
[LONGITUDINAL]
DEFINITION:
y5 = {distribution=uniform, min=-2.4, max=13}
```

```
y6 = {distribution=exponential, rate=0.7}
y7 = {distribution=gamma, shape=2, scale=0.5}
y8 = {distribution=weibull, shape=2, scale=1.2}
y9 = {distribution=extremeValue, location=-10, scale=1.8}
y10 = {distribution=chiSquared, df=2}
y11 = {distribution=cauchy, location=0, scale=1}
y12 = {distribution=fisherF, df1=40, df2=20}
y13 = {distribution=studentT, df=10}
")

y <- list(name=c('y5','y6','y7','y8','y9','y10','y11','y12','y13'),time=(1:N))
continuous.res <- simulx(model = continuous.dist, output=y)

pl5  <- ggplot() + geom_histogram(data=continuous.res$y5,aes(y5), bins=30)
pl6  <- ggplot() + geom_histogram(data=continuous.res$y6,aes(y6), bins=30)
pl7  <- ggplot() + geom_histogram(data=continuous.res$y7,aes(y7), bins=30)
pl8  <- ggplot() + geom_histogram(data=continuous.res$y8,aes(y8), bins=30)
pl9  <- ggplot() + geom_histogram(data=continuous.res$y9,aes(y9), bins=30)
pl10 <- ggplot() + geom_histogram(data=continuous.res$y10,aes(y10), bins=30)
pl11 <- ggplot() + geom_histogram(data=continuous.res$y11,aes(log(abs(y11))), bins=30)
pl12 <- ggplot() + geom_histogram(data=continuous.res$y12,aes(y12), bins=30)
pl13 <- ggplot() + geom_histogram(data=continuous.res$y13,aes(y13), bins=30)
grid.arrange(pl5, pl6, pl7, pl8, pl9, pl10, pl11, pl12, pl13)
```



Discrete probability distributions

```
discrete.dist = inlineModel("
[LONGITUDINAL]
DEFINITION:
y14 = {distribution=bernoulli, prob=0.3}
y15 = {distribution=discreteUniform, min=-4, max=2}
y16 = {distribution=binomial, size=30, prob=0.7}
y17 = {distribution=geometric, prob=0.2}
y18 = {distribution=negativeBinomial, size=3, prob=0.3}
y19 = {distribution=poisson, lambda=2}
")

y <- list(name=c('y14','y15','y16','y17','y18','y19'),time=(1:N))
discrete.res <- simulx(model = discrete.dist, output=y)

pl14 <- ggplot() + geom_bar(data=discrete.res$y14,aes(y14))
pl15 <- ggplot() + geom_bar(data=discrete.res$y15,aes(y15))
pl16 <- ggplot() + geom_bar(data=discrete.res$y16,aes(y16))
pl17 <- ggplot() + geom_bar(data=discrete.res$y17,aes(y17))
pl18 <- ggplot() + geom_bar(data=discrete.res$y18,aes(y18))
pl19 <- ggplot() + geom_bar(data=discrete.res$y19,aes(y19))
grid.arrange(pl14, pl15, pl16, pl17, pl18, pl19)
```



Correlation between variables

R script: correlation.R

Mlxtran code: model/correlation1.txt ; model/correlation2.txt

---

Introduction

Correlation can be introduced between random variables which are normally distributed.

Consider two scalar parameters ψ_{i1} and ψ_{i2} and two transformations h_1 and h_2 such that $h_1(\psi_{i1})$ and $h_2(\psi_{i2})$ are normally distributed:

$$h_1(\psi_{i1}) \sim \mathcal{N}(\mu_{i1}, \omega_1^2) \tag{41}$$
$$h_2(\psi_{i2}) \sim \mathcal{N}(\mu_{i2}, \omega_2^2) \tag{42}$$

It is then possible to define the (linear) correlation between $h_1(\psi_{i1})$ and $h_2(\psi_{i2})$.

If we use instead the following representation for ψ_{i1} and ψ_{i2}

$$h_1(\psi_{i1}) = \mu_{i1} + \eta_{i1} \tag{43}$$
$$h_2(\psi_{i2}) = \mu_{i2} + \eta_{i2} \tag{44}$$

where $\eta_{i1} \sim \mathcal{N}(0, \omega_1^2)$ and $\eta_{i2} \sim \mathcal{N}(0, \omega_2^2)$, then, defining the correlation between $h_1(\psi_{i1})$ and $h_2(\psi_{i2})$ is equivalent to defining the correlation between the so-called *random effects* η_{i1} and η_{i2}.

Remark 1: any correlation which is not defined is assumed to be 0.

Remark 2: with simulx 1.2, it is only possible to define correlations in sections `POPULATION`, `COVARIATE` and `INDIVIDUAL`. It is not possible to define correlations between residual errors in section `LONGITUDINAL`.

Examples

Example 1

We consider the function of time

$$f(t; a_i, b_i, c_i) = a_i + b_i t + c_i t^2$$

where the three individual parameters a_i b_i and c_i are, respectively, log-normally, normally and logit-normally distributed:

$$\log(a_i) \sim \mathcal{N}(\log(a_{\text{pop}}), \omega_a^2) \tag{45}$$
$$b_i \sim \mathcal{N}(b_{\text{pop}}, \omega_b^2) \tag{46}$$
$$\text{logit}(c_i) \sim \mathcal{N}(\text{logit}(c_{\text{pop}}), \omega_c^2) \tag{47}$$

The correlation matrix of the Gaussian vector $(\log(a_i), b_i, \text{logit}(c_i))$ is

$$R = \begin{pmatrix} 1 & r_{ab} & r_{ac} \\ r_{ab} & 1 & r_{bc} \\ r_{ac} & r_{bc} & 1 \end{pmatrix}$$

This model is implemented in the file `correlation1.txt`:

Let us use simulx for simulating 100 vectors of individual parameters $\psi_i(a_i, b_i, c_i)$

```
op <- list(name= c('a','b','c'))
of <- list(name='f', time=seq(0,4, by=0.1))

p <- c(a_pop=10,   b_pop=-5,   c_pop=0.8,
       o_a  =0.3,  o_b  =0.5,  o_c  =0.4,
       r_ab =-0.6, r_ac =-0.4, r_bc =0.7)

g <- list(size=100, level='individual')

res <- simulx(model     = "model/correlation1.txt",
              parameter = p,
              group     = g,
              output    = list(op,of))
```

Maximum likelihood estimates of the population parameters a_{pop}, b_{rm pop}$ and c_{pop} are the empirical medians of the simulated individual parameters:

```
p=res$parameter[,2:4]
print(sapply(p,median))
```

```
##          a          b          c
## 10.5150429 -4.9924886  0.7924052
```

Estimates of the standard deviations of ω_a, ω_b and ω_c are the empirical standard deviations of the transformed parameters $z_i = (\log(a_i), b_i, \text{logit}(c_i))$:

```
z=p
z[,"a"]=log(p[,"a"])
z[,"c"]=log(p[,"c"]/(1-p[,"c"]))
print(sapply(z,sd))
```

```
##         a         b         c
## 0.2995504 0.5112990 0.4072157
```

Estimate of the correlation matrix R is the empirical correlation of the transformed parameters z_i:

```
print(cor(z))
```

```
##            a          b          c
## a  1.0000000 -0.5979629 -0.3249590
## b -0.5979629  1.0000000  0.7359814
## c -0.3249590  0.7359814  1.0000000
```

Example 2

We consider in this example a function of time

$$f(t; a_i, b_i) = a_i + b_i t$$

where $\psi_i = (a_i, b_i)$ is defined by the following hierarchical model:

1. Conditionally to (w_i, h_i), a_i and b_i are normally distributed:

$$a_i = a_{\text{pop}} \left(\frac{w_i}{w_{\text{pop}}}\right) \left(\frac{h_i}{h_{\text{pop}}}\right)^{0.5} + \eta_{a,i} \tag{48}$$

$$b_i \ = b_{\text{pop}} + \eta_{b,i} \tag{49}$$

where $\eta_{a,i} \sim \mathcal{N}(0, \omega_a^2)$, $\eta_{b,i} \sim \mathcal{N}(0, \omega_b^2)$ and where $\text{corr}(\eta_{a,i}, \eta_{b,i}) = r_{ab}$.

2. w_i and h_i are normally distributed:

$$w_i \sim \mathcal{N}(w_{\text{pop}}, \omega_w^2) \tag{50}$$

$$h_i \sim \mathcal{N}(h_{\text{pop}}, \omega_h^2) \tag{51}$$

and $\text{corr}(w_i, h_i) = r_{wh}$.

This model is implemented in the file `correlation2.txt`:

Let us use simulx for simulating 100 vectors of individual covariates (w_i, h_i) and one vector of individual parameters (a_i, b_i) per individual,

```
op <- list(name= c('a_pred','a','b','w','h'))
of <- list(name='f', time=seq(0,4, by=0.1))

p <- c(a_pop=10,   b_pop=-5,   w_pop=70, h_pop=170,
       o_a  =0.3,  o_b  =0.5,  o_w  =10, o_h  =10,
       r_ab =-0.6, r_wh =0.8)

g <- list(size=100, level='covariate')

res <- simulx(model     = "model/correlation2.txt",
              parameter = p,
              group     = g,
              output    = list(op,of))
```

and display the empirical correlation matrix of the vector (a_i, b_i, w_i, h_i):

```
p=res$parameter[,2:6]
print(cor(p[,2:5]))
```

```
##             a           b          w          h
## a  1.00000000 -0.07430198 0.98086263 0.86370089
## b -0.07430198  1.00000000 0.04877629 0.00831015
## w  0.98086263  0.04877629 1.00000000 0.83814789
## h  0.86370089  0.00831015 0.83814789 1.00000000
```

Here, the submatrix composed by the first two rows and first two columns of this matrix is the empirical correlation matrix of (a_i, b_i), it is not the correlation matrix of $(\eta_{a,i}, \eta_{b,i})$. The fact that a_i is function of w_i and h_i also introduces some correlation between these variables.

If we want to estimate the correlation r_{ab} we need to consider the *conditional distribution* of (a_i, b_i) given (w_i, h_i) and not the marginal distribution of (a_i, b_i). We therefore need to use $\tilde{a}_i$ instead of a_i, where

$$\tilde{a}_i = a_i - a_{\text{pop}} \left(\frac{w_i}{w_{\text{pop}}}\right) \left(\frac{h_i}{h_{\text{pop}}}\right)^{0.5}$$

```
p[,"a"]=p[,"a"] - p[,"a_pred"]
print(cor(p[,2:5]))  #conditional correlation between a & b (given w & h)
```

```
##             a           b           w           h
## a  1.00000000 -0.64383497 -0.02564668 -0.07091711
## b -0.64383497  1.00000000  0.04877629  0.00831015
## w -0.02564668  0.04877629  1.00000000  0.83814789
## h -0.07091711  0.00831015  0.83814789  1.00000000
```

Inter occasion variability (mlxR >= 3.2.2)

R script: iov.R

Introduction

Assumption that an individual parameter remains constant over time can be weakened by introducing *intra-individual variability*, or *inter-occasion variability* (IOV), of individual parameters into the model.

The model then consists of splitting the study into K time periods or *occasions* and assuming that individual parameters can vary from occasion to occasion but remain constant within occasions.

In other words, we assume that there exists $\tau_0 = 0 < \tau_1 < \tau_2 < \dots$ such that the individual parameter for individual i is ψ_{ik} within occasion k, i.e. between time τ_{k-1} and time τ_k.

We will assume that the (possibly transformed) individual parameters are normally distributed. Then, there exists for each individual i,

- a vector $\eta_i^{(0)}$, of random effects which describes the random *inter-individual variability* of the individual parameters,
- a vector $\eta_{ik}^{(1)}$, of random effects which describes the random intra-individual variability of the individual parameters in occasion k, for each $1 \leq k \leq K$.

Here and in the following, the superscript (0) is used to represent *inter-individual variability*, i.e., variability at the individual level, while superscript (1) represents *inter-occasion variability*, i.e., variability at the occasion level for each individual.

Then, model for parameter ψ_{ik} assumes that there exists a transformation h (log, logit, probit, identity, ...) such that

$$h(\psi_{ik}) = h(\tilde{\psi}_{ik}) + \eta_i^{(0)} + \eta_{ik}^{(1)}$$

where $\tilde{\psi}_{ik}$ is the prediction of ψ_{ik}, i.e. of parameter ψ for individual i within occasion k.

Assume, for example, that the predicted individual parameter $\tilde{\psi}_{ik}$ has been defined in the `EQUATION` block, then the definition of ψ_{ik} will include the variability levels (id for individual and id*occ for occasion, for example), and the standard deviations of the random effects for each variability level:

Remark 1: `mlxR 3.2.2` supports *inter-occasion variability* for the individual parameters defined in section `[INDIVIDUAL]` of the `Mlxtran`code and for the individual covariates defined in section `[COVARIATE]`

Remark 2: `mlxR 3.2.2` only supports nested levels of variability. When there is only one level of variability, as described in this article, the variability levels should be defined by varlevel={id, id*occ} or just varlevel={id*occ} if there is no IIV in the model.

A basic example

Defining the same occasions for all the individuals

We consider in this example a one compartment PK model with IIV for absorption rate constant ka and volume V, and IIV and IOV for clearance Cl.

```
mymodel <- inlineModel("
[LONGITUDINAL]
input = {ka, V, Cl}
EQUATION:
Cc = pkmodel(ka, V, Cl)
```

```
[INDIVIDUAL]
input = {ka_pop, omega_ka, V_pop, omega_V, Cl_pop, omega_Cl, gamma_Cl}
DEFINITION:
ka = {distribution=lognormal, reference=ka_pop, sd=omega_ka}
V  = {distribution=lognormal, reference=V_pop,  sd=omega_V}
Cl = {distribution=lognormal, reference=Cl_pop, varlevel={id, id*occ}, sd={omega_Cl,gamma_Cl}}
")
```

We consider in this example three occasions of 12 hours. The beginnings of these 3 occasions should then be defined in a vector:

```
occ <- list(name="occ", time=c(0,12,24))
```

Parameters, outputs and design are defined as usual

```
param <- c(ka_pop=1, omega_ka=0.05, V_pop=10, omega_V=0.05, Cl_pop=5, omega_Cl=0.3, gamma_Cl=0.2)
trt <- list(time=c(0,12,24), amount=100)

out.cc <- list(name= 'Cc', time=seq(0,36,by=0.1))
out.param <- list(name=c( 'ka', 'V', 'Cl'))
```

simulx now uses varlevel as an additional argument:

```
res <-simulx(model     = mymodel,
             parameter = param,
             treatment = trt,
             varlevel  = occ,
             output    = list(out.cc, out.param),
             group     = list(size=3, level='individual'),
             settings  = list(seed=123123))

print(names(res))
```

```
## [1] "Cc"            "group"         "occasion"      "treatment"
## [5] "parameter.iiv" "parameter.iov"
```

An additional output occasion is automatically created

```
print(res$occasion)
```

```
##   id time occ
## 1  1    0   1
## 2  1   12   2
## 3  1   24   3
## 4  2    0   1
## 5  2   12   2
## 6  2   24   3
## 7  3    0   1
## 8  3   12   2
## 9  3   24   3
```

Parameters defined as outputs are now splitted into constant individual parameters (with inter individual variability) and time varying individual parameters (with inter occasion variability).

Note that the three typical individual clearances, denoted Cl0 are diplayed with the constant individual parameters

```
print(res$parameter.iiv)
```

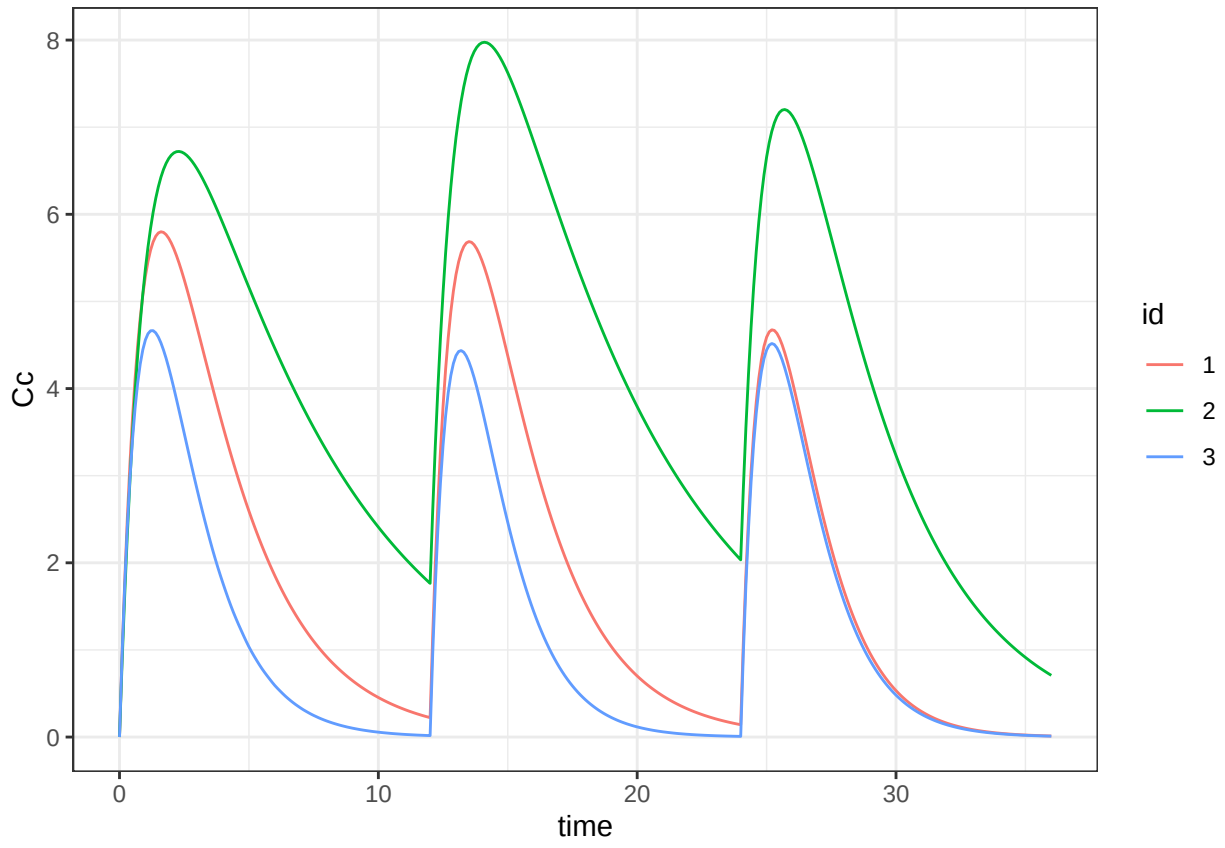
```
##   id        ka         V      Cl0
## 1  1 0.9894982  9.705295 4.464239
## 2  2 0.9481170 10.427893 2.151058
## 3  3 1.0216478  9.995837 6.135879
```

```
print(res$parameter.iov)
```

```
##   id time occ       Cl
## 1  1    0   1 3.462882
## 2  1   12   2 3.897200
## 3  1   24   3 6.301167
## 4  2    0   1 1.627533
## 5  2   12   2 1.628046
## 6  2   24   3 2.661930
## 7  3    0   1 6.072912
## 8  3   12   2 6.871969
## 9  3   24   3 6.576800
```

Plotting the predicted concentrations for the three subjects shows how the clearance varies from one occasion to another.

```
print(ggplot(res$Cc) +geom_line(aes(x=time,y=Cc,color=id)))
```



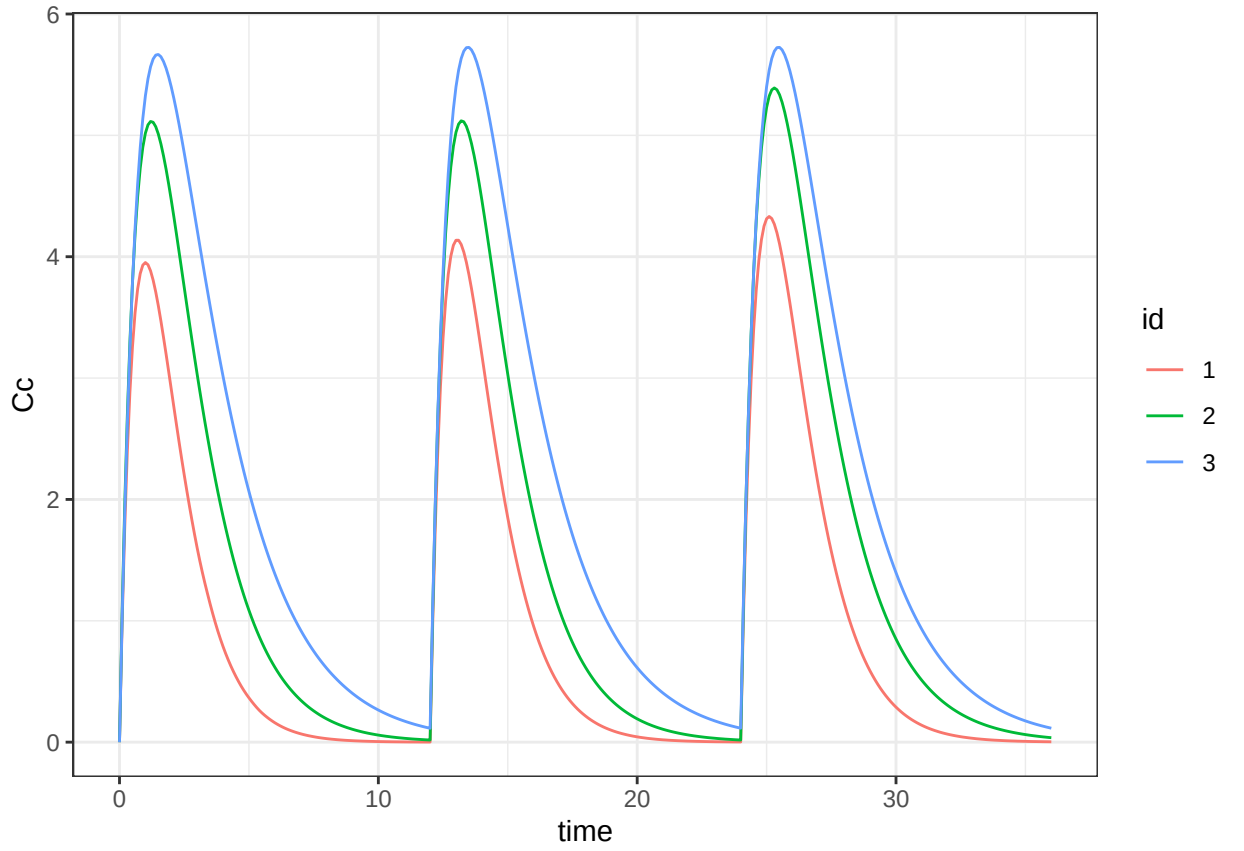
Defining individual occasions

Individual occasions can also be defined in a data frame:

```
occ <- inlineDataFrame("
id time occ
1   0    1
1   12   2
1   24   3
2   0    1
2   24   2
3   0    1
")

res <-simulx(model     = mymodel,
             parameter = param,
             treatment = trt,
             varlevel  = occ,
             output    = list(out.cc, out.param),
             settings  = list(seed=123))

print(res$parameter.iov)
```

```
##   id time occ       Cl
## 1  1    0   1 9.306027
## 2  1   12   2 8.446498
## 3  1   24   3 7.666498
## 4  2    0   1 5.630818
## 5  2   24   2 4.943767
## 6  3    0   1 4.002471
```

```
print(ggplot(res$Cc) +geom_line(aes(x=time,y=Cc,color=id)))
```



Combining IOV and time varying covariates

Simulating individual covariates

In this example, the weight wt is defined as a piecewise constant individual covariate: it changes from one occaion to another but remains constant within each occasion.

Then, any combination beetween time-varying/constant covariates and time-varying/constant parameters is possible

```
mymodel <- inlineModel("
[LONGITUDINAL]
input = {F, ka, V, Cl}

EQUATION:
Cc = pkmodel(p=F, ka, V, Cl)


[INDIVIDUAL]
input = {wt, age, wt_pop, age_pop}
input = {F_pop, omega_F, gamma_F, beta_F_age, ka_pop, omega_ka, beta_ka_wt,
         V_pop, omega_V, Cl_pop, omega_Cl, gamma_Cl, beta_Cl_wt, beta_Cl_age}

EQUATION:
lwt = log(wt/wt_pop)
lage = log(age/age_pop)
```

```
DEFINITION:
F  = {distribution=logitnormal, reference=F_pop, covariate=lage, coefficient=beta_F_age,
      varlevel={id, id*occ}, sd={omega_F,gamma_F}}
ka = {distribution=lognormal, reference=ka_pop, covariate=lwt, coefficient=beta_ka_wt,
      sd=omega_ka}
V  = {distribution=lognormal, reference=V_pop,  sd=omega_V}
Cl = {distribution=lognormal, reference=Cl_pop,
      covariate={lwt, lage}, coefficient={beta_Cl_wt,beta_Cl_age},
      varlevel={id, id*occ}, sd={omega_Cl,gamma_Cl}}


[COVARIATE]
input = {wt_pop, omega_wt, gamma_wt, age_pop, omega_age}

DEFINITION:
wt  = {distribution=normal, reference=wt_pop, varlevel={id, id*occ},
       sd={omega_wt,gamma_wt}}
age = {distribution=normal, reference=age_pop, sd=omega_age}
")
```

Three occasions are defined in this example:

```
occ <- list(name="occ", time=c(0,12,24))
```

3 individuals are simulated with the following parameters and design:

```
param <- c(F_pop=0.8, omega_F=0.5, gamma_F=0.4, ka_pop=1, omega_ka=0.2,
           V_pop=10, omega_V=0.2, Cl_pop=5, omega_Cl=0.2, gamma_Cl=0.2,
           wt_pop=70, omega_wt=10, gamma_wt=0.1, age_pop=45, omega_age=5,
           beta_F_age=0.5, beta_Cl_wt=1, beta_ka_wt=1, beta_Cl_age=0.4)

trt <- list(time=c(0,12,24), amount=100)

out.cc <- list(name= 'Cc', time=seq(0,36,by=0.1))
out.param <- list(name=c( 'ka', 'V', 'Cl'))
out.cov <- list(name=c('wt', 'age'))

res <-simulx(model     = mymodel,
             parameter = param,
             treatment = trt,
             varlevel  = occ,
             output    = list(out.cc, out.param, out.cov),
             group     = list(size=3, level='covariate'),
             settings  = list(seed=12345))

print(names(res))
```

```
## [1] "Cc"            "group"         "occasion"      "treatment"
## [5] "parameter.iiv" "parameter.iov"
```

Constant and time-varying parameters now include the covariates defined as outputs

```
print(res$parameter.iiv)
```

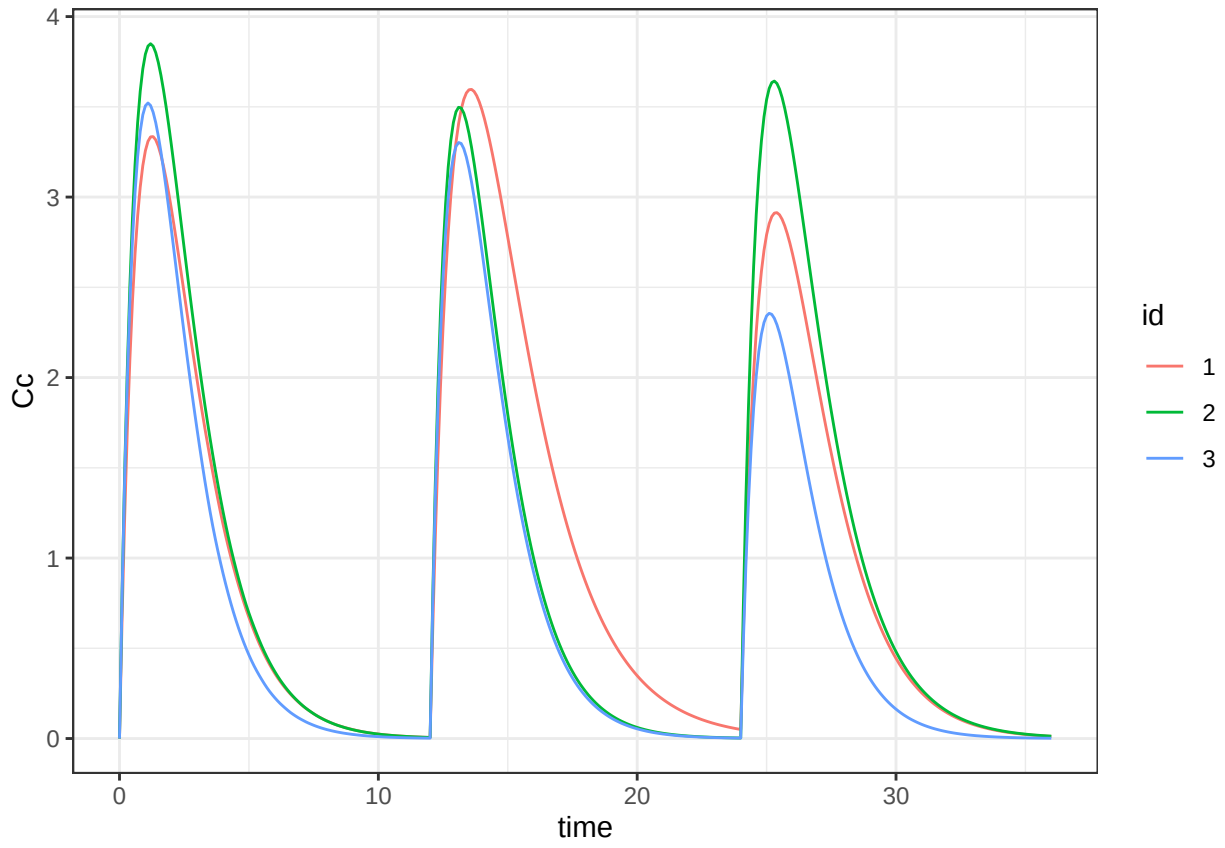
```
##   id        V      age       ka0      Cl0      wt0
## 1  1 8.476292 48.90607 0.9441496 6.758585 59.21119
## 2  2 9.787559 43.56324 1.0941057 6.286581 61.65729
```

```
## 3  3 8.456393 47.24843 0.8147272 7.736379 72.20367
```

```
print(res$parameter.iov)
```

```
##   id time occ       wt       Cl        ka
## 1  1    0   1 59.27273 6.724821 0.7994618
## 2  1   12   2 59.35353 4.243537 0.8005517
## 3  1   24   3 59.17262 5.610613 0.7981115
## 4  2    0   1 61.57464 7.055948 0.9624166
## 5  2   12   2 61.80798 7.835351 0.9660638
## 6  2   24   3 61.59320 6.056024 0.9627068
## 7  3    0   1 72.19293 8.306349 0.8402506
## 8  3   12   2 72.31816 7.890619 0.8417081
## 9  3   24   3 72.05298 8.052512 0.8386217
```

```
print(ggplot(res$Cc) +geom_line(aes(x=time,y=Cc,color=id)))
```



Using existing individual covariates

If, instead of simulating the individual covariates, existing individual covariates are used in the model, then the same model can be used by removing the section [COVARIATE]

```
mymodel <- inlineModel("
[LONGITUDINAL]
input = {F, ka, V, Cl}

EQUATION:
Cc = pkmodel(p=F, ka, V, Cl)
```

```
[INDIVIDUAL]
input = {wt, age, wt_pop, age_pop}
input = {F_pop, omega_F, gamma_F, beta_F_age, ka_pop, omega_ka, beta_ka_wt,
         V_pop, omega_V, Cl_pop, omega_Cl, gamma_Cl, beta_Cl_wt, beta_Cl_age}

EQUATION:
lwt = log(wt/wt_pop)
lage = log(age/age_pop)

DEFINITION:
F  = {distribution=logitnormal, reference=F_pop, covariate=lage, coefficient=beta_F_age,
      varlevel={id, id*occ}, sd={omega_F,gamma_F}}
ka = {distribution=lognormal, reference=ka_pop, covariate=lwt, coefficient=beta_ka_wt, sd=omega_ka}
V  = {distribution=lognormal, reference=V_pop,  sd=omega_V}
Cl = {distribution=lognormal, reference=Cl_pop,
      covariate={lwt, lage}, coefficient={beta_Cl_wt,beta_Cl_age},
      varlevel={id, id*occ}, sd={omega_Cl,gamma_Cl}}
")
```

Both time varying covariates and constant covariates should be defined in a data frame

```
covariate <- inlineDataFrame("
id time  wt  age
1   0    60   50
1   12   65   50
1   24   70   50
2   0    75   60
2   18   75   60
3   0    80   55
3   24   90   55
")
```

Note that it is **mandatory** to also define occasions. These occasions should be consistent with those defined by the time varying covariates

```
occ <- inlineDataFrame("
id time occ
1   0    1
1   12   2
1   24   3
2   0    1
2   18   2
3   0    1
3   24   3
")
```

```
res <-simulx(model     = mymodel,
             parameter = list(param,covariate),
             treatment = trt,
             varlevel  = occ,
             output    = list(out.cc, out.param, out.cov),
             settings  = list(seed=12345))

print(res$parameter.iiv)
```

```
##   id         V age       ka0      Cl0
## 1  1 11.309684  50 1.1691099 6.646691
## 2  2  8.476292  60 0.9441496 6.758585
## 3  3  9.787559  55 1.0941057 6.286581
```

```
print(res$parameter.iov)
```

```
##   id time occ wt        Cl       ka
## 1  1    0   1 60  6.476148 1.002094
## 2  1   12   2 65  6.472403 1.085602
## 3  1   24   3 70  7.047349 1.169110
## 4  2    0   1 75  9.234267 1.011589
## 5  2   18   2 75  5.819130 1.011589
## 6  3    0   1 80 10.063302 1.250407
## 7  3   24   3 90  9.713916 1.406707
```

Defining groups - part I

R scripts: groupI.R

Introduction

It is possible to define groups for several purposes.

1. We can define several groups (or arms) with different values of the parameters, different values of the regression variables, different treatments and/or different outputs (see the example below).
2. We can create groups that consist of several replicates of the longitudinal data, several individuals sharing or not the same covariates, and/or several populations.
3. We can combine these features and create groups of different sizes, with different levels of randomization and different parameters, treatments, outputs, ... (see the examples of the next article)

Example

We will use in this example a very simple PK model for iv administration:

```
pk.model <- inlineModel("
[LONGITUDINAL]
input = {V, k}
EQUATION:
C = pkmodel(V,k)
")
```

We create here three groups with the same PK parameters and the same output, but with three different dosage regimens: 50mg every 6 hours, 100mg every 12 hours, and 150mg every 18 hours.

treatment is not anymore an input argument of `simulx` since it is defined in each element of the list group:

```
adm1 <- list(time=seq(0,to=66,by=6), amount=50)
adm2 <- list(time=seq(0,to=66,by=12), amount=100)
adm3 <- list(time=seq(0,to=66,by=18), amount=150)
g1 <- list(treatment=adm1);
g2 <- list(treatment=adm2);
g3 <- list(treatment=adm3);

C <- list(name='C', time=seq(0, 100, by=1))
p <- c(V=10, k=0.2)


res1 <- simulx(model     = pk.model,
               parameter = p,
               output    = C,
               group     = list(g1,g2,g3))
```

Here, res1 has a unique element C: res1$C is a data frame with three columns, id, time and C where id takes the values 1, 2 and 3:

```
print(res1$C[1:4,])
```

```
##   id group time        C
## 1  1     1    0 5.000000
```

```
## 2  1     1    1 4.093654
## 3  1     1    2 3.351600
## 4  1     1    3 2.744058
```

```
print(res1$C[102:105,])
```

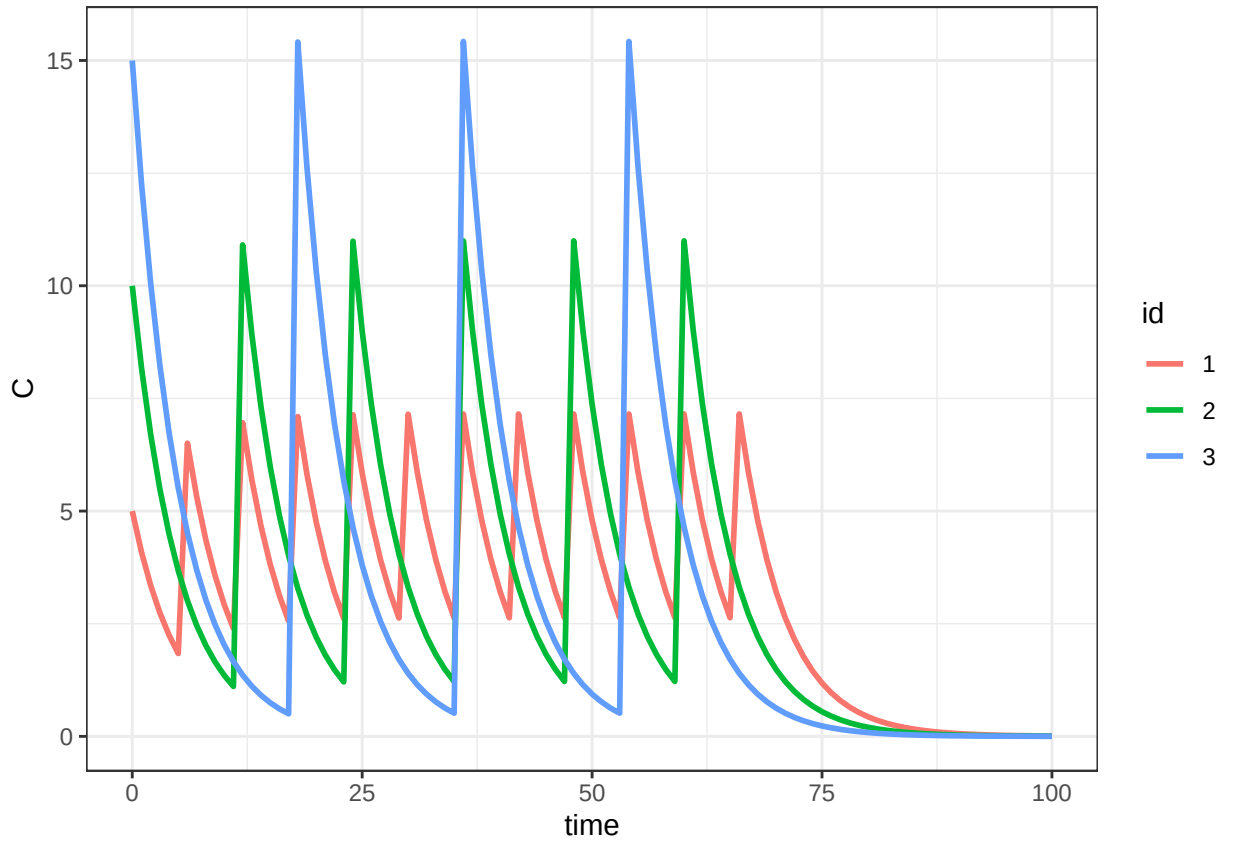
```
##     id group time        C
## 102  2     2    0 10.000000
## 103  2     2    1  8.187308
## 104  2     2    2  6.703200
## 105  2     2    3  5.488116
```

```
print(res1$C[203:206,])
```

```
##     id group time        C
## 203  3     3    0 15.000000
## 204  3     3    1 12.280961
## 205  3     3    2 10.054801
## 206  3     3    3  8.232175
```

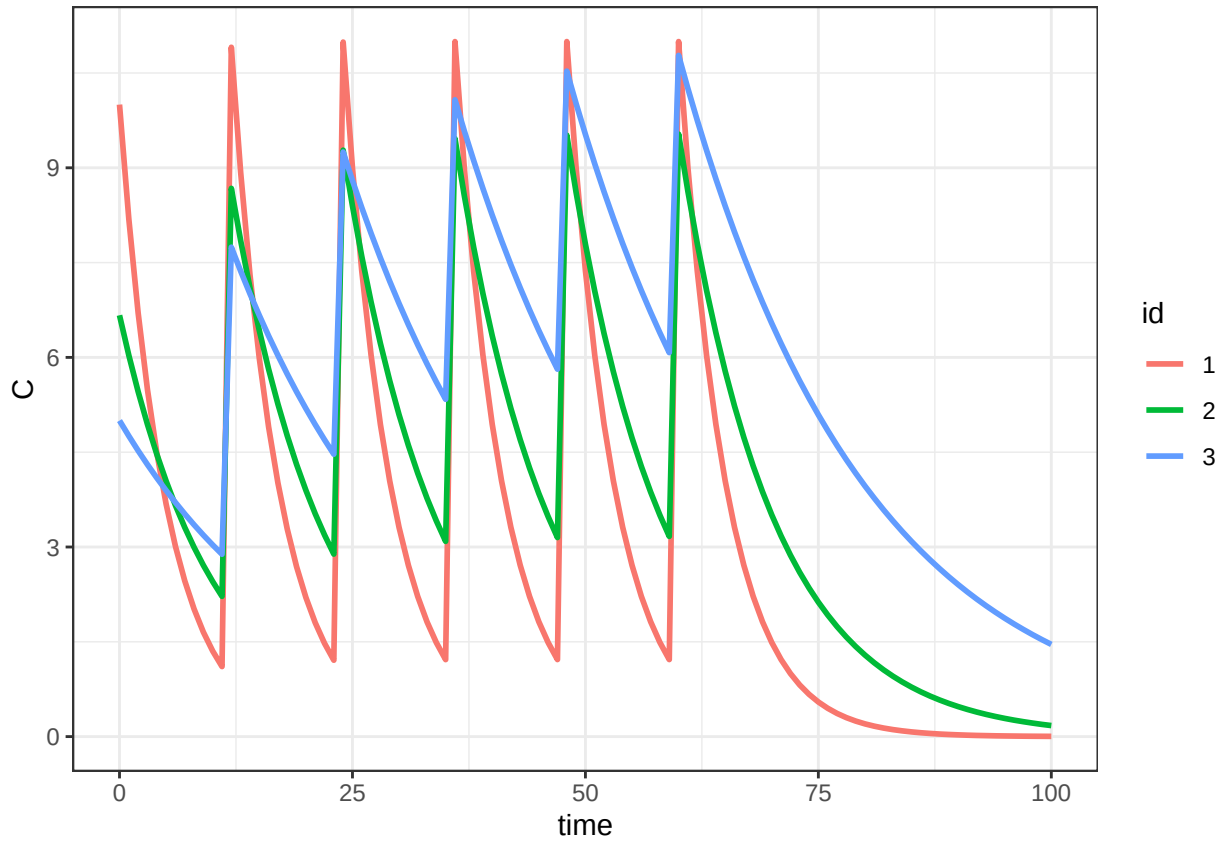
We can plot the concentration C using one colour per group (i.e. per individual):

```
print(ggplot(data=res1$C, aes(x=time, y=C, colour=id)) + geom_line(size=1))
```



We can instead define the same treatment, but different PK parameter values for the three groups. In this case, parameter is not anymore an input argument of `simulx` since it is defined in each element of the list group:

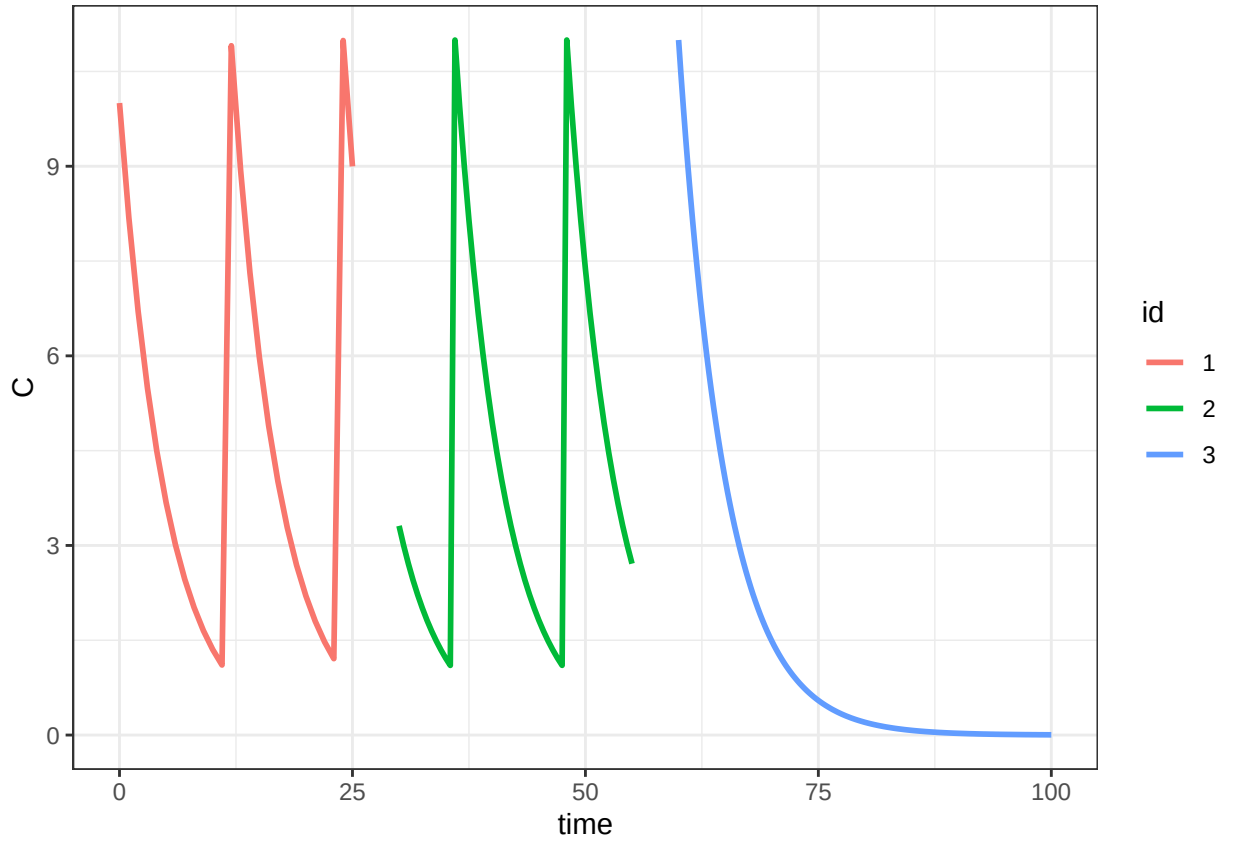
```
adm <- list(time=seq(0,to=66,by=12), amount=100)

g1 <- list(parameter=c(V=10, k=0.2))
g2 <- list(parameter=c(V=15, k=0.1))
g3 <- list(parameter=c(V=20, k=0.05))

res2 <- simulx(model     = pk.model,
               treatment = adm,
               output    = C,
               group     = list(g1,g2,g3))

print(ggplot(data=res2$C, aes(x=time, y=C, colour=id)) + geom_line(size=1))
```



We can instead define different outputs for the three groups:

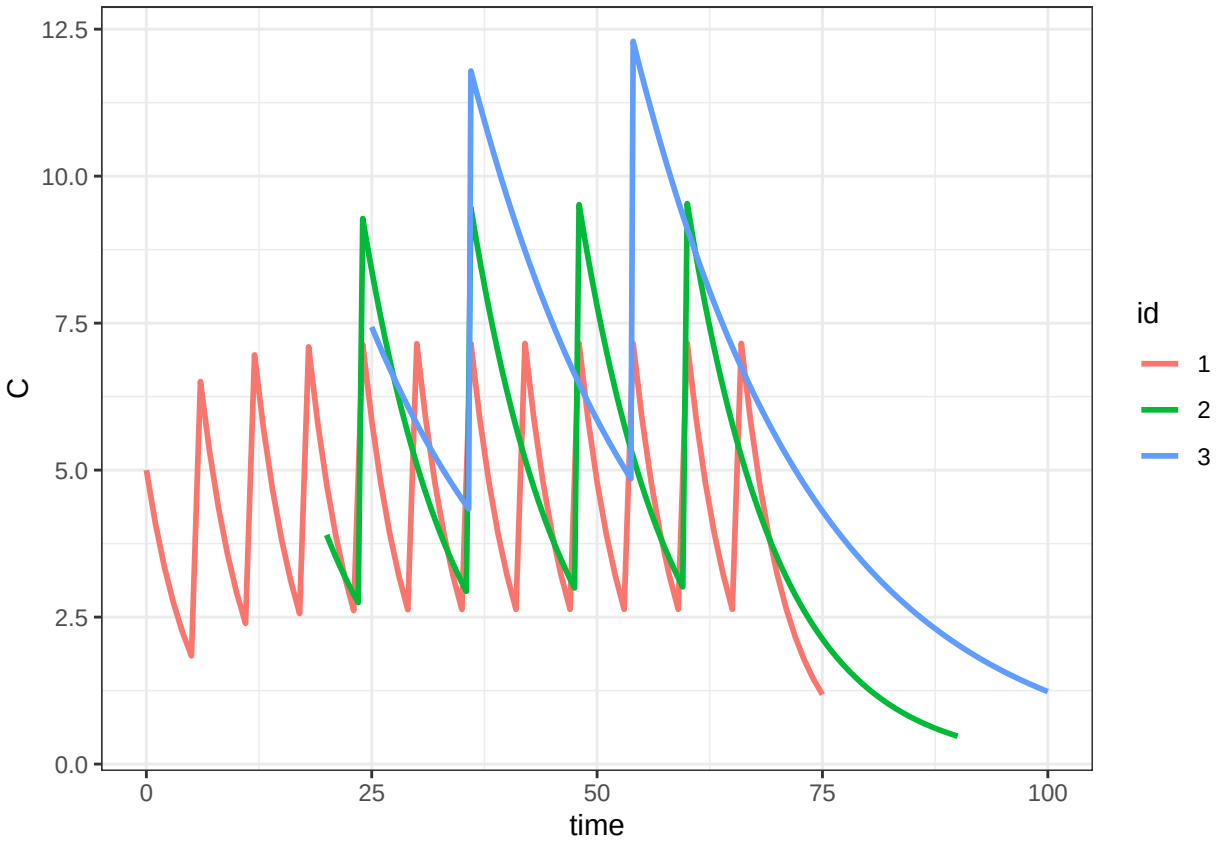
```
adm <- list(time=seq(0,to=66,by=12), amount=100)
p <- c(V=10, k=0.2)
C1 <- list(name='C', time=seq(0,  25, by=1))
C2 <- list(name='C', time=seq(30, 55, by=0.5))
C3 <- list(name='C', time=seq(60,100, by=0.25))
g1 <- list(output=C1)
g2 <- list(output=C2)
g3 <- list(output=C3)

res3 <- simulx(model     = pk.model,
               treatment = adm,
```

```
               parameter = p,
               group     = list(g1,g2,g3))

print(ggplot(data=res3$C, aes(x=time, y=C, colour=id)) + geom_line(size=1))
```



It is also possible to combine these different features by defining different treatments, different outputs and different parameter values in the three groups:

```
adm1 <- list(time=seq(0,to=66,by=6), amount=50)
adm2 <- list(time=seq(0,to=66,by=12), amount=100)
adm3 <- list(time=seq(0,to=66,by=18), amount=150)
p1 <- c(V=10, k=0.2)
p2 <- c(V=15, k=0.1)
p3 <- c(V=20, k=0.05)
C1 <- list(name='C', time=seq(0,  75, by=1))
C2 <- list(name='C', time=seq(20, 90, by=0.5))
C3 <- list(name='C', time=seq(25,100, by=0.25))
g1 <- list(output=C1, treatment=adm1, parameter=p1)
g2 <- list(output=C2, treatment=adm2, parameter=p2)
g3 <- list(output=C3, treatment=adm3, parameter=p3)

res4 <- simulx(model     = pk.model,
               group     = list(g1,g2,g3))

print(ggplot(data=res4$C, aes(x=time, y=C, colour=id)) + geom_line(size=1))
```


12.5
10.0
7.5
5.0
2.5
0.0
C
0
25
50
75
100
time
id
1
2
3

Defining groups - part II

R scripts: groupII.R

Mlxtran codes: model/groupII1.txt ; model/groupII2.txt

Introduction

It is possible to define groups for several purposes.

1. We can define several groups (or arms) with different values of the parameters, different values of the regression variables, different treatments and/or different outputs (see the examples of the previous article)
2. We can create groups that consist of several replicates of the longitudinal data, several individuals sharing or not the same covariates, and/or several populations.
3. We can combine these features and create groups of different sizes, with different levels of randomization and different parameters, treatments, outputs, ... (see the example below).

Definition of the size of a group is unambiguous when there is a unique section `[LONGITUDINAL]` in the model: size=N means that we want to generate N replicates of the longitudinal data defined in section `[LONGITUDINAL]`.

Problem arise when there are several sections. Consider for example a model with sections `[INDIVIDUAL]` and `[LONGITUDINAL]`:

Then, size=N turns out to be ambiguous: does it mean that N replicates should be generated using a single set of simulated individual parameters, or does it mean that N vectors of individual parameters and one replicate per individual should be generated?

To avoid any possible confusion, we propose to explicity define the levels of randomization and provide the number of simulations per level (i.e. per section of the Mlxtran model).

For instance,

```
group = list(size=c(1,5), level=c('individual','longitudinal'))
```

means that

- one single set of individual parameters will be drawn with the model defined in the section `[INDIVIDUAL]`,
- five replicates of the data defined in the section `[LONGITUDINAL]` will be generated.

```
group = list(size=c(5,1), level=c('individual','longitudinal'))
```

means that

- five sets of individual parameters will be drawn with the model defined in the section `[INDIVIDUAL]`,
- one replicate per individual of the data defined in the section `[LONGITUDINAL]` will be generated.

```
group = list(size=c(5,3), level=c('individual','longitudinal'))
```

means that

- five sets of individual parameters will be drawn with the model defined in the section `[INDIVIDUAL]`,
- three replicates per individual of the data defined in the section `[LONGITUDINAL]` will be generated.

If you are really lazy, some defaults are available. Consider for example a model with sections `[COVARIATE]`, `[INDIVIDUAL]` and `[LONGITUDINAL]` (in this order). Then,

1. the default size for each level is 1. Then,

```
group = list(size=5, level='longitudinal')
```

is equivalent to

```
group = list(size=c(1,1,5), level=c('covariate','individual','longitudinal'))
```

and

```
group = list(size=5, level='covariate')
```

is equivalent to

```
group = list(size=c(5,1,1), level=c('covariate','individual','longitudinal'))
```

2. Levels defined in the Mlxtran code are used if they are not provided as an element of group. Then,

```
group = list(size=c(3,1,2))
```

is equivalent to

```
group = list(size=c(3,1,2), level=c('covariate','individual','longitudinal'))
```

In such case, the length of vector size should be exactly the number of sections of the Mlxtran code.

Examples

Example 1

Let us start start with the model for longitudinal data implemented in `groupII1.txt`

We will use the input argument group of `simulix` for simulating 5 replicates of the same model.

```
adm <- list(time=seq(0,to=66,by=12), amount=100)
C <- list(name='C', time=seq(0, 100, by=1))
y <- list(name='y', time=seq(0, 100, by=12))
p <- c(V=10, k=0.2, a=0.1)
g <- list(size=5, level='longitudinal')

res1 <- simulx(model     = "model/groupII1.txt",
               treatment = adm,
               parameter = p,
               output    = list(C,y),
               group     = g,
               settings  = list(seed = 12345))
```

Here, res1 is a list of 2 data frames, C and y

```
names(res1)
```

```
## [1] "C"         "y"         "group"     "treatment"
```

```
names(res1$C)
```

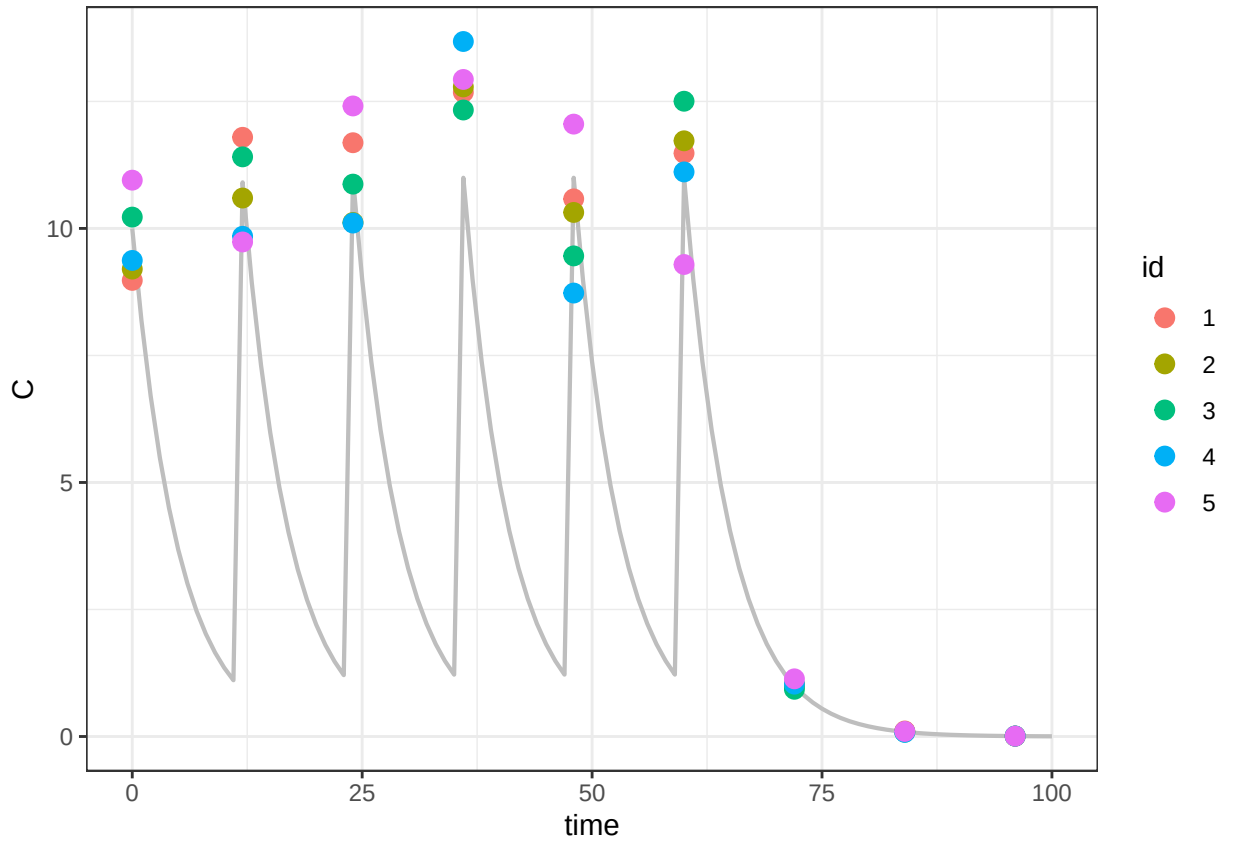
```
## [1] "id"   "time" "C"
```

```
names(res1$y)
```

```
## [1] "id"   "time" "y"
```

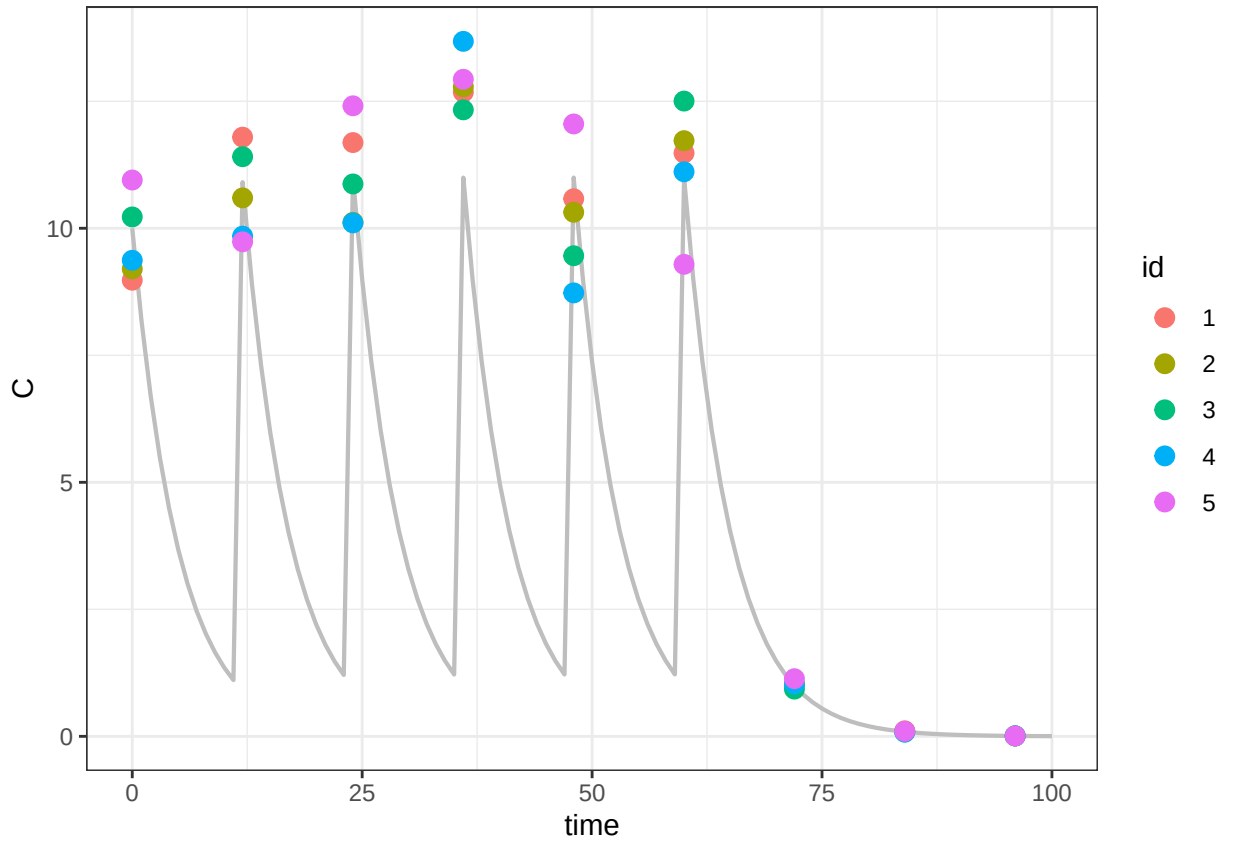
Let us plot the predicted concentration C (wich is the same for the 5 replicates), and the 5 replicates of simulated data (y_j):

```
print(ggplot() + geom_line(data=res1$C, aes(x=time, y=C), colour="grey", size=0.75) +
        geom_point(data=res1$y, aes(x=time, y=y, colour=id), size=3))
```



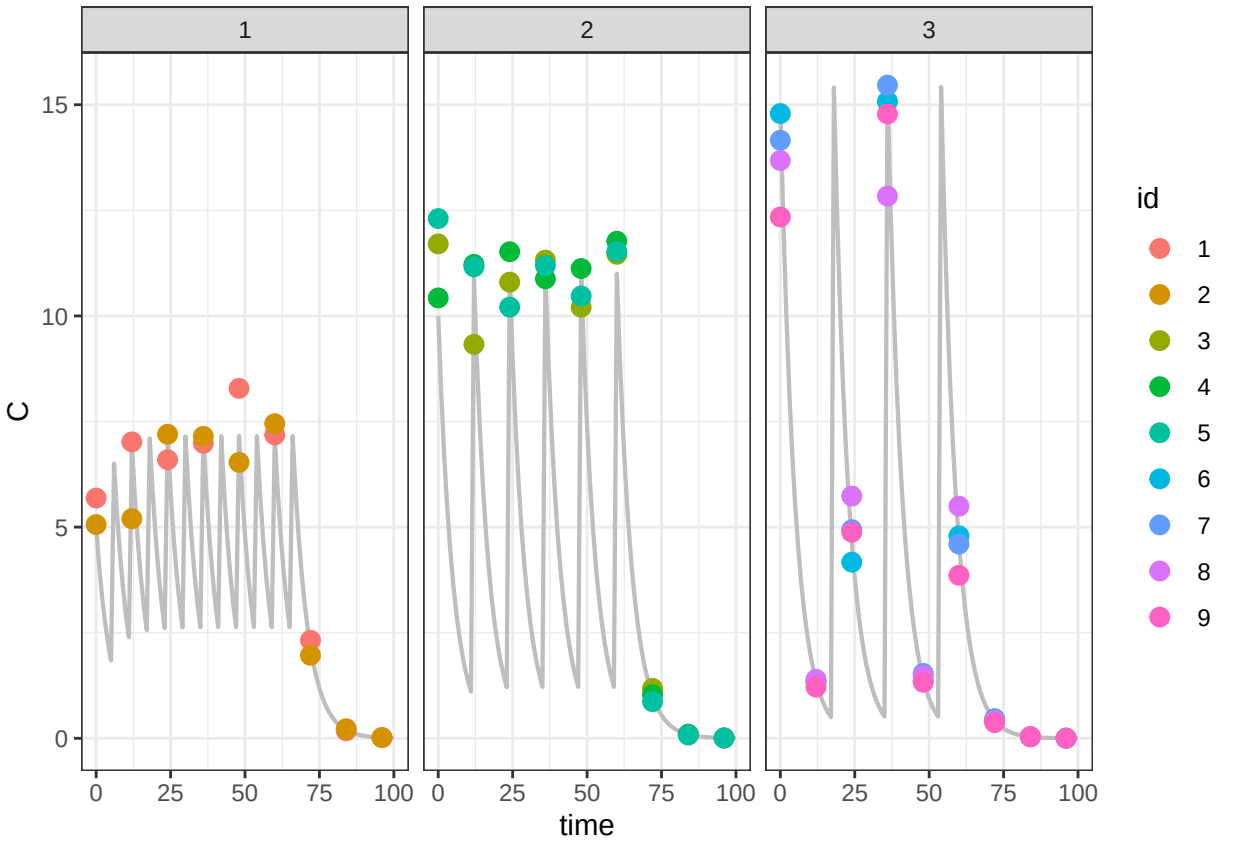
Remark: since there is only one group in this example, it would be equivalent to define the treatment, the parameter values and the outputs as elements of group, instead of input arguments of `simulx`:

```
g <- list(treatment=adm, parameter=p, output=list(C,y), size=5, level='longitudinal')

res3 <- simulx(model     = "model/groupII1.txt",
               group     = g,
               settings  = list(seed = 12345))

print(ggplot() + geom_line(data=res3$C, aes(x=time, y=C), colour="grey", size=0.75) +
        geom_point(data=res3$y, aes(x=time, y=y, colour=id), size=3))
```



With the same model, we can now define three groups with different sizes and different treatments. The three groups share the same parameter values and the same outputs.

```
adm1 <- list(time=seq(0,to=66,by=6), amount=50)
adm2 <- list(time=seq(0,to=66,by=12), amount=100)
adm3 <- list(time=seq(0,to=66,by=18), amount=150)
g1 <- list(treatment=adm1, size=2, level='longitudinal')
g2 <- list(treatment=adm2, size=3, level='longitudinal')
g3 <- list(treatment=adm3, size=4, level='longitudinal')

C <- list(name='C', time=seq(0, 100, by=1))
y <- list(name='y', time=seq(0, 100, by=12))
p <- c(V=10, k=0.2, a=0.1)


res4 <- simulx(model     = "model/groupII1.txt",
               parameter = p,
               output    = list(C,y),
               group     = list(g1,g2,g3))

print(ggplot() + geom_line(data=res4$C, aes(x=time, y=C), colour="grey", size=0.75) +
        geom_point(data=res4$y, aes(x=time, y=y, colour=id), size=3)+
        facet_grid(. ~ group))
```



Example 2

Model `groupII2.txt` now includes a section `[INDIVIDUAL]`

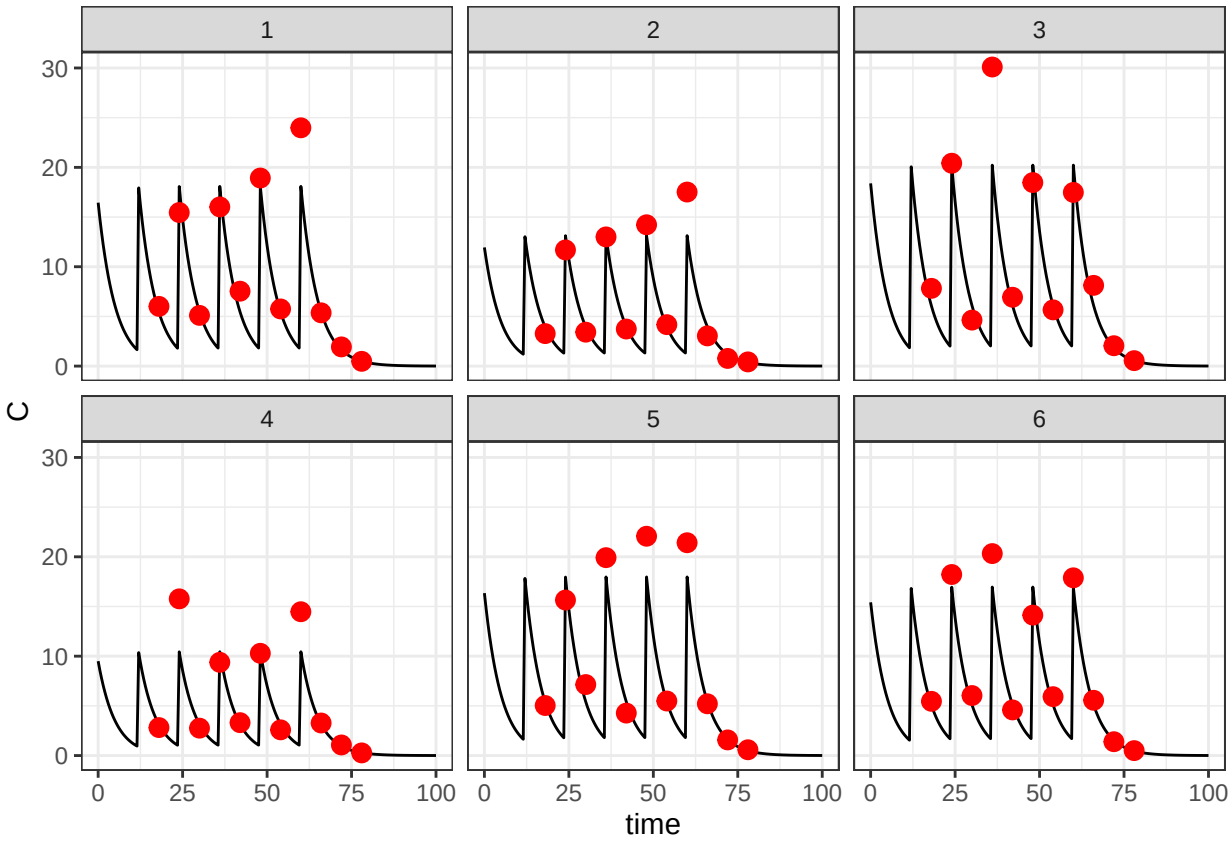
We can then use this model for simulating six individuals with different individual parameter values by setting level='individual'.

```
adm <- list(time=seq(0,66,by=12), amount=100)
y <- list(name="y", time=seq(18, 80, by=6))
C <- list(name="C", time=seq(0,100, by=0.5))
V <- list(name="V")
p <- c(V_pop=10, omega_V=0.3, w=50, k=0.2, a=0.2)

g <- list(size  = c(6,1),
          level = c('individual','longitudinal'))

res5 <- simulx(model    = "model/groupII2.txt",
               output    = list(C,y,V),
               parameter = p,
               treatment = adm,
               group     = g,
               settings  = list(seed=123456))

print(res5$parameter)
```

```
##   id         V
## 1  1  6.081135
```

```
## 2  2  8.376298
## 3  3  5.437911
## 4  4 10.527334
## 5  5  6.118600
## 6  6  6.485735
```

```
print(ggplot()  +
  geom_line(data=res5$C, aes(x=time, y=C), colour="black", size=0.5) +
  geom_point(data=res5$y, aes(x=time, y=y), colour="red", size=3)+
    facet_wrap( ~ id))
```



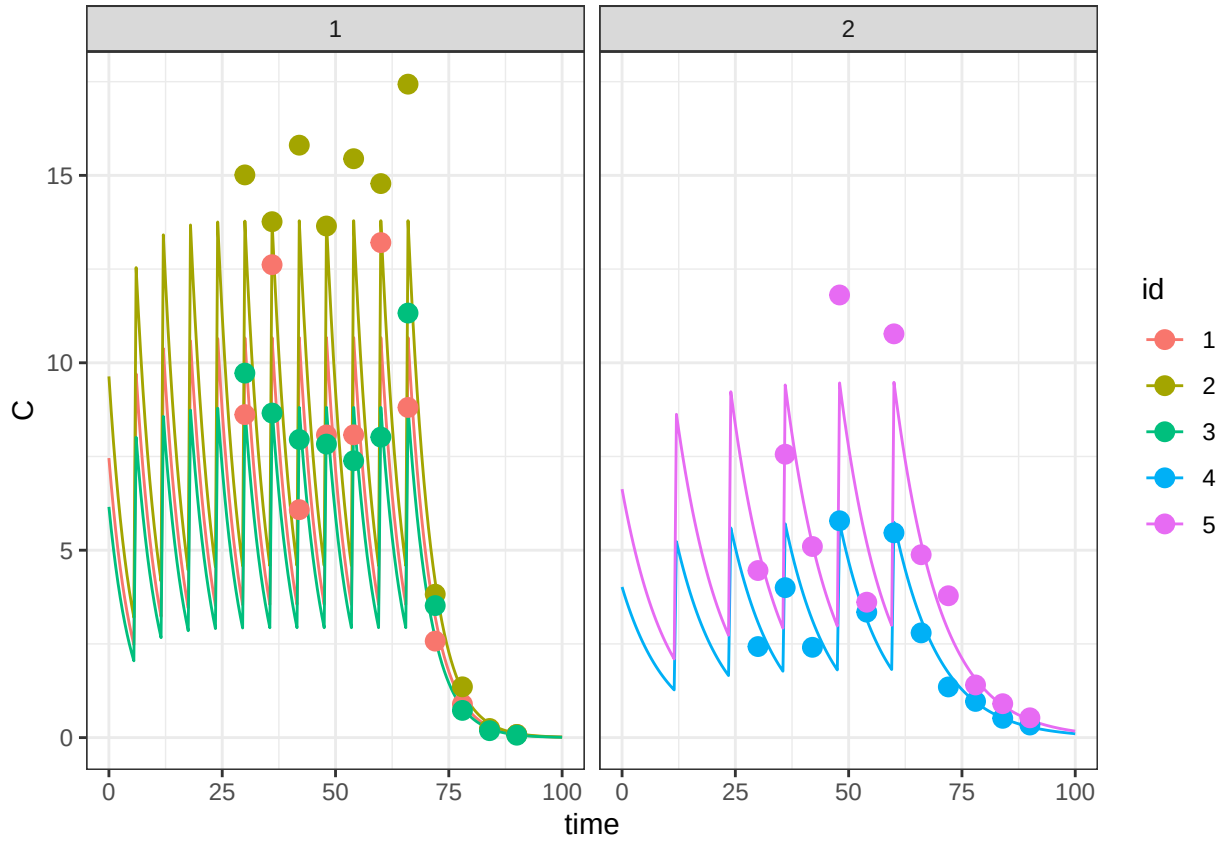
We can also use group for defining two groups with different treatments, different parameter values, and different sizes.

```
adm1 <- list(time=seq(0,66,by=6),  amount=50)
adm2 <- list(time=seq(0,66,by=12), amount=100)
y <- list(name="y", time=seq(30, 90, by=6))
C <- list(name="C", time=seq(0,100, by=0.5))
V <- list(name="V")
p1 <- c(V_pop=10, omega_V=0.3, w=50, k=0.2, a=0.2)
p2 <- c(V_pop=20, omega_V=0.3, w=75, k=0.1, a=0.2)

g1 <- list(treatment = adm1,
           parameter = p1,
           size      = c(3,1),
           level     = c('individual','longitudinal'))
g2 <- list(treatment = adm2,
```

```
            parameter = p2,
            size      = c(2,1),
            level     = c('individual','longitudinal'))

res6 <- simulx(model    = "model/groupII2.txt",
               output   = list(C,y,V),
               group    = list(g1,g2),
               settings = list(seed=123123))

print(res6$parameter)
```

```
##   id group         V
## 1  1     1  6.704433
## 2  2     1  5.188521
## 3  3     1  8.122303
## 4  4     2 24.882190
## 5  5     2 15.085256
```

```
print(ggplot()  +
        geom_line( data=res6$C, aes(x=time, y=C, colour=id), size=0.5) +
        geom_point(data=res6$y, aes(x=time, y=y, colour=id), size=3) +
        facet_grid(. ~ group))
```



Simulating several replicates of the same experiment

R scripts: replicate.R

We will use the following joint model for continuous PK and time-to-event data to illustrate this feature of `simulx`:

```
modelPK <- inlineModel("
[LONGITUDINAL]
input={V,Cl,alpha, beta,b}

EQUATION:
C = pkmodel(V, Cl)
h = alpha*exp(beta*C)
g = b*C

DEFINITION:
y = {distribution=normal, prediction=C, sd=g}
e = {type=event, maxEventNumber=1, rightCensoringTime=30, hazard=h}

;-----------------------------------------
[INDIVIDUAL]
input={V_pop,Cl_pop,omega_V,omega_Cl}

DEFINITION:
V     = {distribution=lognormal,   prediction=V_pop,    sd=omega_V}
Cl    = {distribution=lognormal,   prediction=Cl_pop,   sd=omega_Cl}
")
```

We can define the design and the parameters values as usual

```
adm  <- list(amount=100, time=0)
p <- c(V_pop=10, Cl_pop=1, omega_V=0.2, omega_Cl=0.2, alpha=0.02, beta=0.1, b=0.1)
g <- list(size=5, level='individual')

out.y <- list(name='y', time=seq(0,to=25,by=5))
out.e <- list(name='e', time=0)
out.p <- c("V", "Cl")
out   <- list(out.p, out.y, out.e)
```

The number of replicates is defined with the input argument `nrep`:

```
res1 <- simulx(model=modelPK, treatment=adm, parameter=p, output=out, group=g, nrep=3)
print(head(res1$parameter))
```

```
##   rep id         V        Cl
## 1   1  1 10.685187 1.2643551
## 2   1  2 10.284623 1.1552405
## 3   1  3  9.629121 0.7018032
## 4   1  4  9.318872 0.7399398
## 5   1  5  9.866776 0.9907273
## 6   2  1 13.658256 1.2845700
```

```
print(summary(res1$parameter))
```

```
##  rep   id          V                Cl
```

```
##  1:5   1:3   Min.   : 8.987   Min.   :0.7018
##  2:5   2:3   1st Qu.: 9.433   1st Qu.:0.8378
##  3:5   3:3   Median : 9.867   Median :0.9907
##        4:3   Mean   :10.633   Mean   :1.0208
##        5:3   3rd Qu.:11.374   3rd Qu.:1.2098
##              Max.   :14.733   Max.   :1.3884
```

When summary statistics are defined for some output, they are computed over each replicate

```
out.p2 <- list(name=c("V", "Cl"), FUN="quantile", probs=c(0.05, 0.5, 0.95))
out2   <- list(out.p, out.y, out.e)
res2 <- simulx(model=modelPK, treatment=adm, parameter=p, output=out2, group=g, nrep=3)
print(res2$parameter)
```

```
##    rep id         V        Cl
## 1    1  1 10.656955 1.5337688
## 2    1  2  9.035973 1.3964386
## 3    1  3 10.998969 1.1507265
## 4    1  4  9.825668 0.8495700
## 5    1  5  8.942468 0.8747042
## 6    2  1  9.241310 1.1403039
## 7    2  2  8.738431 1.0390045
## 8    2  3  8.593960 1.0674834
## 9    2  4  8.160384 1.3370119
## 10   2  5  7.675122 1.0471355
## 11   3  1 10.797256 0.8452830
## 12   3  2  8.009274 1.2550288
## 13   3  3  9.514614 1.0880918
## 14   3  4  9.958638 0.8695804
## 15   3  5 11.799436 1.1524367
```

Groups can be defined for each replicate:

```
g1 <- list(size=3, level="individual", treatment=list(amount=100, time=0))
g2 <- list(size=3, level="individual", treatment=list(amount=50,  time=0))
g  <- list(g1, g2)
res3 <- simulx(model=modelPK, parameter=p, output=out, group=g, nrep=2)
print(res3$parameter, digits=3)
```

```
##    rep id group     V    Cl
## 1    1  1     1 13.16 0.737
## 2    1  2     1 10.70 1.443
## 3    1  3     1  8.38 1.041
## 4    1  4     2  9.50 1.326
## 5    1  5     2  6.38 0.849
## 6    1  6     2 13.08 0.931
## 7    2  1     1  6.04 1.224
## 8    2  2     1 11.05 1.267
## 9    2  3     1 12.53 0.689
## 10   2  4     2  9.52 0.927
## 11   2  5     2 12.38 0.793
## 12   2  6     2  7.79 1.067
```

```
print(head(res3$e))
```

```
##   rep id group      time e
## 1   1  1     1  0.000000 0
```

```
## 2   1  1     1  8.474182 1
## 3   1  2     1  0.000000 0
## 4   1  2     1 30.000000 0
## 5   1  3     1  0.000000 0
## 6   1  3     1 22.760047 1
```

Computing summary statistics

R scripts: statout.R

We will use the following joint model for continuous PK and time-to-event data to illustrate this feature of `simulx`:

```
modelPK <- inlineModel("
[LONGITUDINAL]
input={V,Cl,alpha, beta,b}

EQUATION:
C = pkmodel(V, Cl)
h = alpha*exp(beta*C)
g = b*C

DEFINITION:
y = {distribution=normal, prediction=C, sd=g}
e = {type=event, maxEventNumber=1, rightCensoringTime=30, hazard=h}

;-----------------------------------------
[INDIVIDUAL]
input={V_pop,Cl_pop,omega_V,omega_Cl}

DEFINITION:
V     = {distribution=lognormal,   prediction=V_pop,    sd=omega_V}
Cl    = {distribution=lognormal,   prediction=Cl_pop,   sd=omega_Cl}
")
```

We can define the design and the parameters values as usual

```
adm  <- list(amount=100, time=0)
p <- c(V_pop=10, Cl_pop=1, omega_V=0.2, omega_Cl=0.2, alpha=0.02, beta=0.1, b=0.1)
g <- list(size=100, level='individual')
s <- list(seed=123456)
out.y <- list(name='y', time=seq(0,to=25,by=5))
out.e <- list(name='e', time=0)
```

We can compute the mean and standard deviation of the individual parameters V and Cl, compute over the $N = 100$ individuals:

```
out.p <- list(name=c("V", "Cl"), FUN= c("mean", "sd"))

out   <- list(out.p, out.y, out.e)
res1a <- simulx(model=modelPK, treatment=adm, parameter=p, output=out, group=g, settings=s)

res1a$parameter
```

```
##     V.mean  Cl.mean     V.sd     Cl.sd
## 1 9.867676 1.024325 1.931853 0.1932103
```

Available functions are mean, sd, var, median, quantile.

Let us compute now the percentiles of order 5%, 50% and 95%:

```
out.p <- list(name=c("V", "Cl"), FUN="quantile", probs=c(0.05, 0.5, 0.95))
out   <- list(out.p, out.y, out.e)
```

```
res1b <- simulx(model=modelPK, treatment=adm, parameter=p, output=out, group=g, settings=s)
res1b$parameter
```

```
##       V.p5   V.p50    V.p95    Cl.p5    Cl.p50   Cl.p95
## 1 7.006682 9.90696 13.63716 0.750047 0.9940518 1.372121
```

It is possible to compute several functions with a single call:

```
out.p <- list(name=c("V", "Cl"),  FUN=c("mean","quantile"), probs=c(0.05,0.95))
out   <- list(out.p, out.y, out.e)
res1c <- simulx(model=modelPK, treatment=adm, parameter=p, output=out, group=g, settings=s)
res1c$parameter
```

```
##     V.mean  Cl.mean     V.p5    V.p95    Cl.p5   Cl.p95
## 1 9.867676 1.024325 7.006682 13.63716 0.750047 1.372121
```

Remark: we would get the same results by running first `simulx` with V and Cl as outputs, and computing then the summary statistics using `statmlx`:

```
out.p <- c("V", "Cl")
out   <- list(out.p, out.y, out.e)
res1d <- simulx(model=modelPK, treatment=adm, parameter=p, output=out, group=g, settings=s)
statmlx(res1d$parameter, FUN=c("mean","quantile"), probs=c(0.05,0.95))
```

```
##     V.mean  Cl.mean     V.p5    V.p95    Cl.p5   Cl.p95
## 1 9.867676 1.024325 7.006682 13.63716 0.750047 1.372121
```

Summary statistics can also be computed for the observations. In this example, we compute

- the mean and the percentiles of order 5% and 95% of the individual PK parameters,
- the mean, the standard deviation and and the percentiles of order 5% and 95% of the PK data,
- the mean survival rate at time 10 and 20 (note that FUN="mean" is used by default)

```
out.p <- list(name=c("V", "Cl"),  FUN=c("mean","quantile"), probs=c(0.05,0.95))
out.y <- list(name='y', time=seq(0,to=25,by=5), FUN = c("mean", "sd", "quantile"),
              probs = c(0.05, 0.95))
out.e <- list(name='e', time=0, type="event", surv.time=c(10,20))
out   <- list(out.p, out.y, out.e)
res2 <- simulx(model=modelPK, treatment=adm, parameter=p, output=out, group=g, settings=s)
res2$y
```

```
##   time     y.mean      y.sd      y.p5     y.p95
## 1    0 10.5439915 2.2277861 7.3323254 13.990281
## 2    5  6.0566830 0.8914127 4.6959603  7.289806
## 3   10  3.5898067 0.8181192 2.2495514  4.878255
## 4   15  2.1271154 0.6562992 0.9407988  3.214660
## 5   20  1.3220924 0.5658928 0.4853363  2.138741
## 6   25  0.8240538 0.4589827 0.2046777  1.633585
```

```
res2$e
```

```
##   nbEv.mean S10.mean S20.mean
## 1      0.64     0.62     0.42
```

nbEv.mean is the mean number of events per individual during the whole study (i.e. between time 0 and time 30).

When replicates and/or groups are defined, the summary statistics are computed over each group of each replicate.

```
g1 <- list(size=100, level="individual", treatment=list(amount=100, time=0))
g2 <- list(size=100, level="individual", treatment=list(amount=50,  time=0))
g  <- list(g1, g2)
res3a <- simulx(model=modelPK, parameter=p, output=out, group=g, nrep=3, settings=s)
res3a$parameter
```

```
##   rep group    V.mean   Cl.mean     V.p5    V.p95     Cl.p5   Cl.p95
## 1   1     1  9.867676 1.0243249 7.006682 13.63716 0.7500470 1.372121
## 2   1     2  9.962862 1.0257042 6.781151 13.03223 0.6825450 1.377832
## 3   2     1 10.664560 0.9954362 7.663782 14.47422 0.7291493 1.312511
## 4   2     2 10.137444 1.0082019 7.517835 13.02059 0.6959647 1.329253
## 5   3     1  9.662182 1.0292811 6.724148 12.74819 0.6941128 1.359681
## 6   3     2 10.380330 1.0211185 7.576823 13.92189 0.7737851 1.379268
```

```
res3a$e
```

```
##   rep group nbEv.mean S10.mean S20.mean
## 1   1     1      0.64     0.62     0.42
## 2   1     2      0.61     0.75     0.52
## 3   2     1      0.52     0.75     0.64
## 4   2     2      0.51     0.76     0.63
## 5   3     1      0.54     0.66     0.53
## 6   3     2      0.53     0.74     0.60
```

When the simulated data is saved in a data file, the complete data is saved and the summary statistics are returned by `simulx`:

```
res3b <- simulx(model=modelPK, parameter=p, output=out, group=g, nrep=3, settings=s,
                result.file = "res3b.csv")
head(read.table("res3b.csv", header=T, sep=","))
```

```
##   rep id group time      y ytype amount      V     Cl
## 1   1  1     1    0      .     .    100 8.9827 1.2486
## 2   1  1     1    0 12.866     1      . 8.9827 1.2486
## 3   1  1     1    0      0     2      . 8.9827 1.2486
## 4   1  1     1    5   5.69     1      . 8.9827 1.2486
## 5   1  1     1   10 2.5078     1      . 8.9827 1.2486
## 6   1  1     1   15 1.3218     1      . 8.9827 1.2486
```

```
res3b$parameter
```

```
##   rep group    V.mean   Cl.mean     V.p5    V.p95     Cl.p5   Cl.p95
## 1   1     1  9.867676 1.0243249 7.006682 13.63716 0.7500470 1.372121
## 2   1     2  9.962862 1.0257042 6.781151 13.03223 0.6825450 1.377832
## 3   2     1 10.664560 0.9954362 7.663782 14.47422 0.7291493 1.312511
## 4   2     2 10.137444 1.0082019 7.517835 13.02059 0.6959647 1.329253
## 5   3     1  9.662182 1.0292811 6.724148 12.74819 0.6941128 1.359681
## 6   3     2 10.380330 1.0211185 7.576823 13.92189 0.7737851 1.379268
```

```
res3b$e
```

```
##   rep group nbEv.mean S10.mean S20.mean
## 1   1     1      0.64     0.62     0.42
## 2   1     2      0.61     0.75     0.52
## 3   2     1      0.52     0.75     0.64
## 4   2     2      0.51     0.76     0.63
## 5   3     1      0.54     0.66     0.53
```

```
## 6   3     2      0.53     0.74     0.60
```

Reading data files and using data frames

R scripts: data.R

Mlxtran codes: model/groupI.txt ; model/groupII1.txt ; model/groupII2.txt

---

Introduction

Individual parameters and individual designs (dosage regimens and observation times) can be defined in external data files, or as "inline'' data frames.

Function `inlineDataFrame` is extremly useful for converting formatted text into data frames.

Individual parameters defined in a data frame

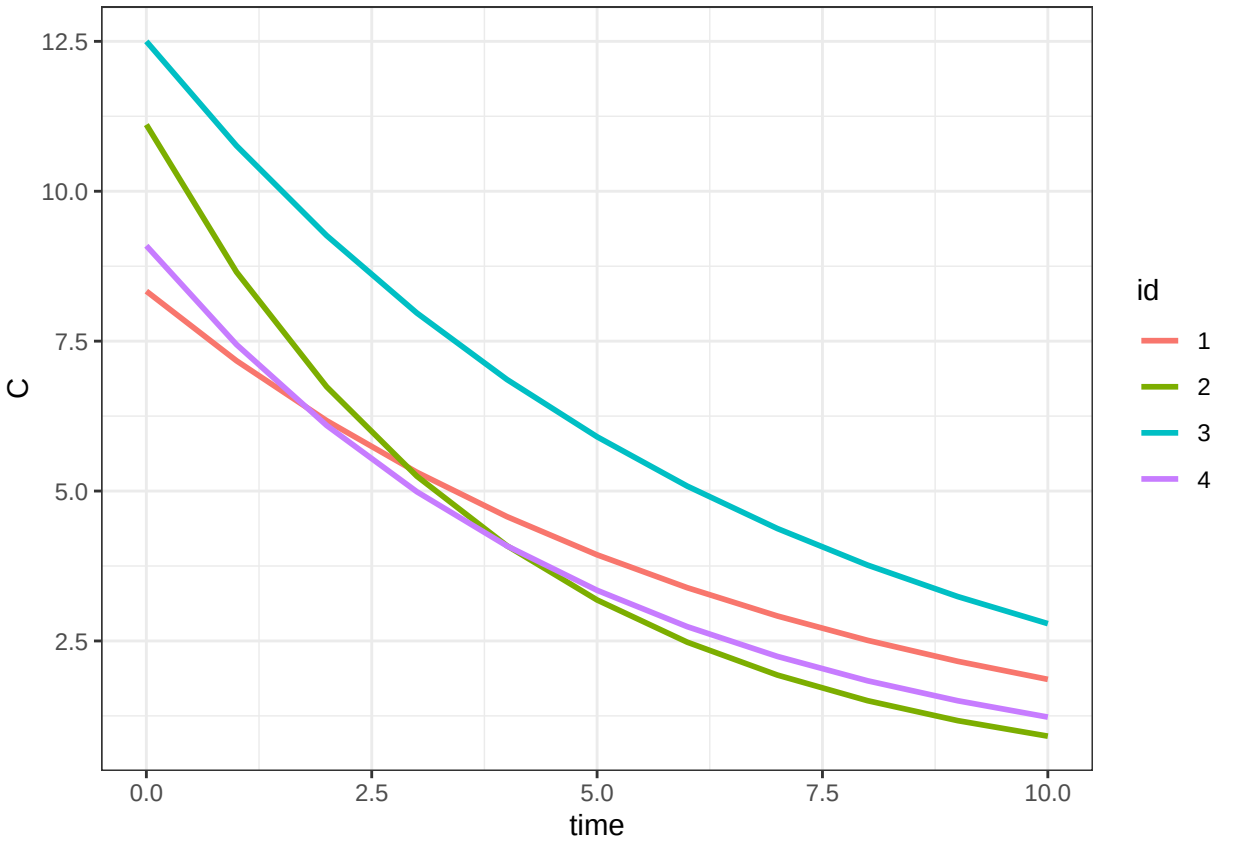
Let us start with the PK model for iv administration implemented in `groupI.txt`:

Suppose that we want to use this model for computing the predicted concentration of 4 individuals with different PK parameters stored in the file `data/data1.txt`.

Column named id (taking the values 1, 2, 3, 4) is required in order to identify these 4 individuals. The names of the two other columns should match with the names of the parameters of the model (i.e. V and k).

We can then load this table and use it as input for `simulx`:

```
p = read.table("data/data1.txt",header=TRUE)

adm <- list(time=0, amount=100)
C <- list(name="C", time=seq(0, 10, by=1))

res1a <- simulx(model     = "model/groupI.txt",
                parameter = p,
                output    = C,
                treatment = adm)

print(ggplot(data=res1a$C, aes(x=time, y=C, colour=id)) + geom_line(size=1))
```



Instead of reading the parameter values in an external data file, this data can be provided directly in the `R` script using the function `inlineDataFrame` which converts formatted text into a data frame:

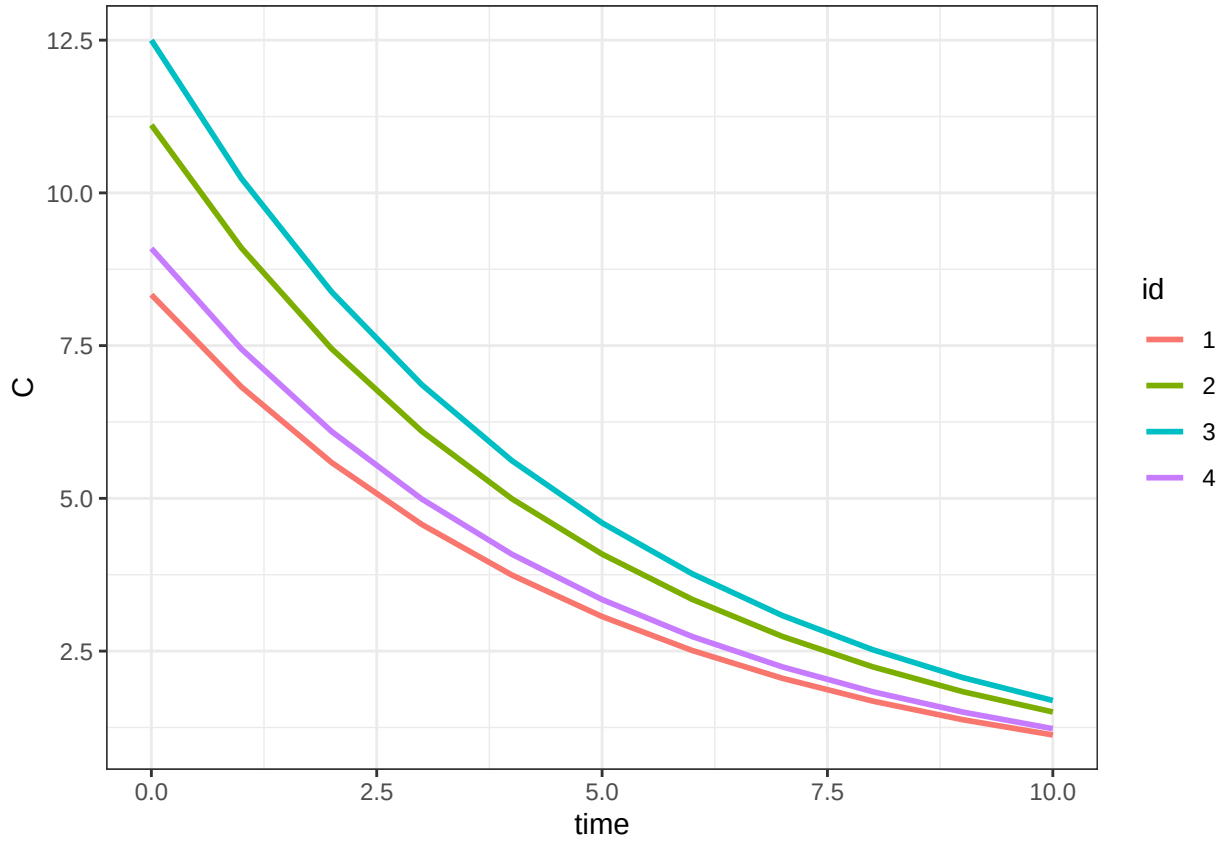
```
p <- inlineDataFrame("
 id     V      k
  1    12   0.15
  2     9   0.25
  3     8   0.15
  4    11   0.20
")

print(p)
```

```
##   id  V    k
## 1  1 12 0.15
## 2  2  9 0.25
## 3  3  8 0.15
## 4  4 11 0.20
```

Assume now that the four individuals share some parameters (k in this example). It is then possible to define the input argument parameter as a list which combines a data frame (for the parameters which are different) and a vector (for the parameters which are shared)

```
V <- inlineDataFrame("
 id     V
  1    12
  2     9
  3     8
```

```
  4    11
")
k <- c(k=0.2)
p <- list(V, k)

res1b <- simulx(model     = "model/groupI.txt",
                parameter = p,
                output    = C,
                treatment = adm)

print(ggplot(data=res1b$C, aes(x=time, y=C, colour=id)) + geom_line(size=1))
```

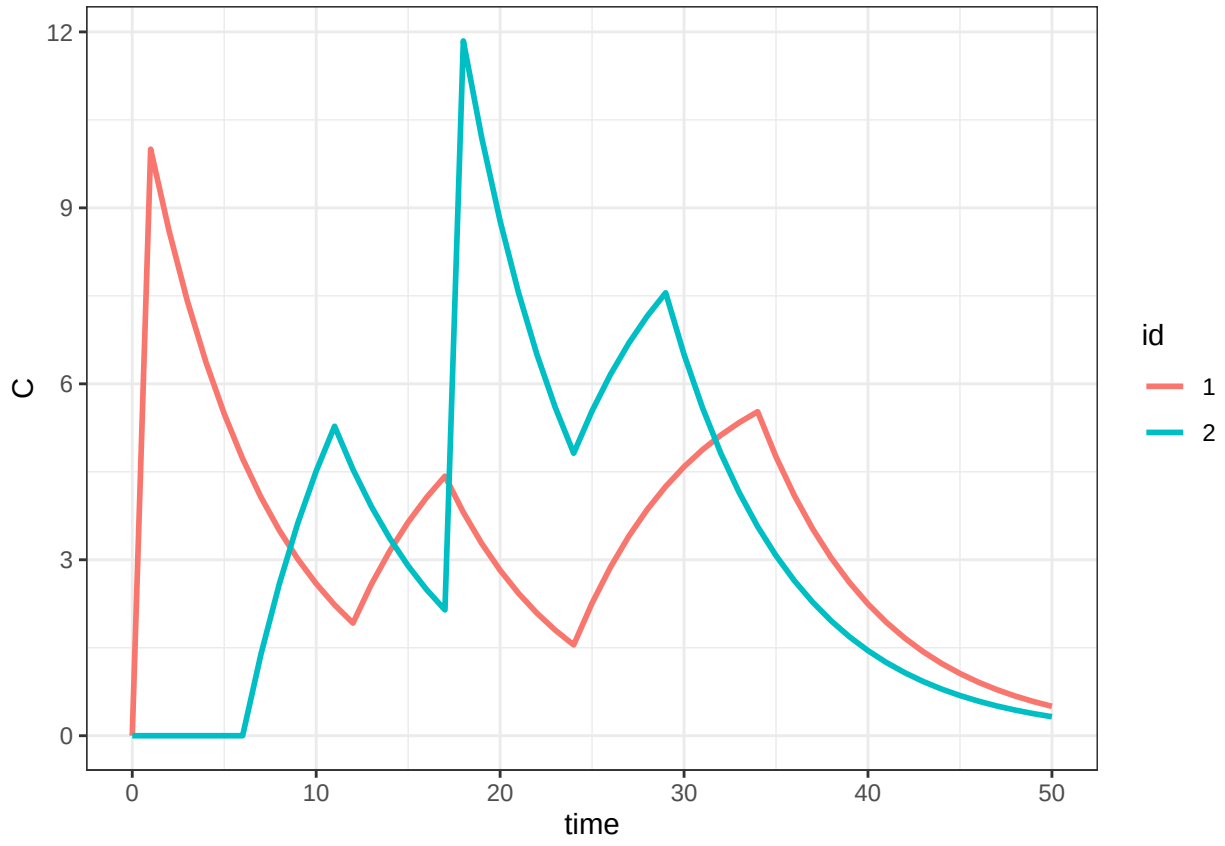


Treatment defined in a data frame

Individual dosage regimens can also be defined in an external data file, or in an "inline'' data frame:

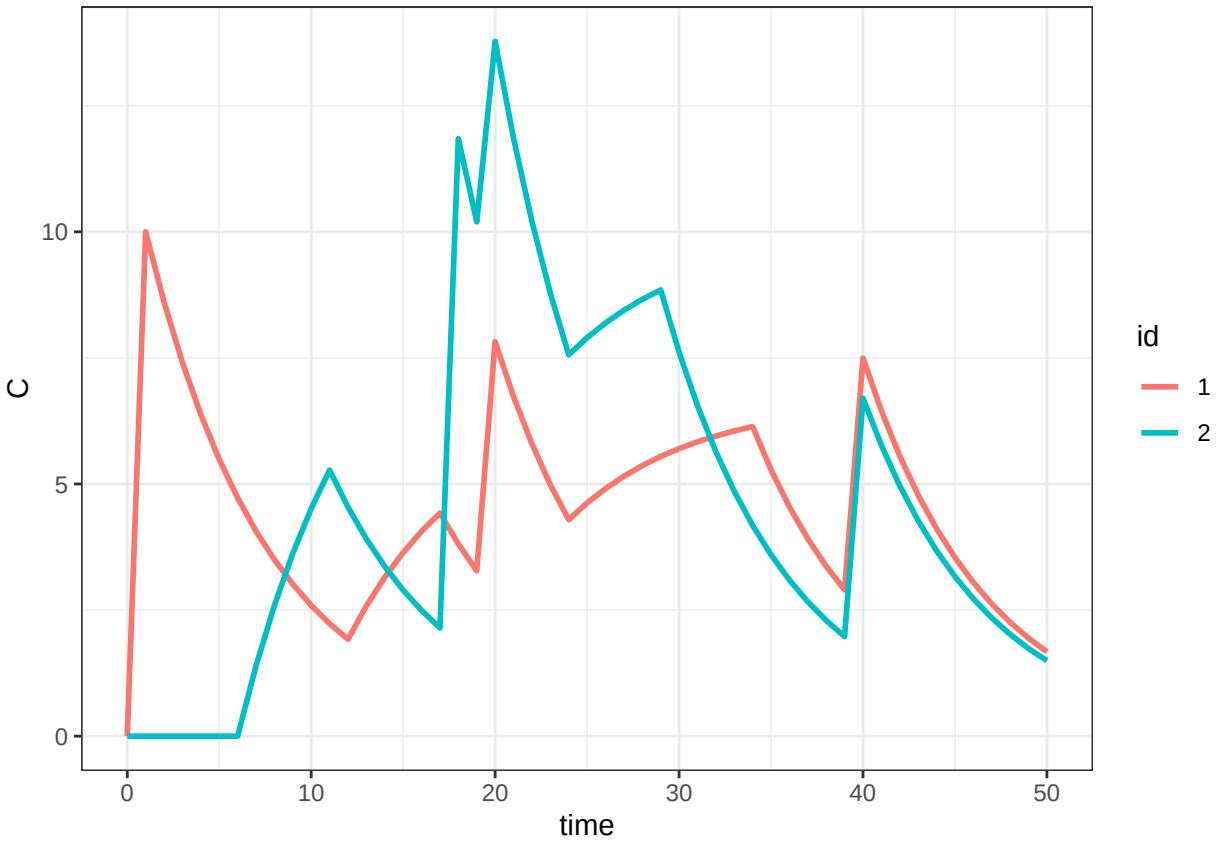
```
adm <- inlineDataFrame("
id  time  amount  rate
 1     1     100   0
 1    12      50   10
 1    24     100   10
 2     6      75   15
 2    18     100   0
 2    24      75   15
")
```

```
p   <- c(V=10, k=0.15)
C  <- list(name="C", time=seq(0, 50, by=1))

res2a <- simulx(model     = "model/groupI.txt",
                parameter = p,
                output    = C,
                treatment = adm)

print(ggplot(data=res2a$C, aes(x=time, y=C, colour=id)) + geom_line(size=1))
```



It is possible to combine these individual dose regimens with doses defined for all the patients:. in this example, 2 additional doses of 50mg are given to all the patients:

```
res2b <- simulx(model     = "model/groupI.txt",
                parameter = p,
                output    = C,
                treatment = list(adm, list(time=c(20,40),amount=50)),
                settings = list(load.design=TRUE))

print(ggplot(data=res2b$C, aes(x=time, y=C, colour=id)) + geom_line(size=1))
```



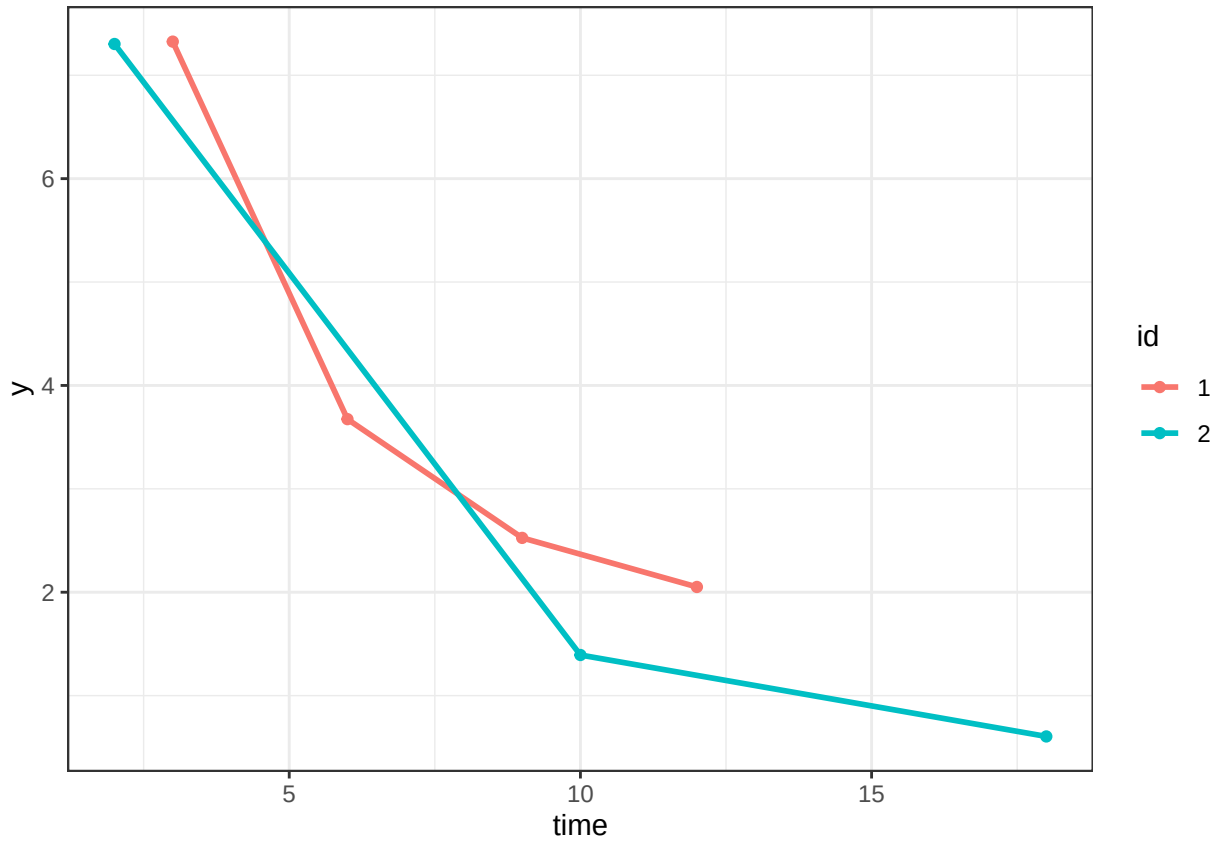
Observation times defined in a data frame

We will use in this example the model implemented in `groupII1.txt` where observed concentrations are defined.

Different individual can have different observation designs, i.e. different observation times (t_{ij}). Individual observation designs can be defined in a data file, or in an inline data frame:

```
p   <- c(V=10, k=0.15, a=0.2)
adm <- list(time=0, amount=100)

design.y  <- inlineDataFrame("
  id  time
   1     3
   1     6
   1     9
   1    12
   2     2
   2    10
   2    18
")
y  <- list(name="y",  time=design.y)


res3 <- simulx(model     = "model/groupII1.txt",
               parameter = p,
               output    = y,
```

```
                treatment = adm)

print(ggplot(data=res3$y, aes(x=time,y=y,color=id))+geom_line(size=1) + geom_point())
```



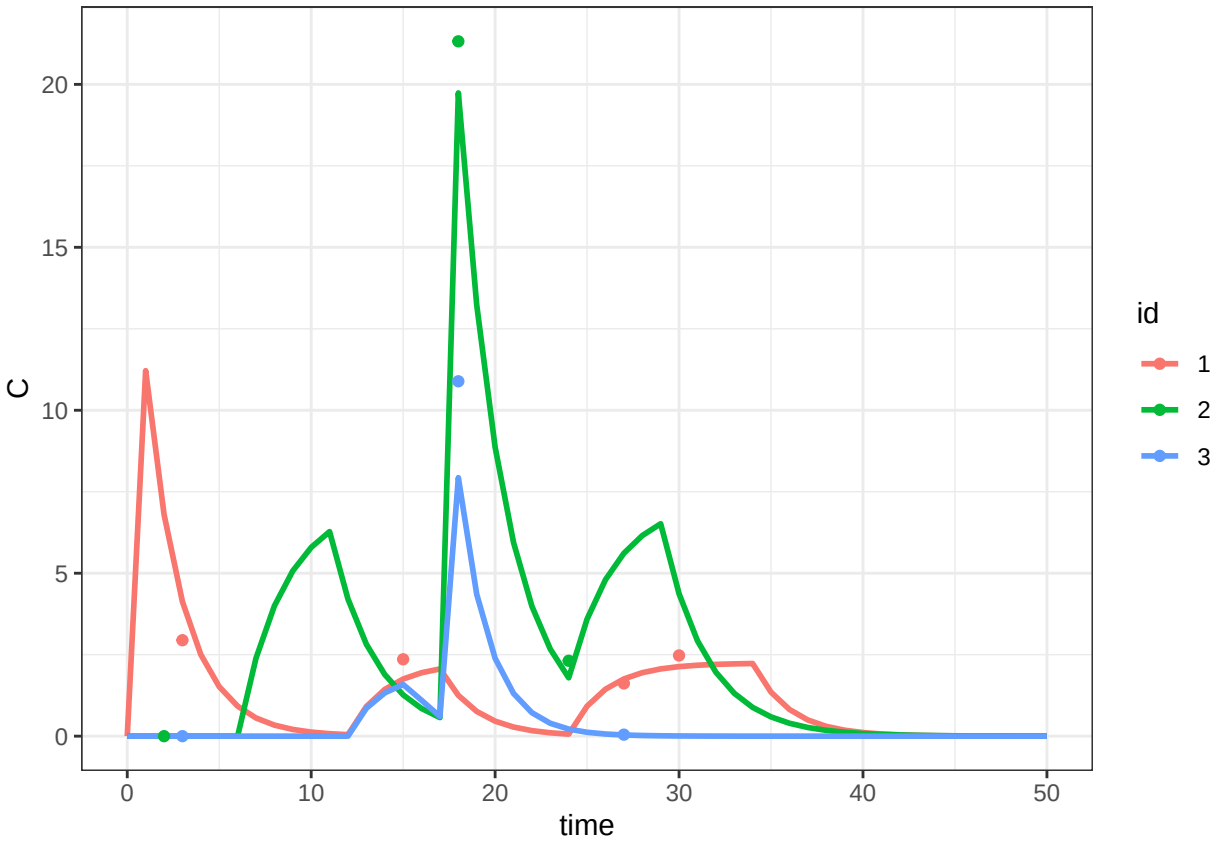
Individual parameters, dose regimens and observation times defined in data frames

Model `groupII2.txt` now includes a section `[INDIVIDUAL]`

We can use this model with individual designs (dosage regimens and observation times) and individual parameters w_i and k_i defined in several data frames. The 3 individuals defined in this example share the same population parameters V_{pop}, ω_V and a.

```
adm <- inlineDataFrame("
id  time amount rate
1     1    100    0
1    12     50   10
1    24    100   10
2     6     75   15
2    18    100    0
2    24     75   15
3    12     50   15
3    18    100    0
")

p.pop <- c(V_pop=10, omega_V=0.3, a=0.2)
```

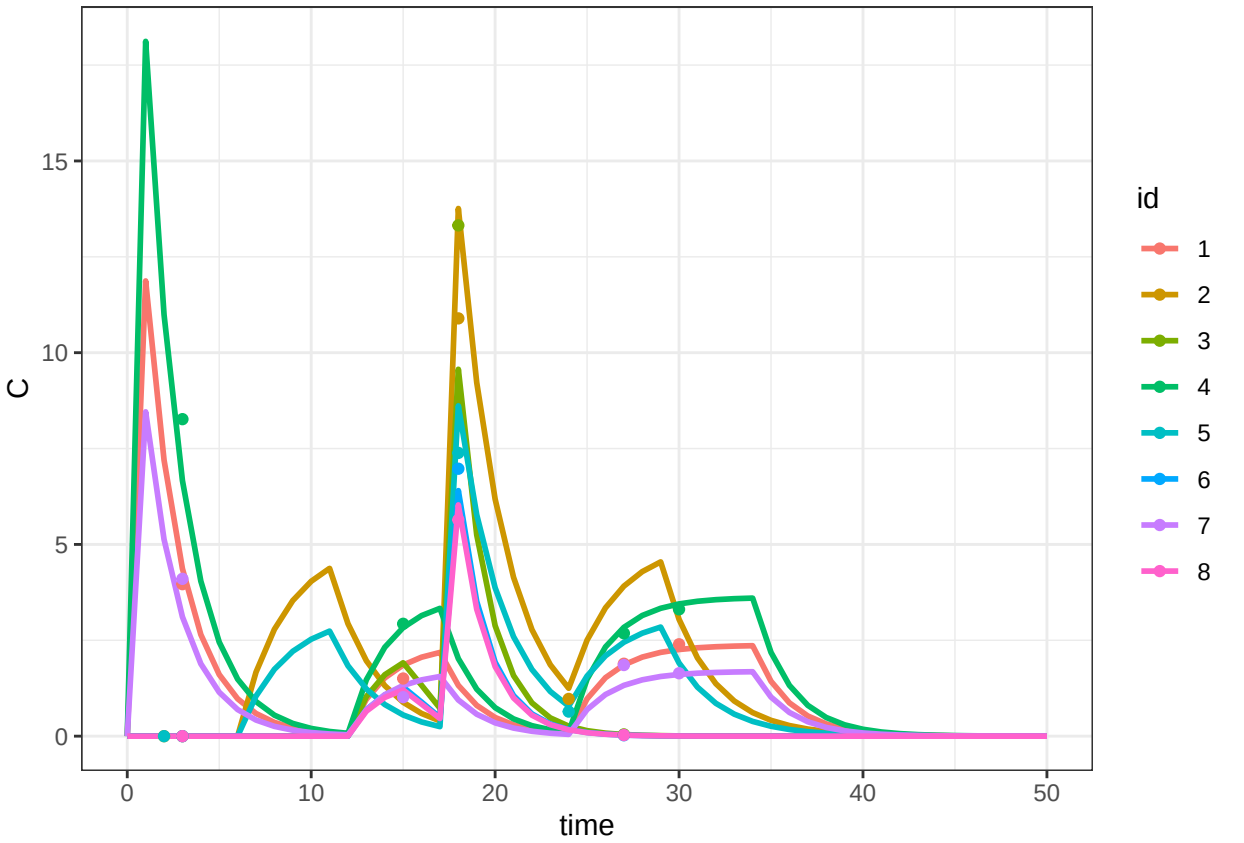
```
p.indiv <- inlineDataFrame("
id   w     k
 1  75   0.5
 2  60   0.4
 3  80   0.6
")


design.y  <- inlineDataFrame("
id  time
1     3
1    15
1    27
1    30
2     2
2    18
2    24
3     3
3    18
3    27
")
out.y <- list(name="y",  time=design.y)
out.C <- list(name="C", time=seq(0, 50, by=1))

res4 <- simulx(model     = "model/groupII2.txt",
               parameter = list(p.pop,p.indiv),
               output    = list(out.C, out.y),
               treatment = adm)

print(ggplot(data=res4$C, aes(x=time, y=C, colour=id)) + geom_line(size=1) +
         geom_point(data=res4$y, aes(x=time, y=y, colour=id)))
```



Sampling/resampling individuals from a database

We may want to use the same individual information provided by these 3 individuals, but for a different number of subjects. in this example, we create 8 individuals by resampling the 3 original individuals:

```
res5a <- simulx(model     = "model/groupII2.txt",
                parameter = list(p.pop,p.indiv),
                output    = list(out.C, out.y),
                treatment = adm,
                group     = list(size=8))

print(ggplot() + geom_line(data=res5a$C, aes(x=time,y=C,colour=id),size=1) +
        geom_point(data=res5a$y, aes(x=time, y=y, colour=id)))
```



You should note that the level of randomization (`level`) is not defined when subjects are resampled.

By default, `settings$replacement=F`, which means that the new id's are sampled within the original ones without replacement

```
print(res5a$originalId)
```

```
##   newId oriId
## 1     1     1
## 2     2     2
## 3     3     3
## 4     4     1
## 5     5     2
## 6     6     3
## 7     7     1
## 8     8     3
```

New id's are sampled with replacement by setting `settings$replacement=T`

```
res5b <- simulx(model      = "model/groupII2.txt",
                parameter = list(p.pop,p.indiv),
                output     = list(out.C, out.y),
                treatment = adm,
                group      = list(size=8),
                settings   = list(replacement=T))

print(res5b$originalId)
```

```
##   newId oriId
```

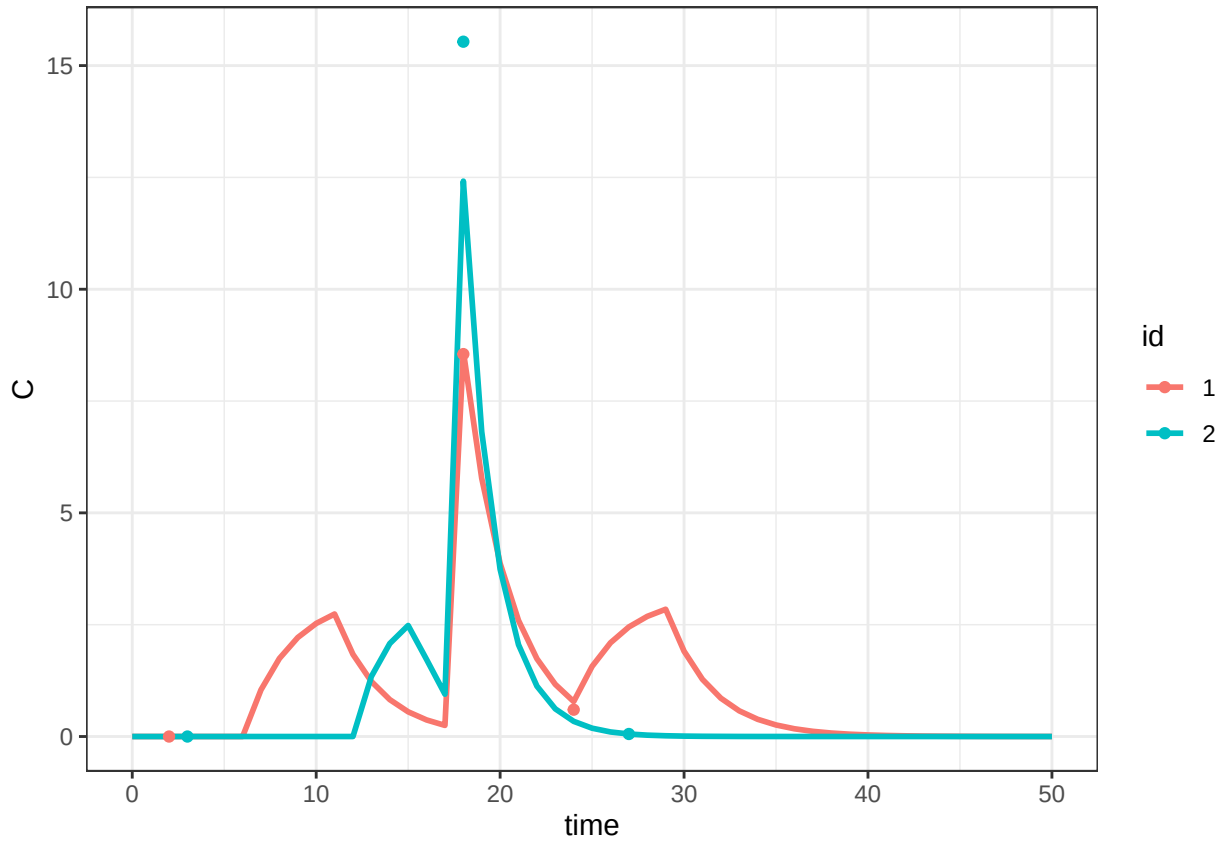
```
## 1     1     1
## 2     2     2
## 3     3     2
## 4     4     1
## 5     5     1
## 6     6     2
## 7     7     1
## 8     8     2
```

The same method can be used for generating less individuals than in the original data base:

```
res5c <- simulx(model     = "model/groupII2.txt",
                parameter = list(p.pop,p.indiv),
                output    = list(out.C, out.y),
                treatment = adm,
                group = list(size=2))

print(res5c$originalId)
```

```
##   newId oriId
## 1     1     2
## 2     2     3
```

```
print(ggplot() + geom_line(data=res5c$C, aes(x=time,y=C,colour=id),size=1) +
        geom_point(data=res5c$y, aes(x=time, y=y, colour=id)))
```



Using and modifying the original design

We show how to simulate PK data using the theophylline project, where the parameters have been estimated with `Monolix`.

Remark: in this project, the amount is given in mg per kilo.

```
project.file <- 'monolixRuns/theophylline_project.mlxtran'
```

Without any other inputs `simulx` will simulate the exact study design underlying the input data for parameter estimation

- Individual parameters are sampled from the estimated population distribution
- Covariate WEIGHT is taken from data set
- Dosage regimen and observation times are taken from data set
- Population parameters are set to the estimated values

```
res1  <- simulx(project = project.file)
names(res1)
```

```
## [1] "CONC"       "treatment"  "originalId" "population"
```

```
print(res1$treatment)
```

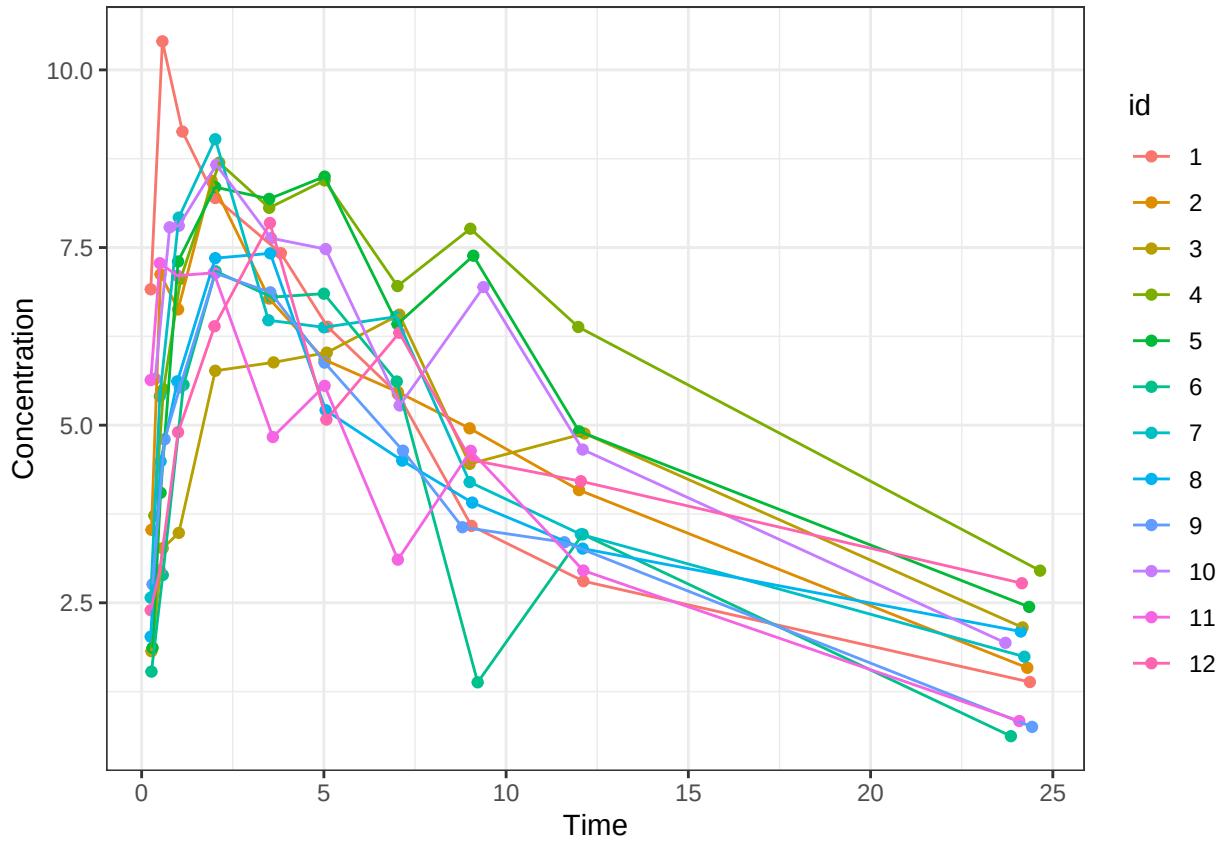
```
##    id time amount
## 1   1    0    320
## 2   2    0    320
## 3   3    0    320
## 4   4    0    320
## 5   5    0    320
## 6   6    0    320
## 7   7    0    320
## 8   8    0    320
## 9   9    0    320
## 10 10    0    320
## 11 11    0    320
## 12 12    0    320
```

```
print(res1$population)
```

```
##       ka_pop        V_pop       Cl_pop beta_Cl_lw70     omega_ka      omega_V
##   1.54903647  32.24013330   2.82665820   0.73866215   0.60809361   0.13839157
##     omega_Cl            a            b
##   0.25173786   0.59472481   0.07645794
```

plot the results ‘’as usual”

```
print(ggplot(data=res1$CONC) +
        geom_point(aes(x=time, y=CONC, colour=id)) +
        geom_line(aes(x=time, y=CONC, colour=id)) +
        scale_x_continuous("Time") + scale_y_continuous("Concentration"))
```



Some parameters values can be modified, e.g. switch off the residual error by setting a = b = 0, and some additional outputs can be defined, e.g. the predicted concentration Cc, the individual PK parameters and the covariates, .

```
sim.param <- c(a=0, b=0)
out1  <- list(name = 'Cc', time = seq(0, 25, by=0.1))
outp <- c("ka","V", "Cl","WEIGHT")
res2  <- simulx(project   = project.file,
                output    = list(out1,outp),
                parameter = sim.param)
names(res2)
```

```
## [1] "CONC"       "Cc"         "parameter"  "treatment"  "originalId"
## [6] "population"
```

```
print(res2$population)
```

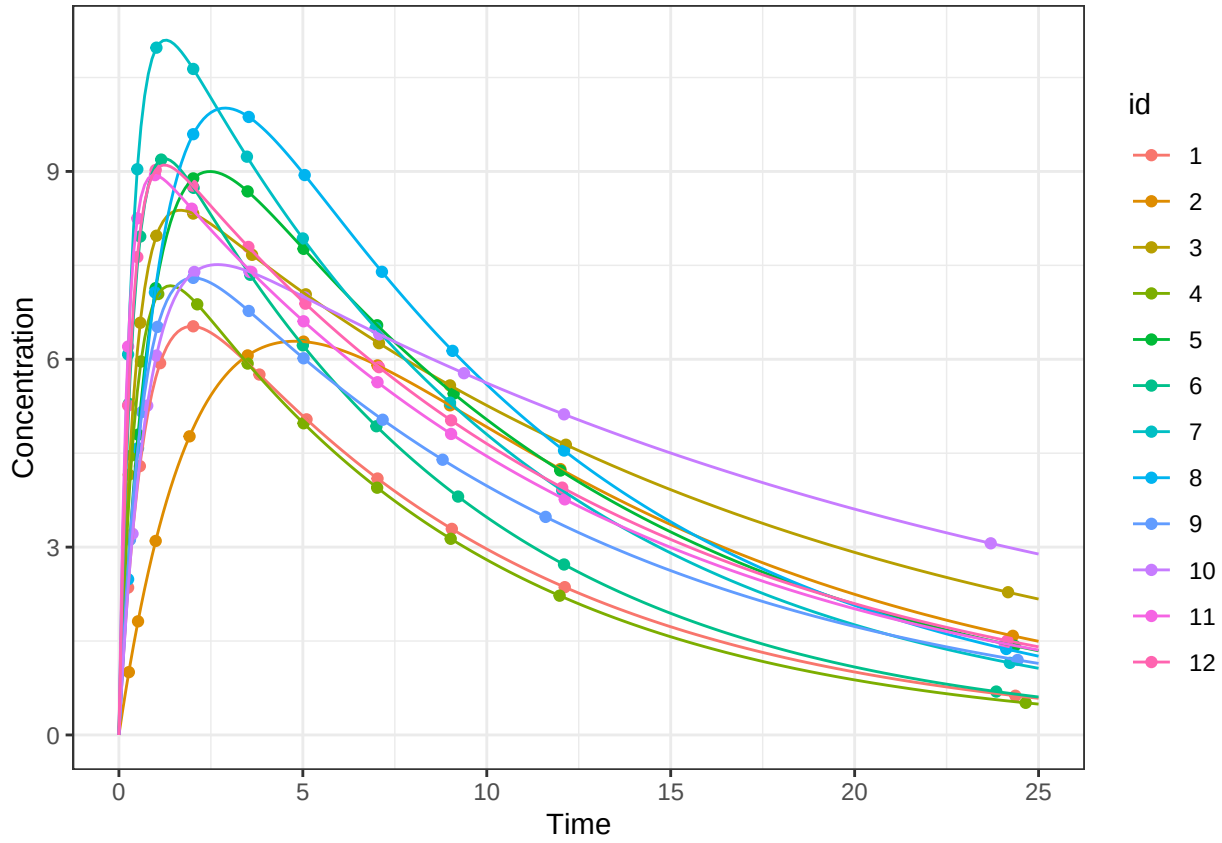
```
##       ka_pop        V_pop       Cl_pop beta_Cl_lw70     omega_ka      omega_V
##    1.5490365   32.2401333    2.8266582    0.7386622    0.6080936    0.1383916
##     omega_Cl            a            b
##    0.2517379    0.0000000    0.0000000
```

```
head(res2$parameter)
```

```
##   id        ka        V       Cl WEIGHT
## 1  1 1.3992381 39.55609 4.283014   79.6
## 2  2 0.4286136 34.47007 2.807393   72.4
## 3  3 2.2364053 34.60116 2.043219   70.5
```

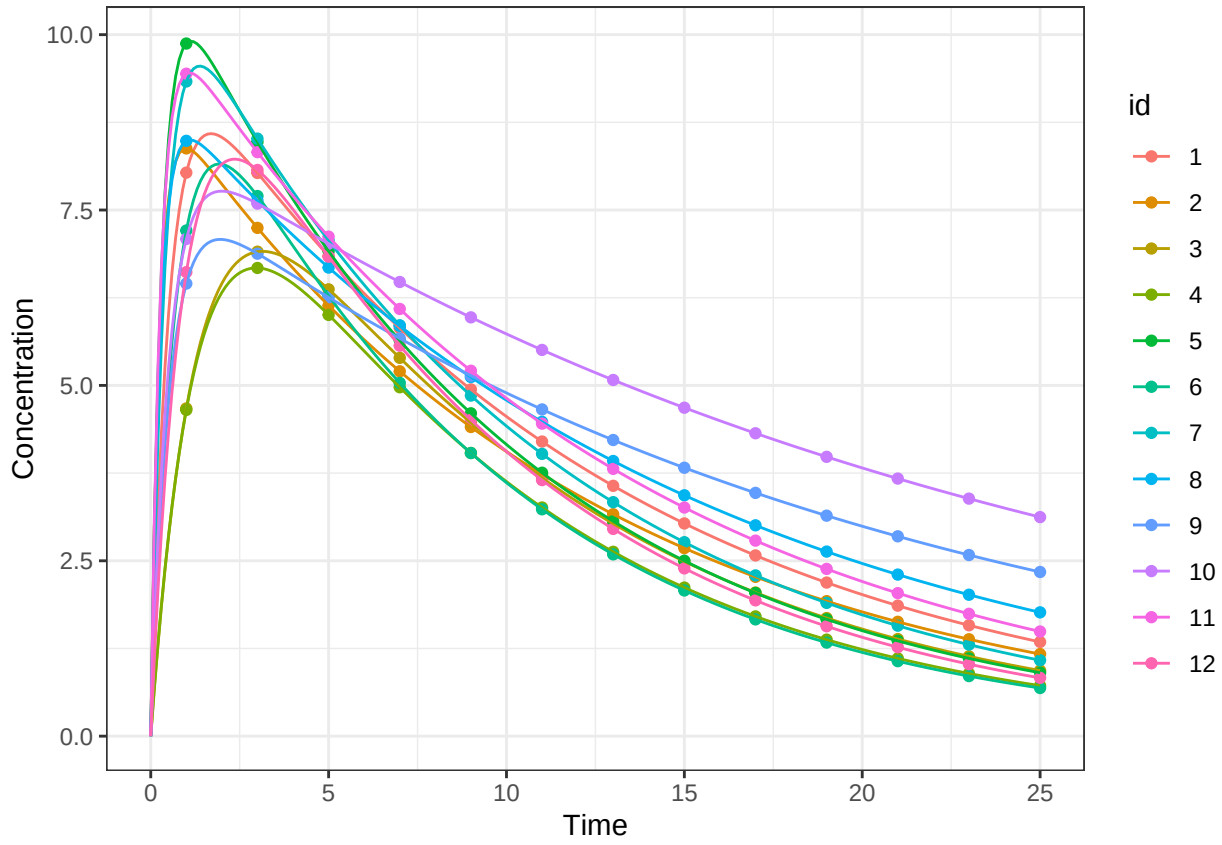
```
## 4  4 2.2280748 37.92824 4.389410   72.7
## 5  5 1.1132636 28.59630 2.518528   54.6
## 6  6 2.6105102 30.06991 3.500995   80.0
```

```
print(ggplot() +
        geom_point(data=res2$CONC, aes(x=time, y=CONC, colour=id)) +
        geom_line(data=res2$Cc, aes(x=time, y=Cc, colour=id)) +
        scale_x_continuous("Time") + scale_y_continuous("Concentration"))
```



Observation times can also be redefined:

```
out2  <- list(name = 'CONC', time = seq(1, 25, by=2))
res3  <- simulx(project   = project.file,
                output    = list(out1, out2),
                parameter = sim.param)

print(ggplot() +
        geom_point(data=res3$CONC, aes(x=time, y=CONC, colour=id)) +
        geom_line(data=res3$Cc, aes(x=time, y=Cc, colour=id)) +
        scale_x_continuous("Time") + scale_y_continuous("Concentration"))
```



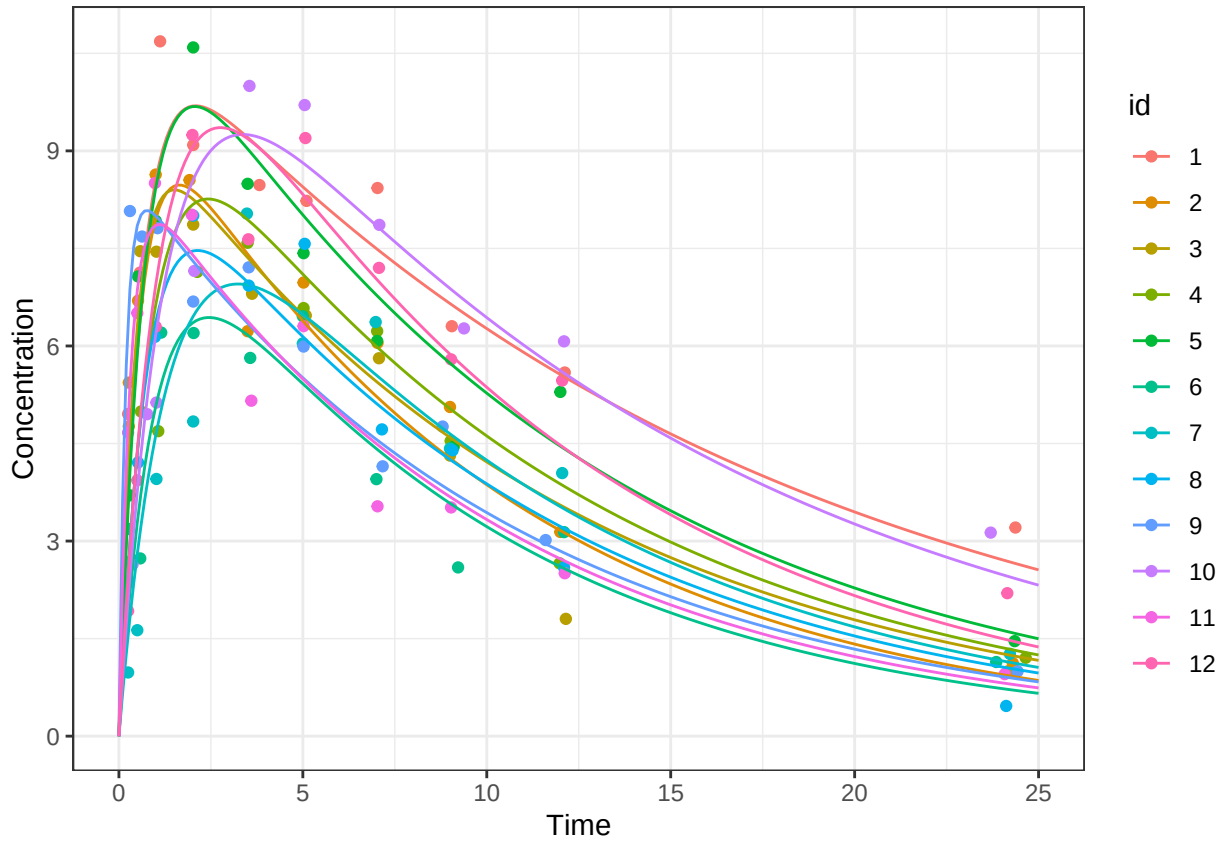
Remove the simulated concentrations from the outputs,

```
out0 <- list(name='CONC', time='none')
res4  <- simulx(project    = project.file,
                output     = list(out1,out0))
names(res4)
```

```
## [1] "Cc"         "treatment"  "originalId" "population"
```

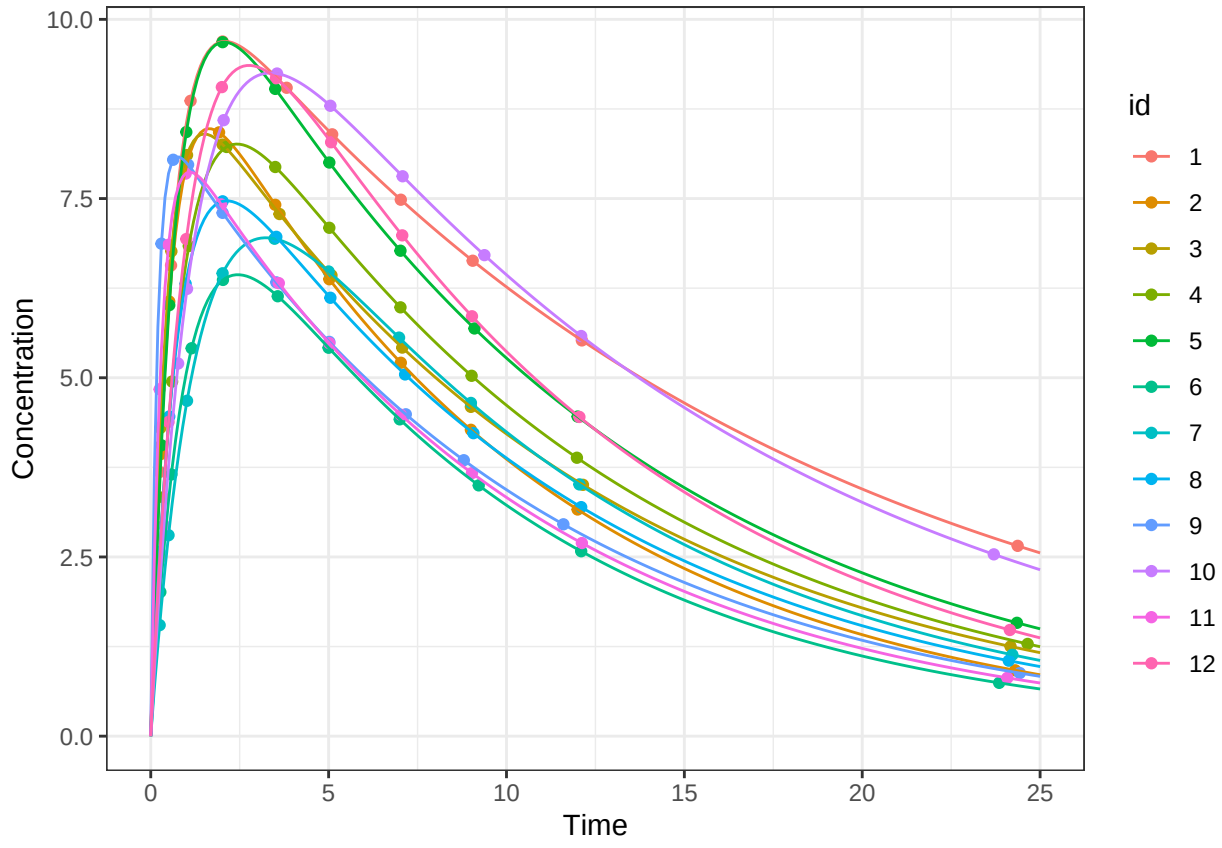
Use the estimated individual parameters (mode = EBEs),

```
sim.param <- list("mode")
res5  <- simulx(project    = project.file,
                output     = out1,
                parameter = sim.param)

print(ggplot() +
        geom_point(data=res5$CONC, aes(x=time, y=CONC, colour=id)) +
        geom_line(data=res5$Cc, aes(x=time, y=Cc, colour=id)) +
        scale_x_continuous("Time") + scale_y_continuous("Concentration"))
```



Use the estimated individual parameters and set b=0

```
sim.param <- list("mode", c(a=0, b=0))
res6  <- simulx(project   = project.file,
                output    = out1,
                parameter = sim.param)

print(ggplot() +
        geom_point(data=res6$CONC, aes(x=time, y=CONC, colour=id)) +
        geom_line(data=res6$Cc, aes(x=time, y=Cc, colour=id)) +
        scale_x_continuous("Time") + scale_y_continuous("Concentration"))
```



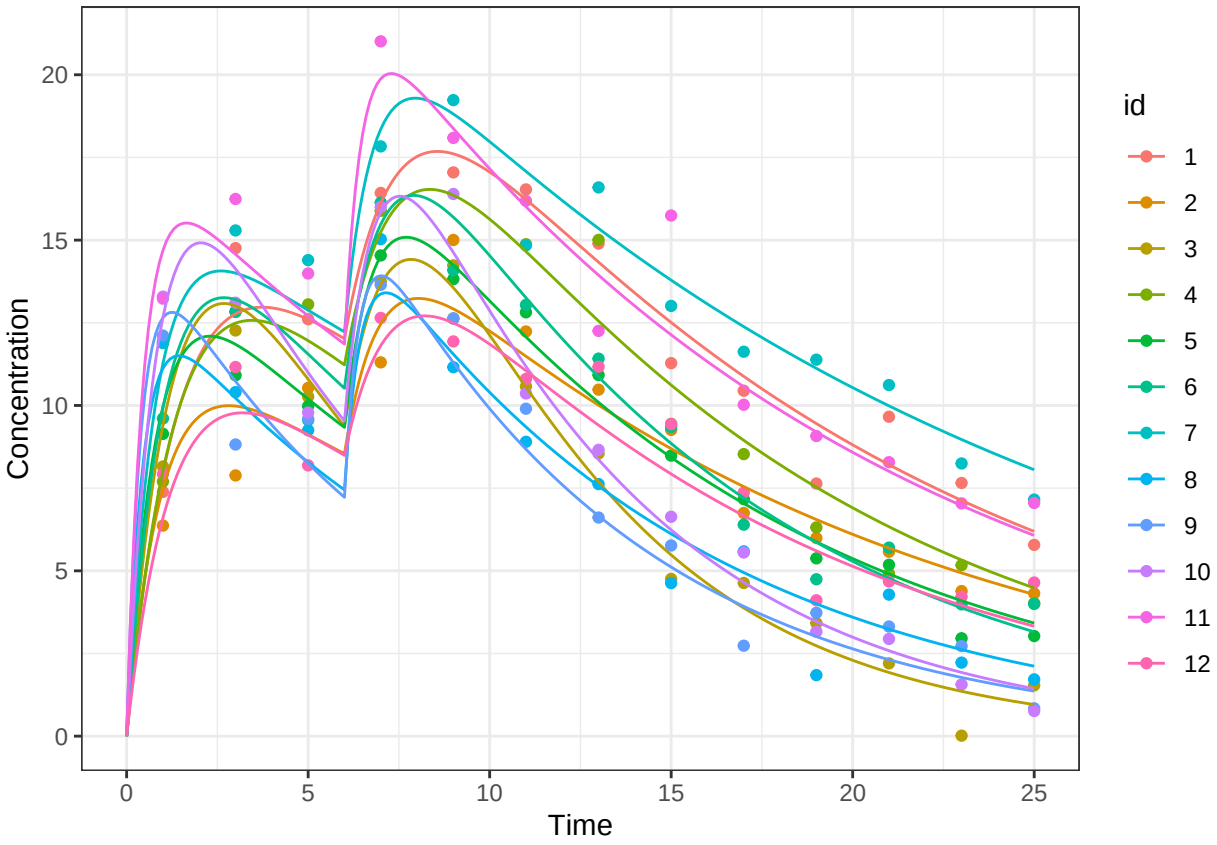
Remove treatement from the outputs,

```
res7  <- simulx(project    = project.file,
                settings   = list(out.trt=F))
names(res7)
```

```
## [1] "CONC"       "originalId" "population"
```

A new administration schedule can be specified

```
adm   <- list(time = c(0,6), amount = c(500, 300))

res8 <- simulx(project = project.file, treatment = adm, output = list(out1, out2))

print(ggplot() +
        geom_point(data=res8$CONC,aes(x=time, y=CONC, colour=id)) +
        geom_line(data=res8$Cc,aes(x=time, y=Cc, colour=id)) +
        scale_x_continuous("Time") + scale_y_continuous("Concentration"))
```


Concentration
Time
0
5
10
15
20
25
id
1
2
3
4
5
6
7
8
9
10
11
12

Resampling the patients of the original study

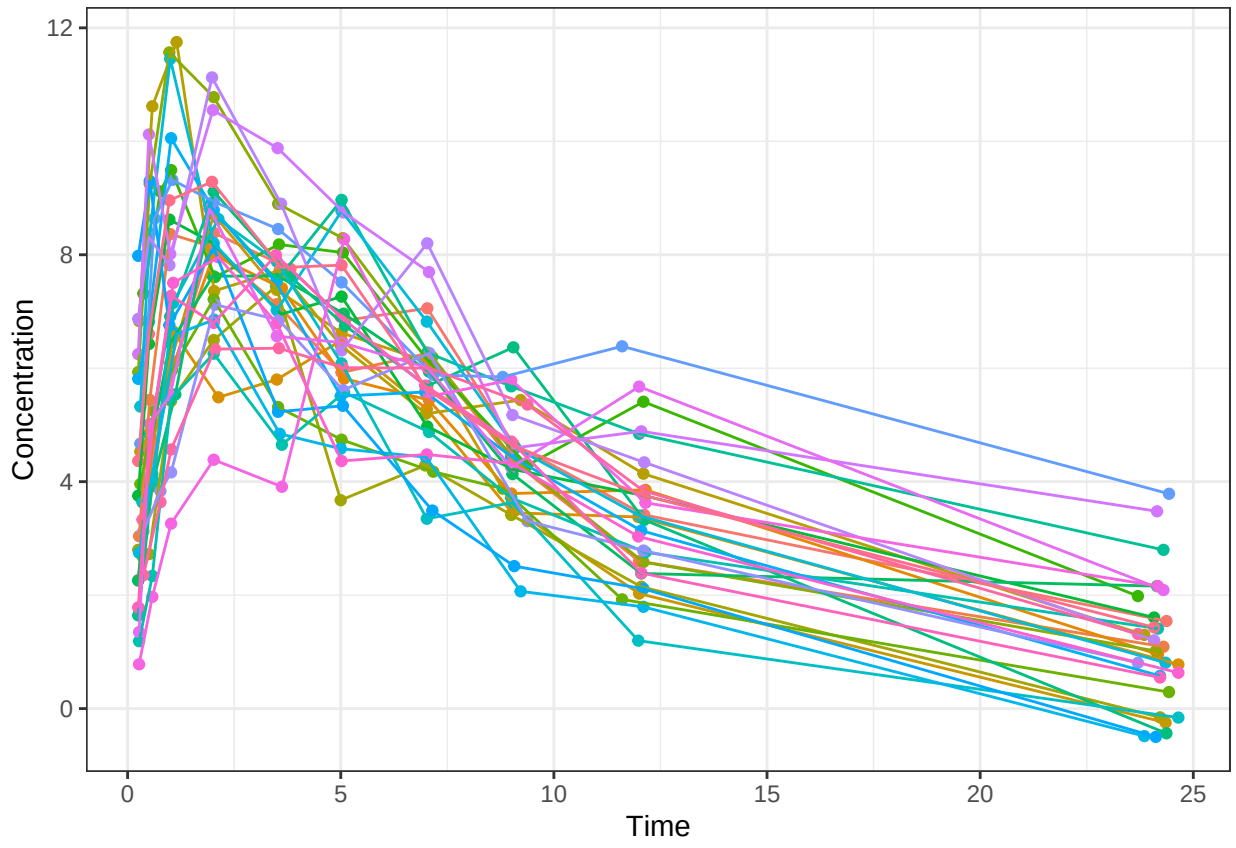
We will use again the theophylline project, where the parameters have been estimated with `Monolix`, but with different numbers of patient.

Remark: in this project, the amount is given in mg per kilo.

```
project.file <- 'monolixRuns/theophylline_project.mlxtran'
```

Simulate a trial with N individuals: the designs and the weights of these N patients are sampled from the original dataset

```
N <- 30
res1  <- simulx(project = project.file,
                group   = list(size = N))

print(ggplot(data=res1$CONC) +
        geom_point(aes(x=time, y=CONC, colour=id)) +
        geom_line(aes(x=time, y=CONC, colour=id)) +
        scale_x_continuous("Time") + scale_y_continuous("Concentration") +
        theme(legend.position="none"))
```



The designs of the $N = 30$ individuals are sampled from the 12 original designs

```
print(res1$treatment)
```

```
##    id time amount
## 1   1    0    320
## 2   2    0    320
```

```
## 3   3    0    320
## 4   4    0    320
## 5   5    0    320
## 6   6    0    320
## 7   7    0    320
## 8   8    0    320
## 9   9    0    320
## 10 10    0    320
## 11 11    0    320
## 12 12    0    320
## 13 13    0    320
## 14 14    0    320
## 15 15    0    320
## 16 16    0    320
## 17 17    0    320
## 18 18    0    320
## 19 19    0    320
## 20 20    0    320
## 21 21    0    320
## 22 22    0    320
## 23 23    0    320
## 24 24    0    320
## 25 25    0    320
## 26 26    0    320
## 27 27    0    320
## 28 28    0    320
## 29 29    0    320
## 30 30    0    320
```

By default, `settings$replacement=F`, which means that the new id's are sampled within the original ones without replacement

```
print(res1$originalId)
```

```
##    newId oriId
## 1      1     1
## 2      2     2
## 3      3     3
## 4      4     4
## 5      5     5
## 6      6     6
## 7      7     7
## 8      8     8
## 9      9     9
## 10    10    10
## 11    11    11
## 12    12    12
## 13    13     1
## 14    14     2
## 15    15     3
## 16    16     4
## 17    17     5
## 18    18     6
## 19    19     7
## 20    20     8
```

```
## 21    21     9
## 22    22    10
## 23    23    11
## 24    24    12
## 25    25     2
## 26    26     3
## 27    27     4
## 28    28     7
## 29    29    10
## 30    30    11
```

New id's are sampled with replacement by setting `settings$replacement=T`

```
res2  <- simulx(project = project.file,
                group   = list(size = N),
                settings  = list(replacement=T))
print(res2$originalId)
```

```
##    newId oriId
## 1      1     6
## 2      2     7
## 3      3     3
## 4      4    12
## 5      5    12
## 6      6    12
## 7      7     9
## 8      8     6
## 9      9    10
## 10    10     1
## 11    11     6
## 12    12     9
## 13    13     5
## 14    14     8
## 15    15     1
## 16    16    10
## 17    17     2
## 18    18     9
## 19    19    12
## 20    20     5
## 21    21     4
## 22    22     7
## 23    23     7
## 24    24     8
## 25    25     6
## 26    26    11
## 27    27    11
## 28    28     1
## 29    29     6
## 30    30    10
```

When a new size group is defined, estimated individual parameters are resampled as well,

```
N <- 5
res3  <- simulx(project = project.file,
                parameter = "mode",
                group   = list(size = N))
```

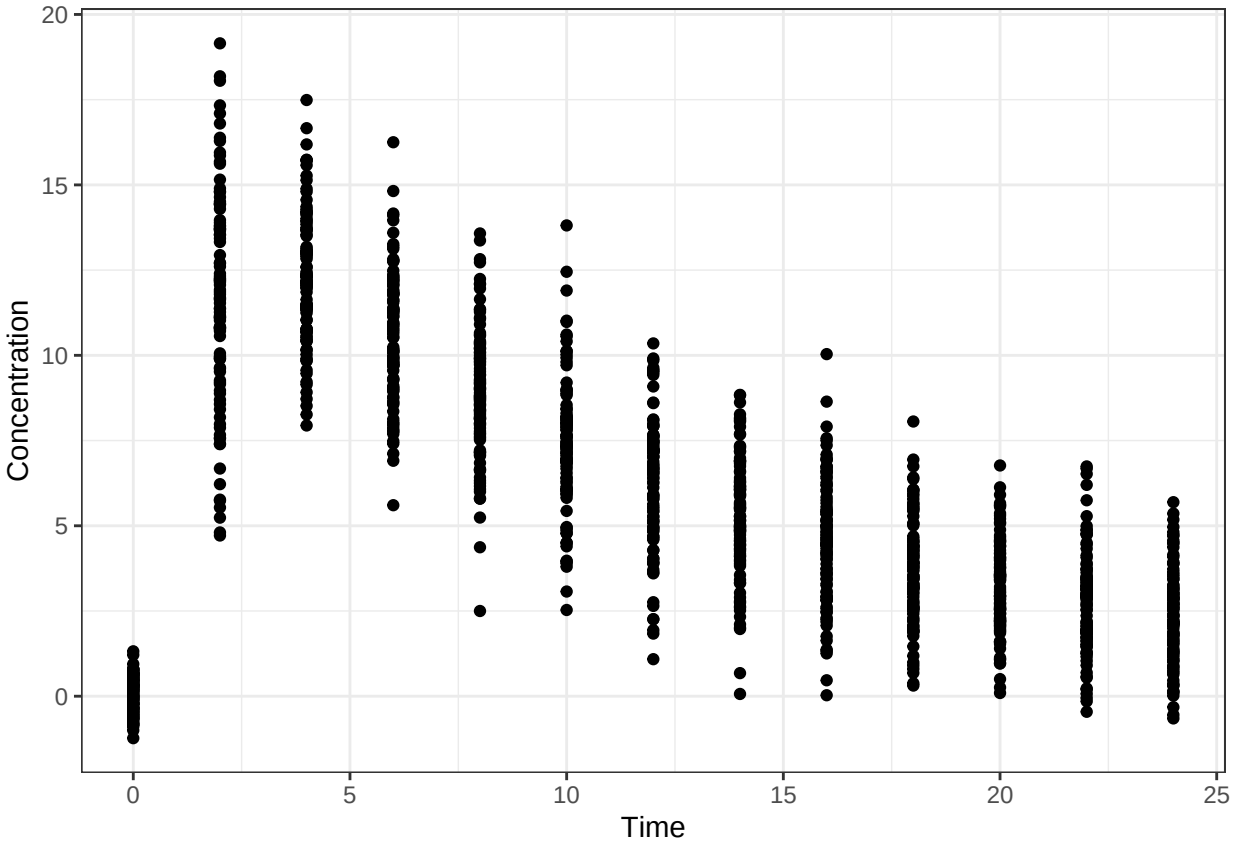
```
print(res3$originalId)
```

```
##   newId oriId
## 1     1     3
## 2     2     5
## 3     3     6
## 4     4     8
## 5     5    10
```

```
N       <- 100
weight <- data.frame(id = (1:N), WEIGHT = rnorm(n=N, mean=70, sd=10))
adm   <- list(time = 1, amount = 500)
out2  <- list(name = 'CONC', time = seq(0, 25, by=2))
outw  <- "WEIGHT"

res4 <- simulx(project = project.file,
               output = list(outw, out2),
               treatment = adm,
               parameter = weight)

print(head(res4$parameter))
```

```
##   id   WEIGHT
## 1  1 69.38058
## 2  2 74.14754
## 3  3 63.27058
## 4  4 93.78562
## 5  5 56.69552
## 6  6 90.94811
```

```
print(ggplot(data=res4$CONC,aes(x=time, y=CONC, by=id)) +  geom_point() +
        scale_x_continuous("Time") + scale_y_continuous("Concentration"))
```


Concentration
Time
0
5
10
15
20
25

Introducing uncertainty on the population parameters

We still use the same theophylline project, where the parameters and the Fisher information matrix have been estimated with `Monolix`.

We will see how to use the estimated Fisher information matrix for taking into account the **uncertainty** on the parameter estimates.

```
project.file <- 'monolixRuns/theophylline_project.mlxtran'
```

The estimated population parameters are used when `simulx` is used with a Monolix project

```
res1  <- simulx(project = project.file)

print(res1$population, digits=3)
```

```
##       ka_pop        V_pop       Cl_pop beta_Cl_lw70     omega_ka      omega_V
##       1.5490      32.2401       2.8267       0.7387       0.6081       0.1384
##     omega_Cl            a            b
##       0.2517       0.5947       0.0765
```

and also when several replicates are simulated

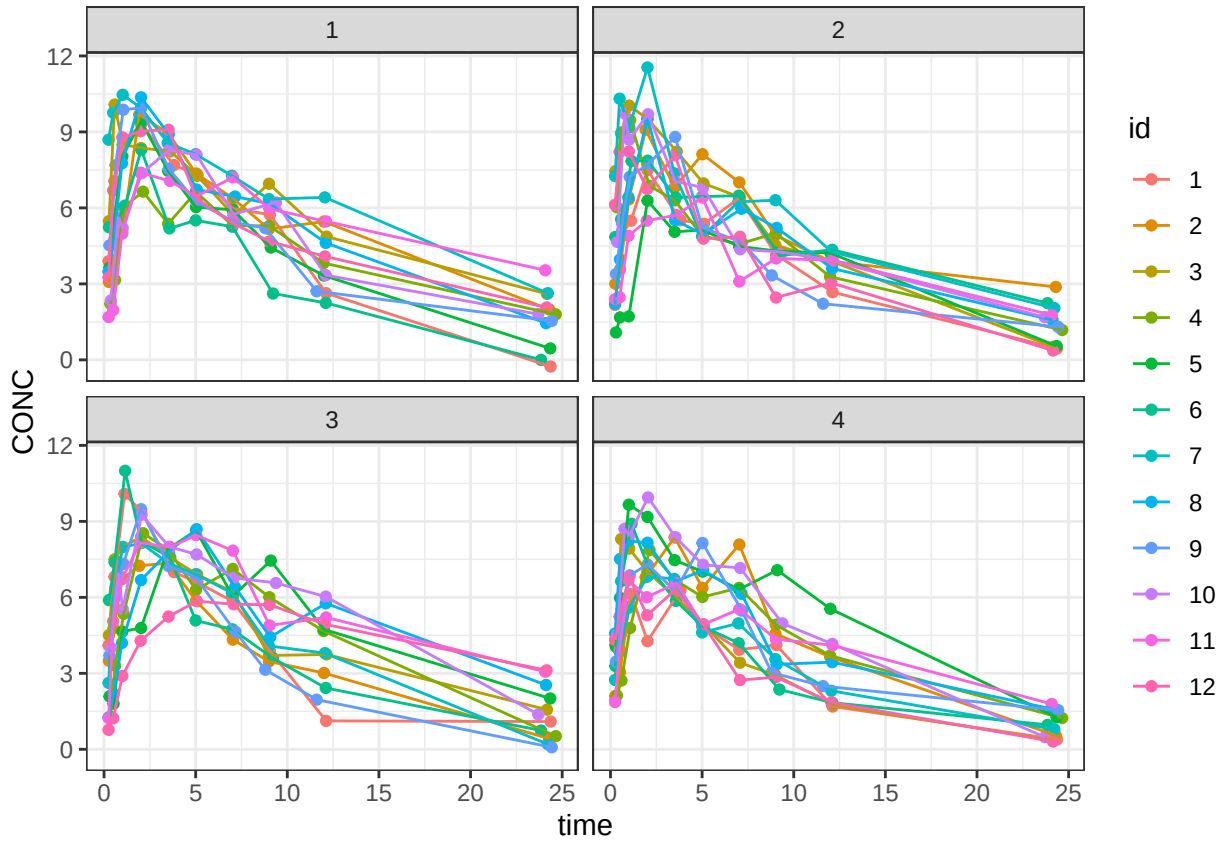
```
res2 <- simulx(project = project.file,
               nrep    = 4,
               settings=list(seed=123456))
print(res2$population, digits=3)
```

```
##       ka_pop        V_pop       Cl_pop beta_Cl_lw70     omega_ka      omega_V
##       1.5490      32.2401       2.8267       0.7387       0.6081       0.1384
##     omega_Cl            a            b
##       0.2517       0.5947       0.0765
```

```
head(res2$CONC)
```

```
##   rep id time     CONC
## 1   1  1 0.25 3.895295
## 2   1  1 0.57 7.057058
## 3   1  1 1.12 8.645716
## 4   1  1 2.02 9.295008
## 5   1  1 3.82 7.690024
## 6   1  1 5.10 7.316662
```

```
print(ggplot(data=res2$CONC,aes(x=time, y=CONC, colour=id)) +
        geom_point() +  geom_line() + facet_wrap(~ rep, ncol=2))
```



when npop is defined, then the population parameters are drawn from the distribution of the estimates (derived from the Fisher information matrix)

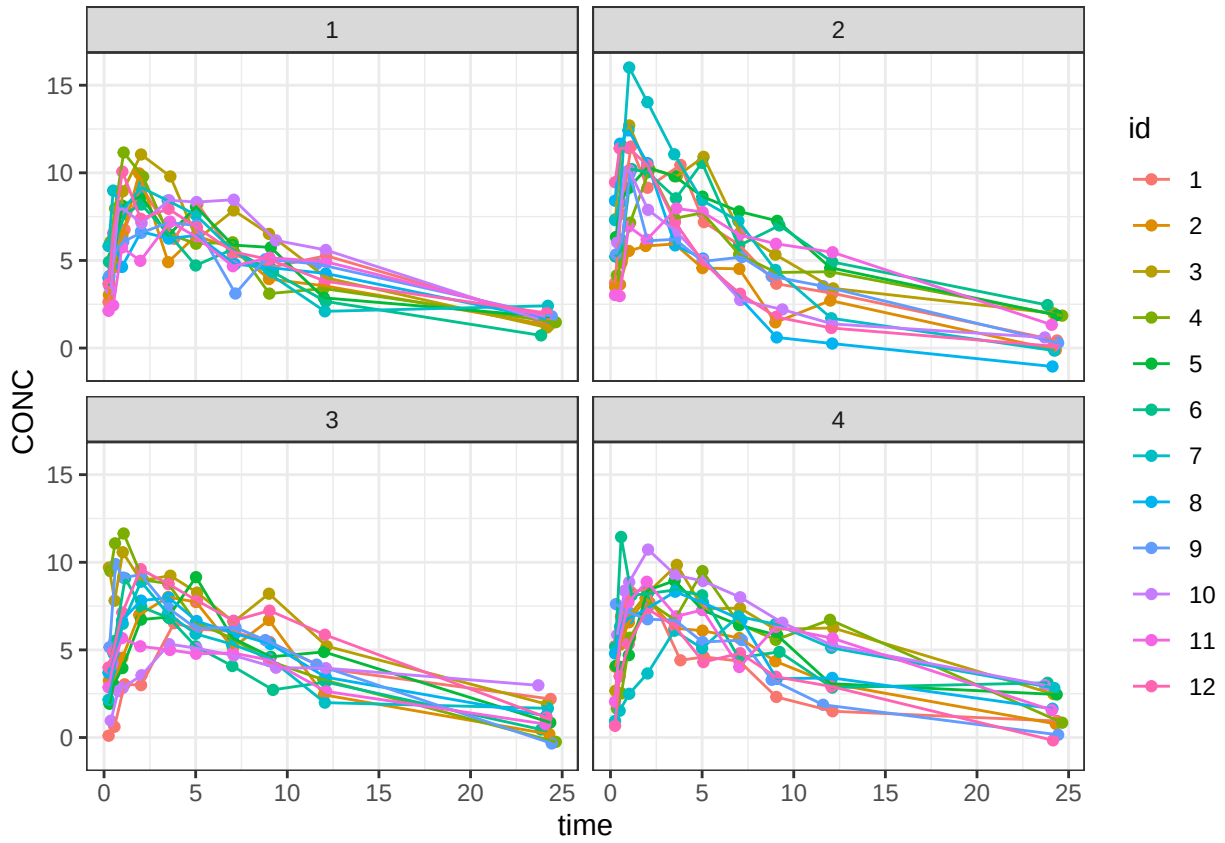
```
res3a <- simulx(project = project.file,
                npop    = 4,
                settings=list(seed=12345))
print(res3a$population, digits=3)
```

```
##   pop ka_pop V_pop Cl_pop beta_Cl_lw70 omega_ka omega_V omega_Cl     a      b
## 1   1   1.70  33.3   2.73        0.938    0.479  0.1611    0.161 0.684 0.0929
## 2   2   1.74  29.7   2.65        0.988    0.555  0.1959    0.361 0.601 0.0973
## 3   3   1.49  33.2   2.78        0.277    0.770  0.1178    0.220 0.679 0.0628
## 4   4   1.40  31.7   3.27        1.367    0.630  0.0829    0.305 0.854 0.0317
```

```
head(res3a$CONC)
```

```
##   pop id time     CONC
## 1   1  1 0.25 2.626375
## 2   1  1 0.57 5.046601
## 3   1  1 1.12 6.756273
## 4   1  1 2.02 9.241136
## 5   1  1 3.82 6.235755
## 6   1  1 5.10 8.240179
```

```
print(ggplot(data=res3a$CONC,aes(x=time, y=CONC, colour=id)) +
        geom_point() +  geom_line() + facet_wrap(~ pop, ncol=2))
```



If the FIM has been estimated by stochastic approximation, then this matrix is used by default (which is equivalent to set `fim="sa"`).

If the FIM has been estimated by linearization of the model, this matrix can be used by setting `fim="lin"`,

```
res3b <- simulx(project = project.file,
                npop    = 4,
                fim     = "lin",
                settings=list(seed=12345))
print(res3b$population, digits=3)
```

```
##   pop ka_pop V_pop Cl_pop beta_Cl_lw70 omega_ka omega_V omega_Cl     a      b
## 1   1   1.70  33.3   2.73        0.940    0.487  0.1662    0.167 0.670 0.0948
## 2   2   1.74  29.7   2.65        0.978    0.551  0.2005    0.375 0.564 0.1053
## 3   3   1.49  33.2   2.78        0.285    0.762  0.1105    0.218 0.681 0.0650
## 4   4   1.40  31.7   3.27        1.354    0.634  0.0859    0.285 0.819 0.0442
```

Drawing population parameters can also be done by first simulating the population parameters, and then using this data frame with `simulx` as parameter.

Remark: the column pop is mandatory.

```
pop4 <- simpopmlx(n=4, project=project.file)
print(pop4, digits=3)
```

```
##   pop ka_pop V_pop Cl_pop beta_Cl_lw70 omega_ka omega_V omega_Cl     a      b
## 1   1   1.43  33.9   2.99       1.1503    0.590  0.1580    0.247 0.526 0.1001
## 2   2   1.12  28.5   3.26       0.0689    0.651  0.1687    0.174 0.353 0.1035
## 3   3   2.11  30.9   2.52       0.3103    0.519  0.2475    0.256 0.733 0.0410
```

```
## 4   4   1.53  33.7   2.92       1.9501    0.618  0.0656    0.396 0.789 0.0265
```

```
res4 <- simulx(project   = project.file,
               parameter = pop4)
print(res4$population, digits=3)
```

```
##   pop ka_pop V_pop Cl_pop beta_Cl_lw70 omega_ka omega_V omega_Cl     a      b
## 1   1   1.43  33.9   2.99       1.1503    0.590  0.1580    0.247 0.526 0.1001
## 2   2   1.12  28.5   3.26       0.0689    0.651  0.1687    0.174 0.353 0.1035
## 3   3   2.11  30.9   2.52       0.3103    0.519  0.2475    0.256 0.733 0.0410
## 4   4   1.53  33.7   2.92       1.9501    0.618  0.0656    0.396 0.789 0.0265
```

it is possible to combine the simulation of several populations and several replicates with a given group size,

```
res5a <- simulx(project = project.file,
                npop    = 4,
                nrep    = 3,
                group   = list(size=20))
head(res5a$CONC)
```

```
##   pop rep id time     CONC
## 1   1   1  1 0.25 3.285568
## 2   1   1  1 0.57 6.272405
## 3   1   1  1 1.12 8.236781
## 4   1   1  1 2.02 8.104756
## 5   1   1  1 3.82 6.877909
## 6   1   1  1 5.10 6.469457
```

Set `settings$disp.iter=TRUE` to display the iteration numbers

```
res5b <- simulx(project = project.file,
                npop    = 4,
                nrep    = 3,
                group   = list(size=20),
                settings=list(disp.iter=TRUE))
```

```
##
## population:  1
## replicate:  1
## replicate:  2
## replicate:  3
##
## population:  2
## replicate:  1
## replicate:  2
## replicate:  3
##
## population:  3
## replicate:  1
## replicate:  2
## replicate:  3
##
## population:  4
## replicate:  1
## replicate:  2
## replicate:  3
```

If the results are stored in a datafile, then a column pop is automatically added,

```
g1 = list(size=100, treatment=list(amount=300, time=0))
g2 = list(size=100, treatment=list(amount=600, time=0))
res5c <- simulx(project = project.file,
                npop    = 4,
                group   = list(g1, g2),
                result.file = "theo5c.csv")
head(read.table("theo5c.csv", header=T, sep=","))
```

```
##   pop id time      y group amount
## 1   1  1 0.00      .     1    300
## 2   1  1 0.25 4.0036     1      .
## 3   1  1 0.57 5.4019     1      .
## 4   1  1 1.12 7.0702     1      .
## 5   1  1 2.02 9.0676     1      .
## 6   1  1 3.82 8.7735     1      .
```

Using a model defined inline

R script: inline.R

Introduction

Instead of using an external text file for the model, it is possible to define the model *inline*, i.e. in the R script with the command

```
myModel <- inlineModel("
[LONGITUDINAL]
...
")
```

This model can then be used as any model Mlxtran:

```
res <- simulx( model=myModel, ... )
```

The syntax for implementing the model is exactly the same: you can copy and paste any Mlxtran model in the R script, in the comment area defined by (" and "). The only difference is that all the information required for performing a given simulation is in one single R file.

Function inlineModel creates a temporary model file filename. Default name is "tempModel.txt". filename="random" generates a random name.

Example

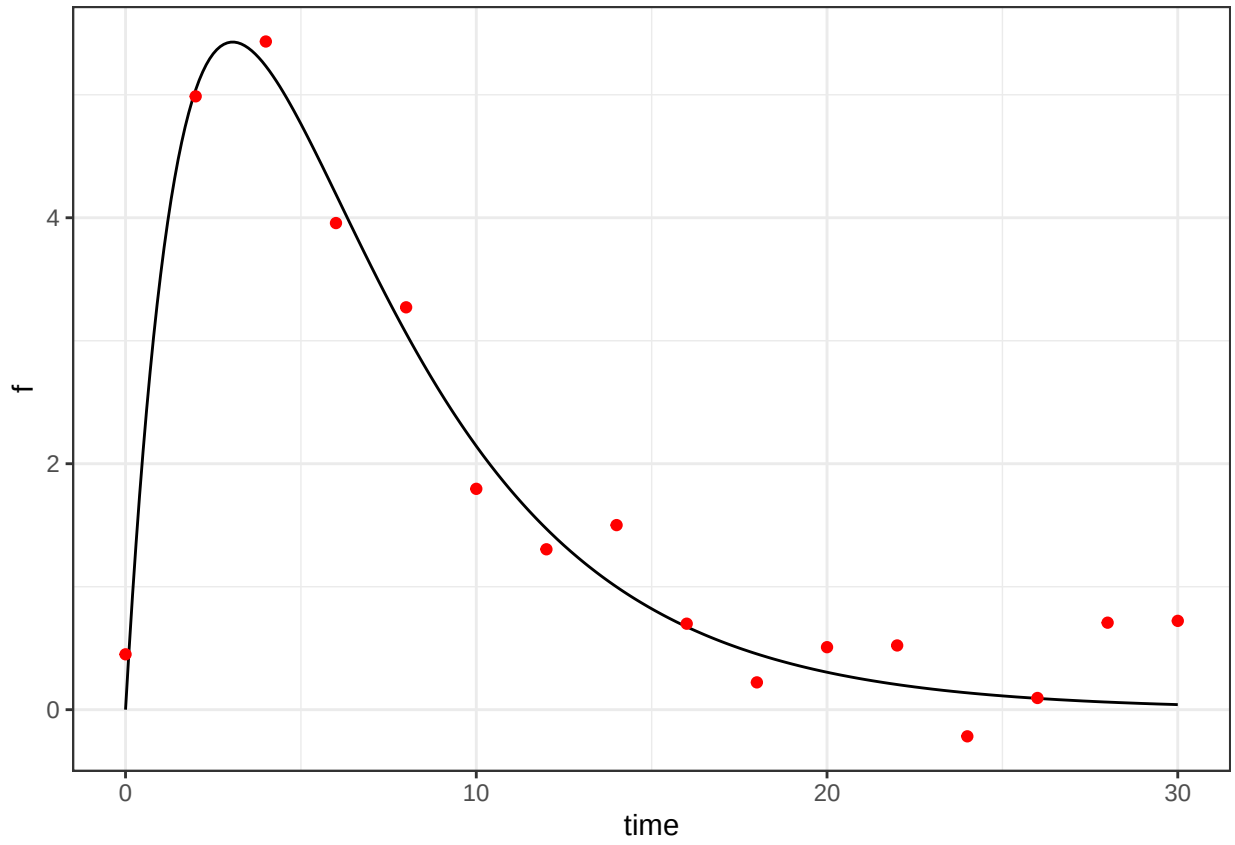
```
myModel1 <- inlineModel("
[LONGITUDINAL]
input = {ka, V, k, a}

EQUATION:
D = 100
f = D*ka/(V*(ka-k))*(exp(-k*t) - exp(-ka*t))

DEFINITION:
y = {distribution=normal, prediction=f, sd=a}
")

print(myModel1)
```

```
## $filename
## [1] "tempModel.txt"
##
## $str
## [1] "\n[LONGITUDINAL]\ninput = {ka, V, k, a}\n\nEQUATION:\nD = 100\nf = D*ka/(V*(ka-k))*(exp(-k*t) -
```

We can now use myModel1.txt with simulx:

```
f <- list(name='f', time=seq(0, 30, by=0.1))
y <- list(name='y', time=seq(0, 30, by=2))

res <- simulx(model     = myModel1,
              parameter = c(ka=0.5, V=10, k=0.2, a=0.3),
```

```
              output    = list(f, y))

plot(ggplot(data=res$f, aes(x=time, y=f)) + geom_line(size=0.5) +
  geom_point(data=res$y, aes(x=time, y=y), colour="red"))
```



filename="random" creates a tempory model file with a random name. This may be useful when several people use siultaneously the same Shiny app for instance.

```
myModel2 <- inlineModel("
[LONGITUDINAL]
input = {ka, V, k, a}

EQUATION:
f = 100*ka/(V*(ka-k))*(exp(-k*t) - exp(-ka*t))

DEFINITION:
y = {distribution=normal, prediction=f, sd=a}
", filename="random")

print(myModel2)
```

```
## $filename
## [1] "tempModel_54d20d11-e8c3-4af7-bccc-34effb5c6e8b.txt"
##
## $str
## [1] "\n[LONGITUDINAL]\ninput = {ka, V, k, a}\n\nEQUATION:\nf = 100*ka/(V*(ka-k))*(exp(-k*t) - exp(-ka
```

```
res <- simulx(model     = myModel2,
              parameter = c(ka=0.5, V=10, k=0.2, a=0.3),
              output    = list(f, y))
```

Obviously, the model can be written directly in any text file using basic R command:

```
model.str <- "
[LONGITUDINAL]
input = {ka, V, k, a}

EQUATION:
f = 100*ka/(V*(ka-k))*(exp(-k*t) - exp(-ka*t))

DEFINITION:
y = {distribution=normal, prediction=f, sd=a}
"

write(model.str,"pk_model.txt")


res <- simulx(model     = "pk_model.txt",
              parameter = c(ka=0.5, V=10, k=0.2, a=0.3),
              output    = list(f, y))
```

Creation of new outputs

R scripts: newoutput.R

Mlxtran codes: model/multiblock.txt ; model/newoutput.txt

Monolix project: theophylline_project.mlxtran

Imagine that we want to define as output of `simulx` some variables which are not defined in the Mlxtran model code. It is then possible to create a new model file with additional lines defined in the `R` script.

The new input argument `addlines` of `simulx` is a list (or a list of lists) with fields * section: "[LONGITUDINAL]" (default), "[INDIVIDUAL]", "[COVARIATE]" or "[POPULATION]" * block: "EQUATION:" or "DEFINITION:" * formula: a single string (e.g. `formula = "ddt_auc=Cc"`), or a vector of strings e.g. `formula = c("lc = log(Cc)", "ddt_auc=Cc")`

Example 1

We are using in this example the model "newoutput.txt":

Suppose that we want to compute two variables which are not defined in the model:

- the so-called "area under the curve"" $auc(t) = \int_0^t Cc(u)du$ where Cc is the concentration defined in the model,
- the observed log-concentration $z = log(y)$.

We just need for that to define the input argument addlines with 2 additional formula:

```
add1 <- list(formula=c("ddt_auc = Cc", "z = log(y)"))

adm <- list(time=seq(0,66,by=12), amount=100)
out1 <- list(name= c("Cc","auc"), time=seq(0,100, by=0.5))
out2 <- list(name=c("y","z"), time=seq(18, 80, by=6))
p <- c(V_pop=10, omega_V=0.3, w=50, k=0.2, a=0.2)

res2 <- simulx(model     = "model/newoutput.txt",
               addlines  = add1,
               output    = list(out1, out2),
               parameter = p,
               treatment = adm)
```

Then, `auc`and `z` are part of the outputs:

```
names(res2)
```

```
## [1] "treatment"
```

```
head(merge(res2$Cc,res2$auc))
```

```
## data frame with 0 columns and 0 rows
```

```
head(merge(res2$y,res2$z))
```

```
## data frame with 0 columns and 0 rows
```

We could also define Vn as a new individual parameter:

```
add2  = list(section="[INDIVIDUAL]", block="DEFINITION:",
                formula=c("Vn = {distribution=normal, prediction=Vpred, sd=1}"))

out3 <- list(name= c("Vn", "V"))
res3 <- simulx(model     = "model/newoutput.txt",
               addlines  = list(add1, add2),
               output    = list(out1, out3),
               parameter = p,
               treatment = adm,
               group = list(size=5, level="individual"))
names(res3)
```

```
## [1] "group"     "treatment"
```

```
res3$parameter
```

```
## NULL
```

We can also create groups with different outputs which are not defined in the original model ($z1 = 2y$ for group 1 and $z2 = 20$ for group2 in this example):

```
adm1 <- list(time=seq(0,66,by=6),  amount=50)
adm2 <- list(time=seq(0,66,by=12), amount=100)
p1 <- c(V_pop=10, omega_V=0.3, w=50, k=0.2, a=0.2)
p2 <- c(V_pop=20, omega_V=0.3, w=75, k=0.1, a=0.2)

outc  <- list(name = c('Cc','lc','auc'), time=seq(0,100, by=0.5))
y1 <- list(name="y", time=c(10,30,60))
y2 <- list(name="y", time=c(20, 50))
z1 <- list(name="z1", time=c(10,50))
z2 <- list(name="z2", time=c(20,40,60))
V <- list(name="V")

add3 <- list( formula = c("lc = log(Cc)", "ddt_auc=Cc", "z1=2*y", "z2=20") )

g1 <- list(treatment=adm1, parameter=p1, output=list(y1,z1), size=3, level='individual')
g2 <- list(treatment=adm2, parameter=p2, output=list(y2,z2), size=2, level='individual')

res4 <- simulx(model    = "model/newoutput.txt",
               addlines  = add3,
               output   = list(outc,V),
               group    = list(g1,g2))

names(res4)
```

```
## [1] "group"     "treatment"
```

```
res4$z1
```

```
## NULL
```

```
res4$z2
```

```
## NULL
```

Example 2

This feature is particularly useful when simulx is used with a Monolix project and when we don't want to modify the original model used by Monolix.

Computing the AUC, for instance, only requires defining this new variable in addlines:

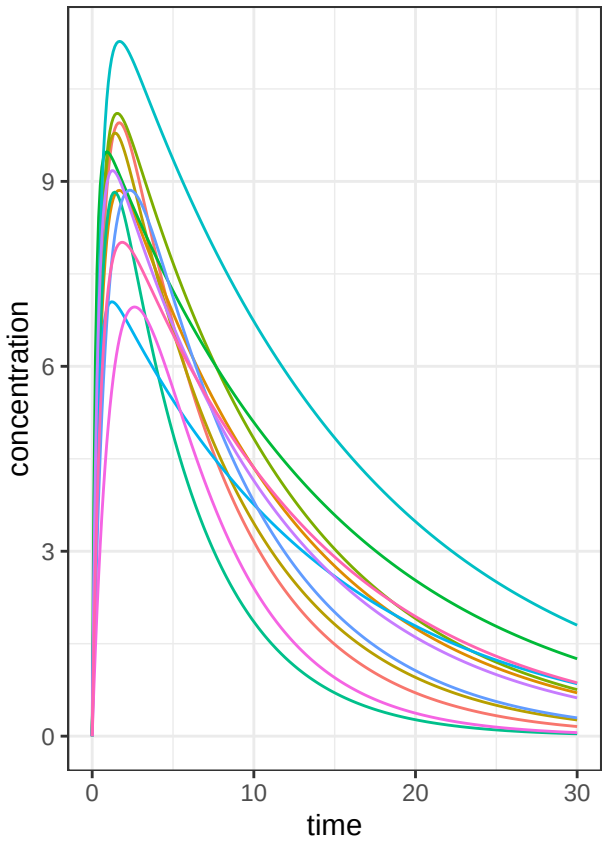
```
res5  <- simulx(project  = 'monolixRuns/theophylline_project.mlxtran',
                output   = list(name = c('Cc','auc'), time = seq(0, 30, by=0.1)),
                addlines = list(formula="ddt_auc = Cc"))

names(res5)
```

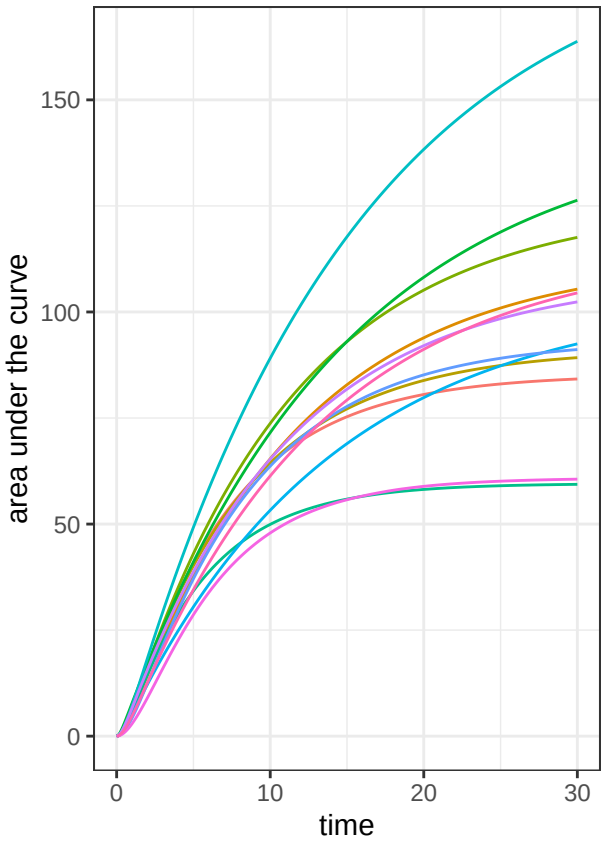
```
## [1] "CONC"       "Cc"         "auc"        "treatment"  "originalId"
## [6] "population"
```

```
head(merge(res5$Cc,res5$auc))
```

```
##   id time       Cc        auc
## 1  1  0.0 0.000000 0.00000000
## 2  1  0.1 1.810706 0.09309949
## 3  1  0.2 3.336371 0.35264549
## 4  1  0.3 4.618018 0.75222906
## 5  1  0.4 5.690817 1.26926004
## 6  1  0.5 6.584921 1.88440376
```

```
library(gridExtra)

pl1 <- ggplot() +  geom_line(data=res5$Cc, aes(x=time, y=Cc, colour=id)) +
        xlab("time") + ylab("concentration") + theme(legend.position="none")

pl2 <- ggplot() +  geom_line(data=res5$auc, aes(x=time, y=auc, colour=id)) +
        xlab("time") + ylab("area under the curve") + theme(legend.position="none")

grid.arrange(pl1, pl2, ncol=2)
```


concentration
9
6
3
0
0
10
20
30
time


area under the curve
150
100
50
0
0
10
20
30
time

Writing the results to a file or a folder

R script: writeresults.R

Writing simulated data

Writing results in a single file

Let's start with a PKPD model:

```
pkpd.model <- inlineModel("
[LONGITUDINAL]
input = {V, Cl, EC50, a1, a2}

EQUATION:
Cc = pkmodel(V, Cl)
E  = 100*Cc/(Cc+EC50)

DEFINITION:
y1 ={distribution=lognormal, prediction=Cc, sd=a1}
y2 ={distribution=normal,    prediction=E,  sd=a2}

[INDIVIDUAL]
input={V_pop,o_V,Cl_pop,o_Cl,EC50_pop,o_EC50}

DEFINITION:
V   ={distribution=lognormal, prediction=V_pop,   sd=o_V}
Cl  ={distribution=lognormal, prediction=Cl_pop,  sd=o_Cl}
EC50={distribution=lognormal, prediction=EC50_pop,sd=o_EC50}
")

p <- c(V_pop=10, o_V=0.1, Cl_pop=1, o_Cl=0.2, EC50_pop=3, o_EC50=0.2, a1=0.1, a2=1)
adm  <- list(amount=100, time=seq(0,50,by=12))
y1 <- list(name=c('y1'), time=seq(5,to=50,by=5))
y2 <- list(name='y2', time=seq(2,to=50,by=6))
```

We can save the outputs y1 and tt>y2 in a file "res1a.csv" (using the standard NONMEM/Monolix format), thanks to the additional input argument result.file:

```
res1 <- simulx(model     = pkpd.model,
               treatment = adm,
               parameter = p,
               group     = list(size=5, level="individual"),
               output    = list(y1,y2),
               result.file   ="res1a.csv",
               settings      = list(seed = 32323))

print(head(read.table("res1a.csv", header=T, sep=",")))
```

```
##   id time      y ytype amount
## 1  1    0      .     .    100
## 2  1    2   77.2     2      .
## 3  1    5 6.4569     1      .
## 4  1    8 64.444     2      .
## 5  1   10 3.6855     1      .
```

```
## 6  1   12      .     .    100
```

Remark: These outputs y1 and tt>y2 are not anymore outputs of `simulx`:

```
names(res1)
```

```
## [1] "group"
```

Results can be saved in various types of files, using different separators. The separator and the number of digits are additional settings:

```
res1 <- simulx(model     = pkpd.model,
               treatment = adm,
               parameter = p,
               group     = list(size=5, level="individual"),
               output    = list(y1,y2),
               result.file   ="res1a.txt",
               settings      = list(seed = 32323, sep="\t", digits=1))
print(head(read.table("res1a.txt", header=T, sep="\t")))
```

```
##   id time    y ytype amount
## 1  1    0    .     .    100
## 2  1    2 77.2     2      .
## 3  1    5  6.5     1      .
## 4  1    8 64.4     2      .
## 5  1   10  3.7     1      .
## 6  1   12    .     .    100
```

Writing results in separated files

Results can be saved in separated files instead of a single one:

```
res1 <- simulx(model     = pkpd.model,
               treatment = adm,
               parameter = p,
               group     = list(size=5, level="individual"),
               output    = list(y1,y2),
               result.folder ="res1a",
               settings      = list(seed = 32323))
print(list.files(path="res1a"))
```

```
## [1] "treatment.csv" "y1.csv"        "y2.csv"
```

Both the output file and the output folder can be defined

```
res1 <- simulx(model     = pkpd.model,
               treatment = adm,
               parameter = p,
               group     = list(size=5, level="individual"),
               output    = list(y1,y2),
               result.file   ="res1b.csv",
               result.folder ="res1b",
               settings      = list(seed = 32323))
print(head(read.table("res1b.csv", header=T, sep=",")))
```

```
##   id time      y ytype amount
## 1  1    0      .     .    100
## 2  1    2   77.2     2      .
## 3  1    5 6.4569     1      .
```

```
## 4  1    8 64.444     2      .
## 5  1   10 3.6855     1      .
## 6  1   12      .     .    100
```

```
print(list.files(path="res1b"))
```

```
## [1] "treatment.csv" "y1.csv"        "y2.csv"
```

Adding individual parameters

individual parameters can be saved as well in the data file

```
res1 <- simulx(model     = pkpd.model,
               treatment = adm,
               parameter = p,
               group     = list(size=5, level="individual"),
               output    = list(y1,y2, c("V" ,"Cl", "EC50")),
               result.file   ="res1c.csv",
               settings      = list(seed = 32323))
print(head(read.table("res1c.csv", header=T, sep=",")))
```

```
##   id time      y ytype amount      V      Cl   EC50
## 1  1    0      .     .    100 9.4832 0.90628 2.5527
## 2  1    2   77.2     2      . 9.4832 0.90628 2.5527
## 3  1    5 6.4569     1      . 9.4832 0.90628 2.5527
## 4  1    8 64.444     2      . 9.4832 0.90628 2.5527
## 5  1   10 3.6855     1      . 9.4832 0.90628 2.5527
## 6  1   12      .     .    100 9.4832 0.90628 2.5527
```

Writing results using writeDatamlx

Instead of saving the results ''within" `simulx`, it is possible to first use `simulx` as usual and then, write the results using the `writeDatamlx`function.

```
res2 <- simulx(model     = pkpd.model,
               treatment = adm,
               parameter = p,
               group     = list(size=5, level="individual"),
               output    = list(y1,y2),
               settings  = list(seed = 32323))

writeDatamlx(res2, result.file = "res2.csv")
print(head(read.table("res2.csv", header=T, sep=",")))
```

```
##   id time        y ytype amount
## 1  1    0        .     .    100
## 2  1    2  77.1999     2      .
## 3  1    5  6.45691     1      .
## 4  1    8 64.44424     2      .
## 5  1   10  3.68555     1      .
## 6  1   12        .     .    100
```

```
writeDatamlx(res2, result.folder = "res2")
print(list.files(path="res2"))
```

```
## [1] "treatment.csv" "y1.csv"        "y2.csv"
```

Using a Monolix project

Using a simplified format

```
project.file <- 'monolixRuns/theophylline_project.mlxtran'
res3 <- simulx(project=project.file, result.file="theo1.csv", settings=list(seed=12345))
print(head(read.csv("theo1.csv")))
```

```
##   id time      y amount
## 1  1 0.00      .    320
## 2  1 0.25 2.2614      .
## 3  1 0.57 4.4562      .
## 4  1 1.12 6.2658      .
## 5  1 2.02 8.8488      .
## 6  1 3.82 6.6862      .
```

Using the original format of the data

```
res4 <- simulx(project=project.file, result.file="theo2.csv", settings=list(format.original=TRUE,seed=1
print(head(read.csv("theo2.csv", sep="\t")))
```

```
##   ID AMT AMT.KG TIME   CONC WEIGHT SEX
## 1  1 320   4.02 0.00      .   79.6   M
## 2  1   .      . 0.25 2.2614   79.6   M
## 3  1   .      . 0.57 4.4562   79.6   M
## 4  1   .      . 1.12 6.2658   79.6   M
## 5  1   .      . 2.02 8.8488   79.6   M
## 6  1   .      . 3.82 6.6862   79.6   M
```

Reproducing the same random results

R script: seed.R

Introduction

The same sequence of random numbers can be used for successive runs of simulx by using the field seed of the list settings. The seed should be an integer.

```
s <- list(seed=123456)}
res <- simulx( ... , settings=list(seed=s, ...))
```

Then, the same results will be obtained at each run if the seed is not modified.

Example

we set first the seed to 12345 in this example:

```
myModel = inlineModel("
[LONGITUDINAL]
input =  {a, b, s}
EQUATION:
f = a + b*t
DEFINITION:
y = {distribution=normal, prediction=f, sd=s}
")

res <- simulx(model     = myModel,
              parameter = c(a=10, b=10, s=0.5),
              settings  = list(seed=12345),
              output    = list(name='y',time=(1:5)))
print(res$y)
```

```
##   time        y
## 1    1 19.46056
## 2    2 30.39061
## 3    3 40.30769
## 4    4 50.71170
## 5    5 59.80711
```

We obtain again the same results if we run again simulx with the same seed

```
res <- simulx(model     = myModel,
              parameter = c(a=10, b=10, s=0.5),
              settings  = list(seed=12345),
              output    = list(name='y',time=(1:5)))
print(res$y)
```

```
##   time        y
## 1    1 19.46056
## 2    2 30.39061
## 3    3 40.30769
## 4    4 50.71170
## 5    5 59.80711
```

Change the seed, or don't use it in settings, to obtain different results.

```
res <- simulx(model     = myModel,
              parameter = c(a=10, b=10, s=0.5),
              settings  = list(seed=54321),
              output    = list(name='y',time=(1:5)))
print(res$y)
```

```
##   time        y
## 1    1 19.87482
## 2    2 29.58286
## 3    3 39.44269
## 4    4 50.54345
## 5    5 59.92412
```

```
res <- simulx(model     = myModel,
              parameter = c(a=10, b=10, s=0.5),
              output    = list(name='y',time=(1:5)))
print(res$y)
```

```
##   time        y
## 1    1 20.17442
## 2    2 30.31067
## 3    3 39.93725
## 4    4 49.87375
## 5    5 59.99575
```

Optimizing the computation time

R script: optimcode.R

Introduction

1. When simulx is called for the first time with a given `Mlxtran` model, a `C++` code is automatically generated from this model and automatically compiled. This process takes some time (a few seconds). If simulx is called again with the same model, this process is not performed anymore. The run is then much faster than the first one.
2. For each run of simulx, the design of the simulation must be created and loaded in memory. This process is also time consuming. When several calls of simulx are done successively, a test is automatically performed to check if the design has changed and should be loaded again, or if the design already in memory can be used. The user can also force simulx to create and load the design with the field load.design of settings that takes the values TRUE/FALSE.
3. simulx converts the inputs of the function defined in several lists into a unique list which can be processed by the C++ `mlxLibrary`. It is possible to bypass the execution of this `R` code and use directly the API (Application Programming Interface) of `mlxLibrary` instead of the simulx standard one.

All these techniques for reducing the computation time of `simulx` are shown to be extremly efficient for simulating a very large number of clinical trials.

Example

We will use in this example a quite complex PK model which requires to solve a system of ODEs (a two compartement model with transit compartments and non linear elimination).

Furthermore, each call of `simulx` will simulate PK data for 100 individuals.

```
myModel = inlineModel("
[LONGITUDINAL]
input = {Mtt, Ktr, ka, V, Vm, Km, k12, k21}
EQUATION:
C = pkmodel(Mtt, Ktr, ka, V, Vm, Km, k12, k21)
[INDIVIDUAL]
input = {V_pop, omega_V}
DEFINITION:
V = {distribution=lognormal, prediction=V_pop, sd=omega_V}
")


adm <- list(time=seq(0, 200, by=12), amount=100)
p   <- c(V_pop=10, omega_V=0.2, Mtt=2, Ktr=0.5, ka=1, Vm=10, Km=1, k12=0.5, k21=0.3)
C   <- list(name='C',time=seq(100, 200, by=10))
g   <- list(size = 100, level='individual')
```

Let us run `simulx` and display the computation time for this first run which requires the creation and the compilation of a `C++` code:

```
ptm <- proc.time()
res <- simulx(model = myModel,parameter = p, output = C, treatment = adm, group = g)
print(proc.time() - ptm)
```

```
##    user  system elapsed
##    0.11    0.01    1.27
```

A second run of `simulx` with the same model will be much faster since the model is already compiled:

```
ptm <- proc.time()
res <- simulx(model = myModel,parameter = p, output = C, treatment = adm, group = g)
print(proc.time() - ptm)
```

```
##    user  system elapsed
##    0.11    0.00    0.05
```

Computation time becomes critical if we need to run `simulx` a large number of times (with different parameters values, or if we want to simulate several replicates of the same experience for instance).

As an illustration, let us run 500 times `simulx` and display the total running time:

```
M=500

ptm <- proc.time()
for(i in seq(1,M)){
res <- simulx(model = myModel,
              parameter = p,
              output = C,
              treatment = adm,
              group = g)
}
print(proc.time() - ptm)
```

```
##    user  system elapsed
##   29.83    1.97   23.61
```

`simulx` is quite fast beacause it uses the same compiled model for each run, but also because a test, automatically performed, avoids to load the same design at each run.

Forcing `simulx` to load the design at each run significantly increases the computation time:

```
ptm <- proc.time()
for(i in seq(1,M)){
  res <- simulx(model = myModel,
                parameter = p,
                output = C,
                treatment = adm,
                group = g,
                settings  = list(load.design=TRUE))
}
print(proc.time() - ptm)
```

```
##    user  system elapsed
##   35.83    1.99   28.16
```

`simulx`will return the input of `mlxLibrary` as a list by setting the field data.in to TRUE:

```
dataIn <- simulx(model    = myModel,
                 parameter = p,
                 output    = C,
                 treatment = adm,
                 group     = g,
                 settings  = list(data.in=TRUE))
```

We can then bypass the `R` code by using `simulx` with the input argument data:

```
ptm <- proc.time()
for(i in seq(1,M)){
  dd <- simulx(data     = dataIn,
               settings = list(load.design=FALSE))
}
print(proc.time() - ptm)
```

```
##    user  system elapsed
##   32.29    0.38   17.35
```

Simulating random variables in a block EQUATION:

R script: random.R

Introduction

It is possible to define normal random variables in the `EQUATION` block instead of the `DEFINITION` block of the Mlxtran script, using the syntax

where mu is the mean and sigma the standard deviation of the normal random variable x.

Then x can be used in the `EQUATION` block as any other variable.

Remark: For simulation, we can either draw random variables in the block `EQUATION`, or define its probability distribution in the block `DEFINITION`. This is not the case for other tasks, such estimation, where an explicit description of the probability distributions is required by the estimation algotihms. Then, this definition should be provided in the block `DEFINITION`.

Using random variables in Section [LONGITUDINAL]

Consider the following model

$$y_j = (a + b\,t_j)e^{e_j}$$

where $e_j \sim_{\text{i.i.d.}} \mathcal{N}(0, s^2)$.

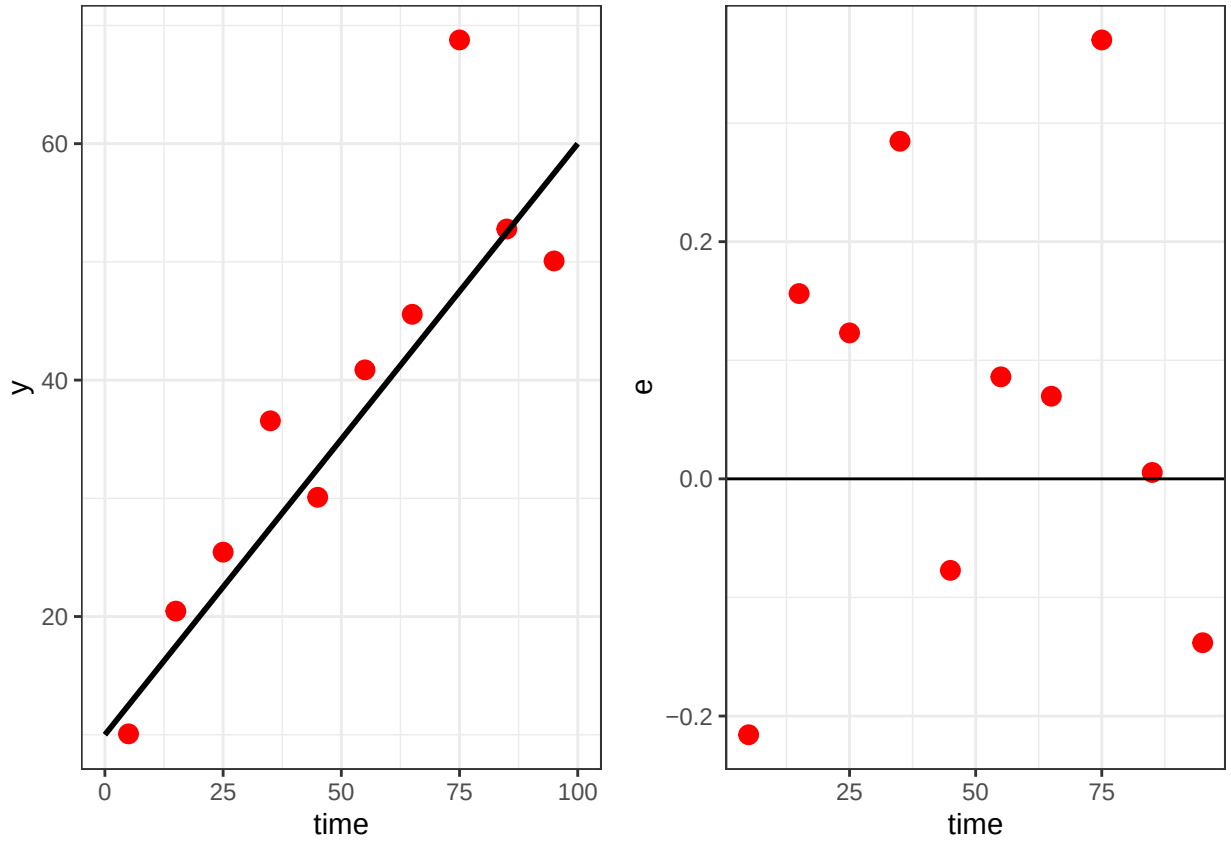
When this model is used for simulation only, it is possible to define the sequence of residual errors (e_j) directly in the block `EQUATION`:

```
myModel1 = inlineModel("
[LONGITUDINAL]
input =  {a, b, s}

EQUATION:
f = a + b*t
e ~ normal(0,s)
y = f*exp(e)
")

f <- list(name='f',time=seq(0, 100, by=1))
y <- list(name='y',time=seq(5, 100, by=10))
e <- list(name='e',time=seq(5, 100, by=10))
p <- c(a=10, b=0.5, s=0.2)
s <- list(seed=12345)

res <- simulx(model     = myModel1,
              parameter = p,
              settings  = s,
              output    = list(f, y, e))
```

We can then plot both sequences (y_j) and (e_j)

```
library(gridExtra)
pl1 <- ggplot() +
  geom_point(aes(x=time, y=y), data=res$y, color="red", size=3) +
  geom_line(aes(x=time, y=f), data=res$f, size=1)
```

```
pl2 <- ggplot(aes(x=time, y=e), data=res$e) +  geom_point(color="red", size=3) +
  geom_hline(yintercept = 0)

grid.arrange(pl1, pl2, nrow=1)
```




```
print(head(res$y))
```

```
##   time        y
## 1    5 10.07395
## 2   15 20.45942
## 3   25 25.44679
## 4   35 36.55680
## 5   45 30.08677
## 6   55 40.86828
```

Remark: In this example, it would be equivalent to define the probability distribution of (y_j) in the block `DEFINITION`:

```
myModel2 = inlineModel("
[LONGITUDINAL]
input =  {a, b, s}

EQUATION:
f = a + b*t

DEFINITION:
y = {distribution=lognormal, prediction=f, sd=s}
```

```
")
res <- simulx(model     = myModel2,
              parameter = p,
              settings  = s,
              output    = list(f, y))

print(head(res$y))
```

```
##   time        y
## 1    5 10.07395
## 2   15 20.45942
## 3   25 25.44679
## 4   35 36.55680
## 5   45 30.08677
## 6   55 40.86828
```

The results are the same because the same seed is used with both models.

Using random variables in Section [INDIVIDUAL]

Random normal variables can also be defined in the bloc `EQUATION`of Section `[INDIVIDUAL]`

```
myModel3 = inlineModel("
[LONGITUDINAL]
input =  {a, b, s}

EQUATION:
f = a + b*t
e ~ normal(0,s)
y = f*exp(e)

[INDIVIDUAL]
input = {a_pop, omega_a, b_pop, omega_b}
EQUATION:
a ~ normal(a_pop,omega_a)
b ~ normal(b_pop,omega_b)
")

i <- list(name=c('a', 'b'))

res <- simulx(model     = myModel3,
              parameter = c(a_pop=10, omega_a=1, b_pop=0.5, omega_b=0.1, s=0.2),
              settings  = list(seed=12345),
              output    = list(y, i))

print(res$parameter)
```

```
##          a         b
## 1 8.921119 0.5781214
```

```
print(head(res$y))
```

```
##   time         y
## 1    5  9.996536
## 2   15 16.610368
## 3   25 19.812616
```

```
## 4   35 39.409807
## 5   45 30.733531
## 6   55 46.279873
```

This is equivalent to defining the probability distributions in the 2 blocs `DEFINITION`:

```
myModel4 = inlineModel("
[LONGITUDINAL]
input =  {a, b, s}
EQUATION:
f = a + b*t
DEFINITION:
y = {distribution=lognormal, prediction=f, sd=s}

[INDIVIDUAL]
input = {a_pop, omega_a, b_pop, omega_b}
DEFINITION:
a = {distribution=normal, mean=a_pop, sd=omega_a}
b = {distribution=normal, mean=b_pop, sd=omega_b}
")

res <- simulx(model     = myModel4,
              parameter = c(a_pop=10, omega_a=1, b_pop=0.5, omega_b=0.1, s=0.2),
              settings  = list(seed=12345),
              output    = list(y, i))

print(res$parameter)
```

```
##          a         b
## 1 8.921119 0.5781214
```

```
print(head(res$y))
```

```
##   time         y
## 1    5  9.996536
## 2   15 16.610368
## 3   25 19.812616
## 4   35 39.409807
## 5   45 30.733531
## 6   55 46.279873
```

Using a R model instead of Mlxtran

R script: Rmodel.R

A basic PK model

A PK model is implemented in `Rmodel/modelMlxt_pk1.txt`:

This model is implemented as a R function in `Rmodel/modelR_pk1.R`

Show the R code

```
modelR_pk1 <- function(parameter,dose,time)
{
  with(as.list(parameter),{
    V_pred<- V_pop*(w/70)
    V <- V_pred*exp(rnorm(1)*omega_V)
    k <- k_pop*exp(rnorm(1)*omega_k)
    t1 <- time[[1]]
    t2 <- time[[2]]
    A1 <- rep(0,length(t1))
    A2 <- rep(0,length(t2))
    d.time <- dose$time
    d.amt <- dose$amount
    nd <- length(d.time)
    for (m in (1:nd)){
      jm1 <- which(t1>=d.time[m])
      A1[jm1] <- A1[jm1] + d.amt[m]*exp(-k*(t1[jm1]-d.time[m]))
      jm2 <- which(t2>=d.time[m])
      A2[jm2] <- A2[jm2] + d.amt[m]*exp(-k*(t2[jm2]-d.time[m]))
    }
    y2 <- A2/V + rnorm(length(A2),0,a)
    C <- data.frame(time=t1, C=A1/V)
    y <- data.frame(time=t2, y=y2)
    r <- list(C,y,V,k)
    return(r)
  })
}
```

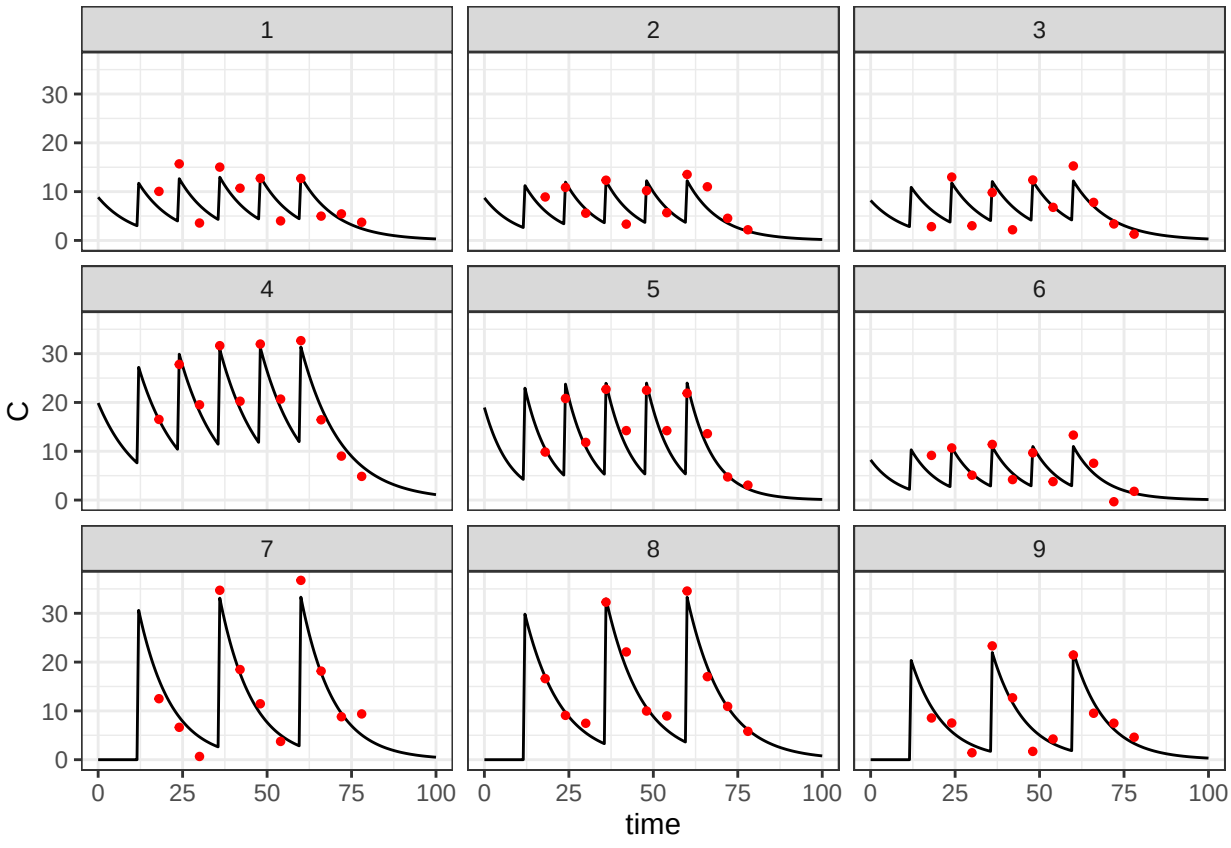
Then, any of these two models can be used with `simulx`

```
pk.model <- "Rmodel/modelR_pk1.R"
## pk.model <- "Rmodel/modelMlxt_pk1.txt"

adm1 <- list(time=seq(0,66,by=12), amount=100)
adm2 <- list(time=seq(12,78,by=24), amount=200)
y <- list(name="y", time=seq(18, 80, by=6))
C <- list(name="C", time=seq(0,100, by=0.5))
ind <- list(name=c("V","k"))
p <- c(V_pop=10, omega_V=0.3, w=50, k_pop=0.1, omega_k=0.2, a=2)
g1 <- list(treatment=adm1, size=6)
g2 <- list(treatment=adm2, size=3)
```

```
res1 <- simulx(model     = pk.model,
               output    = list(C,y,ind),
               parameter = p,
               group     = list(g1,g2))

print(res1$parameter)
```

```
##   group id         V          k
## 1     1  1 11.301211 0.09376232
## 2     1  2 11.465117 0.10375109
## 3     1  3 12.205319 0.09256750
## 4     1  4  5.027077 0.08345587
## 5     1  5  5.273438 0.13036570
## 6     1  6 12.154436 0.11494123
## 7     2  7  6.533914 0.10446714
## 8     2  8  6.706712 0.09390312
## 9     2  9  9.832247 0.10532675
```

```
print(ggplot()  +
        geom_line( data=res1$C, aes(x=time, y=C), colour="black", size=0.5) +
        geom_point(data=res1$y, aes(x=time, y=y), colour="red", size=1)+
        facet_wrap( ~ id))
```



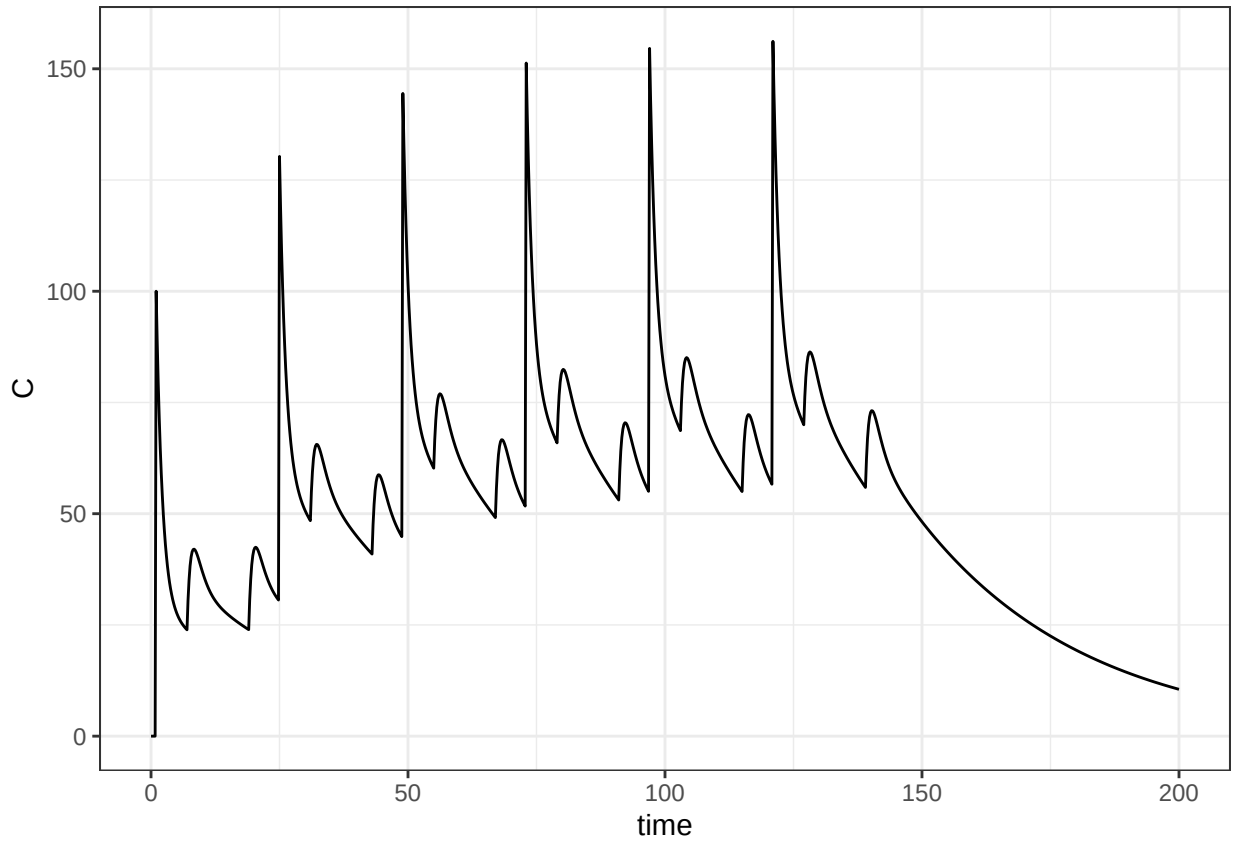
An ODE based model

The ODE based model `Rmodel/modelMlxt_pk2.txt`

can also be implemented in R:

```
library(deSolve)

modelR_pk2 <- function(parameter,dose,time)
{
  MMmod <- function(Time, State, Pars) {
    with(as.list(c(State, Pars)), {
      k <- Cl/V
      ddt_Ad = -ka*Ad
      ddt_Ac = ka*Ad - k*Ac + k21*Ap - k12*Ac + k31*Aq - k13*Ac
      ddt_Ap = -k21*Ap + k12*Ac
      ddt_Aq = -k31*Aq + k13*Ac
      return(list(c(ddt_Ad, ddt_Ac, ddt_Ap, ddt_Aq)))
    })
  }

  yini  <- c(Ad=0, Ac=0, Ap=0, Aq=0)

  t <- time[[1]]
  t <- sort(c(t,dose$data$time))
  out   <- lsode(yini, t, MMmod, parameter, events=dose)
  i0 <- which(diff(t)==0)
  out <- out[-i0,]

  C <- data.frame(time=out[,"time"], C=out[,"Ac"]/parameter["V"])
  r <- list(C=C)
  return(r)
}
```

```
pk.model <- "Rmodel/modelR_pk2.R"
## pk.model <- "Rmodel/modelMlxt_pk2.txt"

adm1 <- list(time=seq(1,144,by=24), amount=10, target="Ac")
adm2 <- list(time=seq(7,144,by=12), amount=4,  target="Ad")
adm <- list(adm1, adm2)
C <- list(name="C", time=seq(0,200, by=0.2))

p = c(ka=1, V=0.1, Cl=0.01, k12=0.2, k21=0.2, k13=0.3, k31=0.3)

res2 <- simulx(model     = pk.model,
               output    = C,
               treatment = adm,
               parameter = p)

print(ggplot(data=res2$C)  + geom_line(aes(x=time, y=C), size=0.5))
```



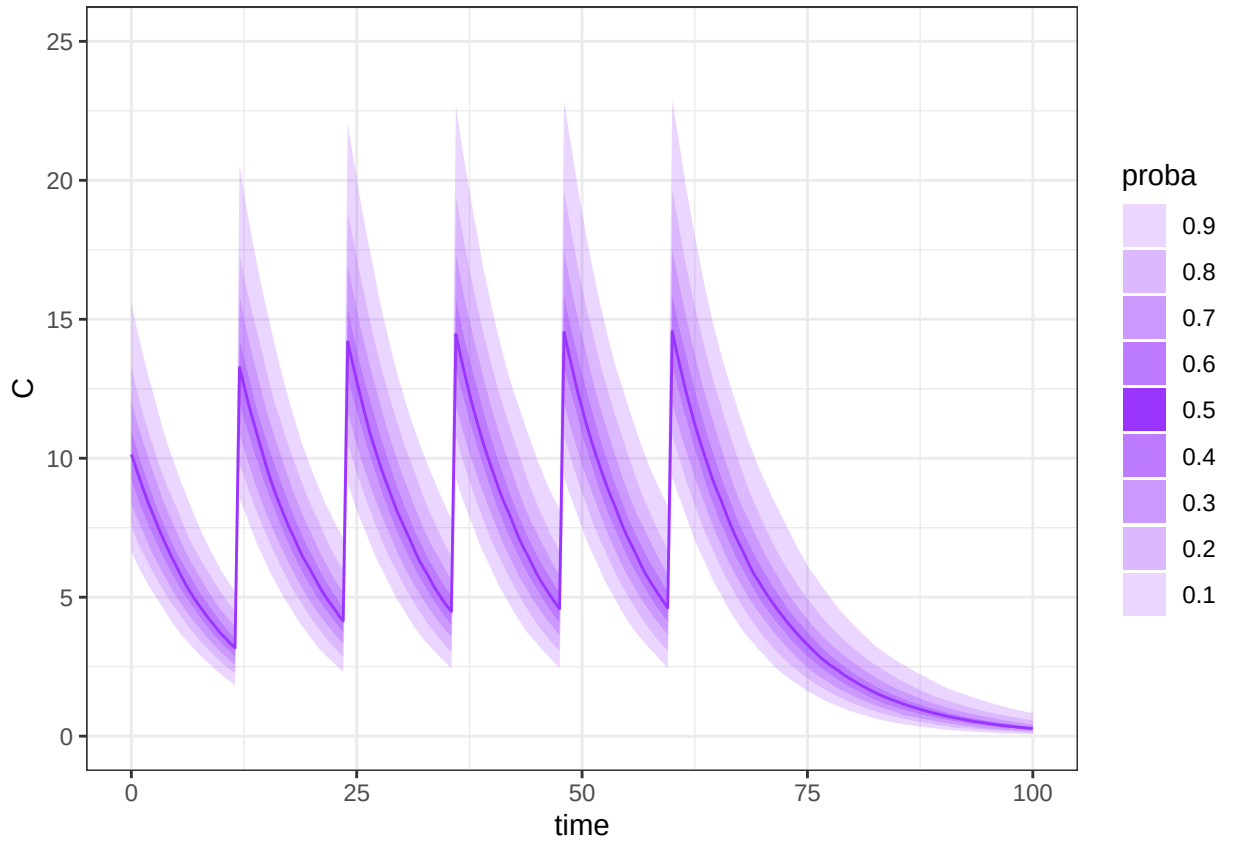
Comparing computational times between R and Mlxtran

Using R instead of Mlxtran can be extremly slow, even with a model which is not based on ODEs

```
adm <- list(time=seq(0,66,by=12), amount=100)
C <- list(name="C", time=seq(0,100, by=0.5))
ind <- list(name=c("w","V","k"))
p <- c(w_pop=70, omega_w=10, V_pop=10, omega_V=0.3, k_pop=0.1, omega_k=0.2)
g <- list(size=1000)

ptm <- proc.time()
res3a <- simulx(model     = "Rmodel/modelR_pk3.R",
                output    = list(C,ind),
                parameter = p,
                treatment = adm,
                group     = g)
print(proc.time() - ptm)
```

```
##    user  system elapsed
##   13.42    0.48   14.20
```

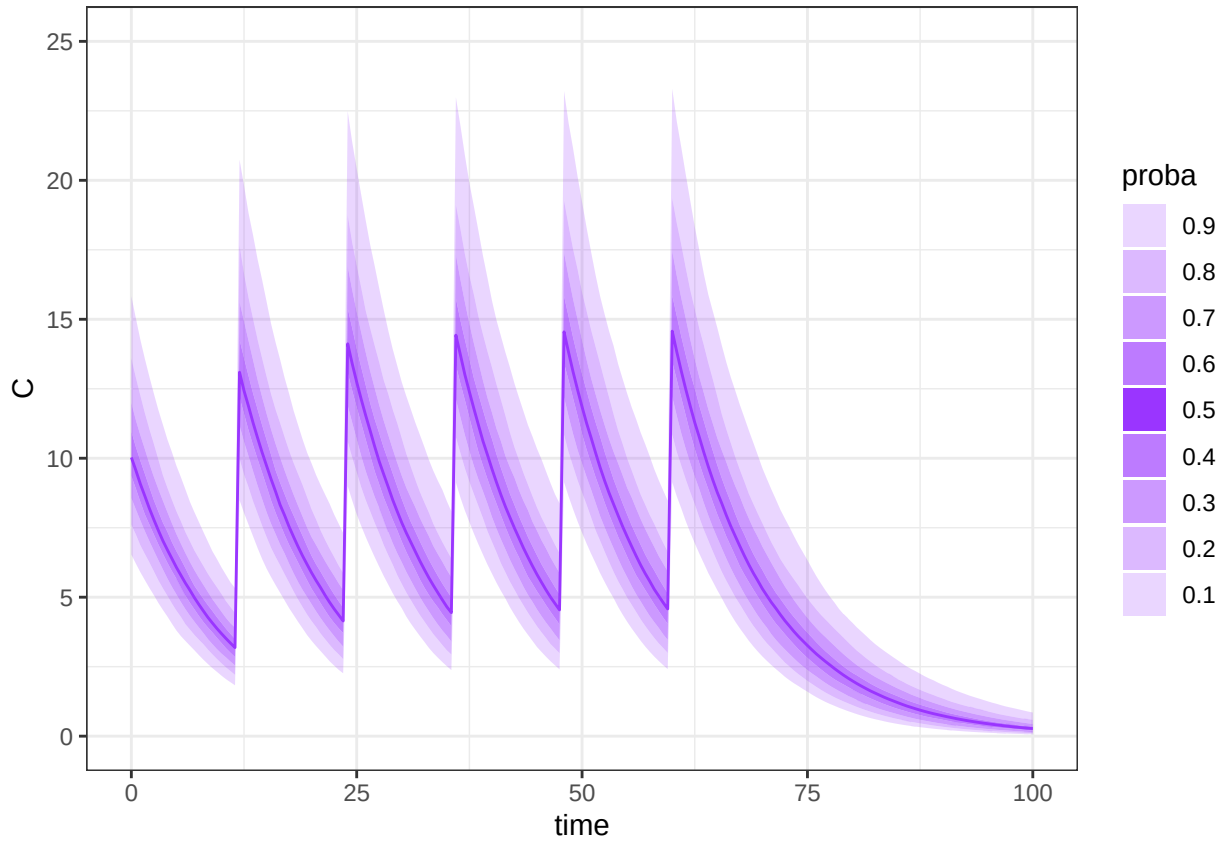
```
print(prctilemlx(res3a$C)+ylim(c(0,25)))
```




```
ptm <- proc.time()
res3b <- simulx(model     = "Rmodel/modelMlxt_pk3.txt",
                output    = list(C,ind),
                parameter = p,
                treatment = adm,
                group     = g)
print(proc.time() - ptm)
```

```
##    user  system elapsed
##    7.22    1.66   10.11
```

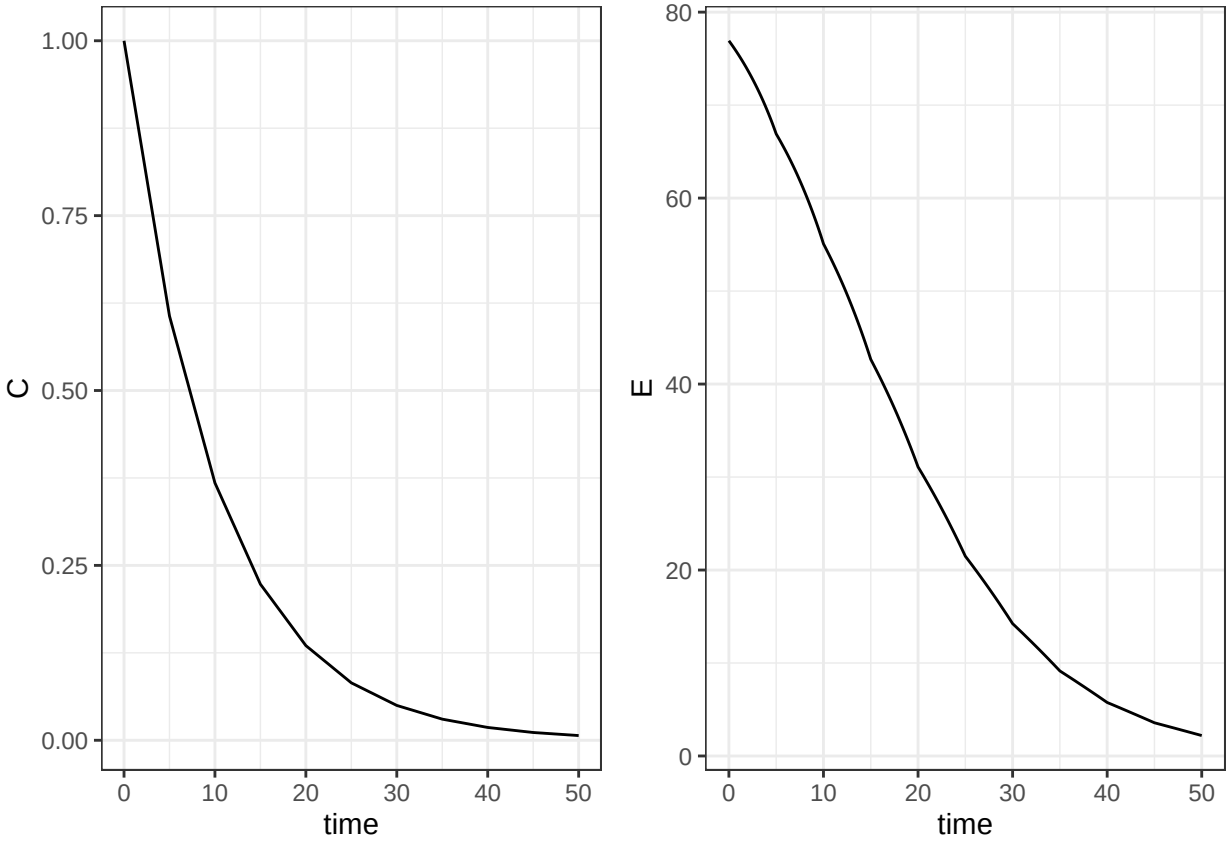
```
print(prctilemlx(res3b$C)+ylim(c(0,25)))
```



A model with regression variables

```
modelR_reg1 <- function(parameter,regressor,time)
{
  with(as.list(parameter),{
    t <- time[[1]]
    regt <- regressor[[1]]$time
    regC <- regressor[[1]]$value
    C <- approx(regt,regC,t)$y
    E = Emax*C/(C+EC50)
    r <- list(C=data.frame(time=t, C=C),
              E=data.frame(time=t, E=E))
    return(r)
  })
}

reg <- data.frame(time=seq(0,50,by=5),  Cin=exp(-0.1*seq(0,50,by=5)))
out <- list(name=c('C','E'), time=seq(0,50, by=0.2))

res4 <- simulx( model     = "modelR_reg1",
                parameter = c(Emax=100, EC50=0.3),
                regressor = reg,
                output    = out)

plot1 <- ggplot(data=res4$C) + geom_line(aes(x=time, y=C))
```

```
plot2 <- ggplot(data=res4$E) + geom_line(aes(x=time, y=E))
gridExtra::grid.arrange(plot1, plot2, ncol=2)
```



‘‘‘