mlxR Package User Guide

Contents

Simulation

Model visualization

Data visualization

Miscellaneous

pkmodel.R

Overview

Description

Easy simulation of basic PK models

Usage

```
pkmodel(time, treatment, parameter)
```

Arguments

`time`

- a vector of observation times

`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),

`parameter`

- vector of parameters with their names and values*

Details

The PK model is defined by the names of the input parameters of the `pkmodel` function. These names are **reserved keywords**.

Absorption

- `p` : Fraction of dose which is absorbed
- `ka` : absorption constant rate (first order absorption)
- or, `Tk0` : absorption duration (zero order absorption)
- `Tlag` : lag time before absorption
- or, `Mtt, Ktr`: mean transit time & transit rate constant

Distribution

- `V` : Volume of distribution of the central compartment,
- `k12, k21` : Transfer rate constants between compartments 1 (central) & 2 (peripheral)
- or `V2, Q2` : Volume of compartment 2 (peripheral) & inter compartment clearance, between compartments 1 and 2,
- `k13, k31` : Transfer rate constants between compartments 1 (central) & 3 (peripheral)
- or `V3, Q3` : Volume of compartment 3 (peripheral) & inter compartment clearance, between compartments 1 and 3.

Elimination

- `k` : Elimination rate constant,
- or `Cl` : Clearance,
- `Vm, Km` : Michaelis Menten elimination parameters.

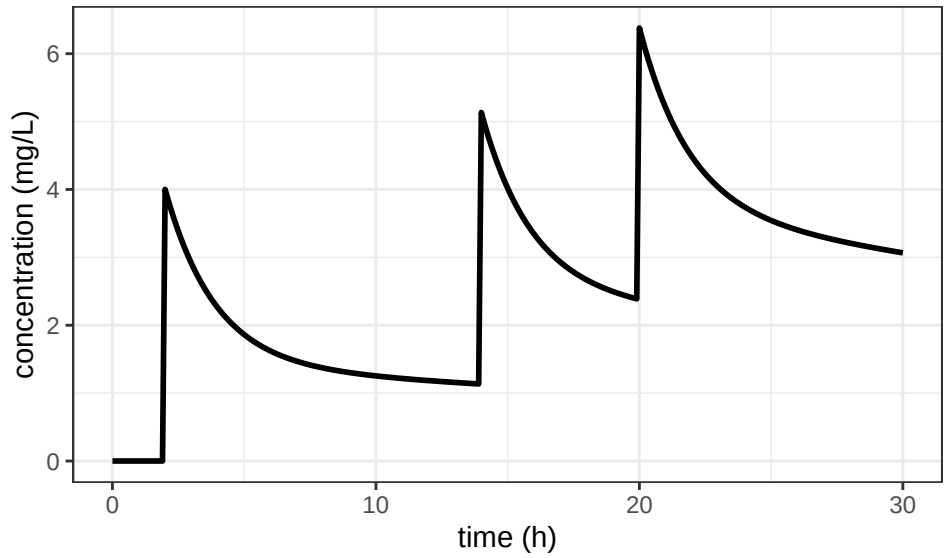
Effect compartment

- `ke0` : Effect compartment transfer rate constant

Examples

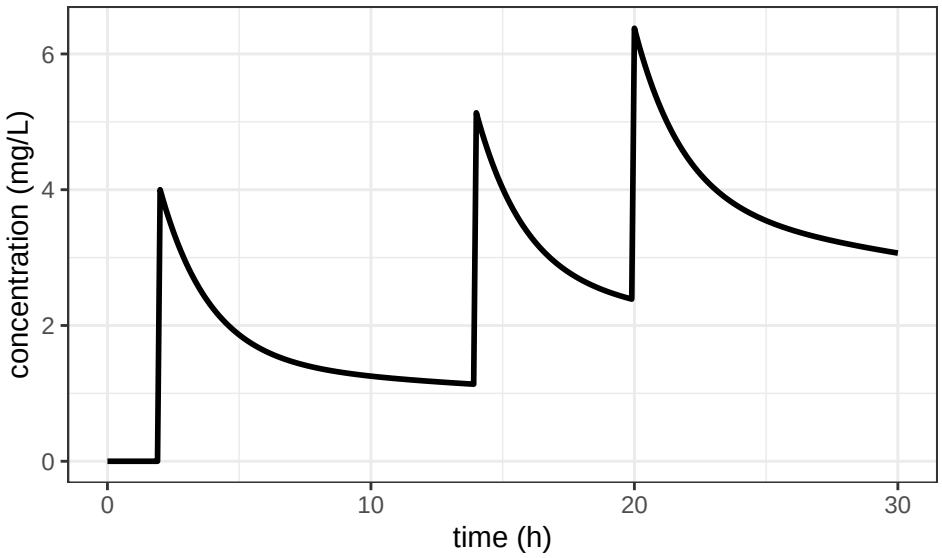
Example 1

```
adm <- list(time=c(2,14,20), amount=40)
t   <- seq(0, 30, by=0.1)
param   <- c(V=10, k=0.05, k12=0.3, k21=0.2)

res   <- pkmodel(time = t, treatment = adm, parameter = param)

print(ggplot(data=res, aes(x=time, y=cc)) + geom_line(size=1) +
  xlab("time (h)") + ylab("concentration (mg/L)"))
```



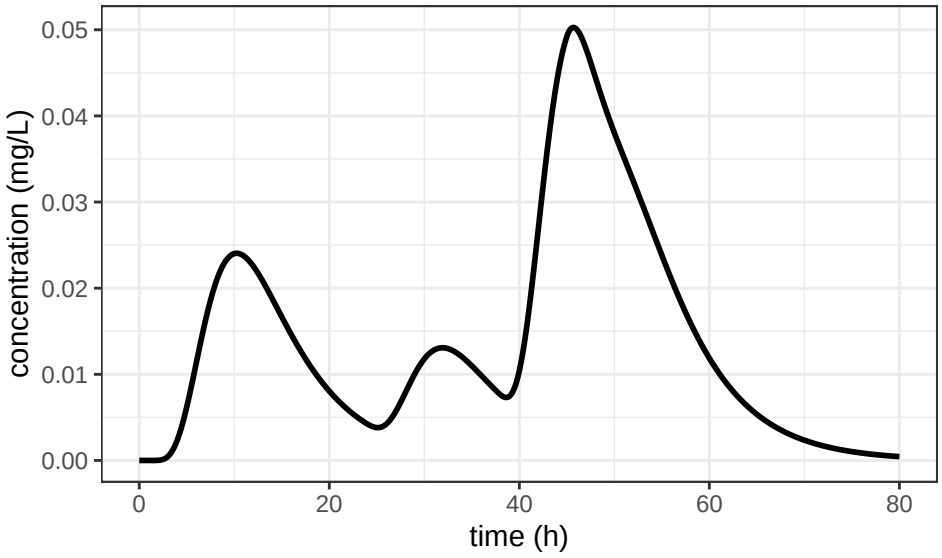
We can use parameterization (V_2, Q_2) instead of (k_{12}, k_{21}):

```
param   <- c(V=10, Cl=0.5, Q2=3, V2=15)
res   <- pkmodel(time = t, treatment = adm, parameter = param)
print(ggplot(data=res, aes(x=time, y=cc)) + geom_line(size=1) +
  xlab("time (h)") + ylab("concentration (mg/L)"))
```



Example 2

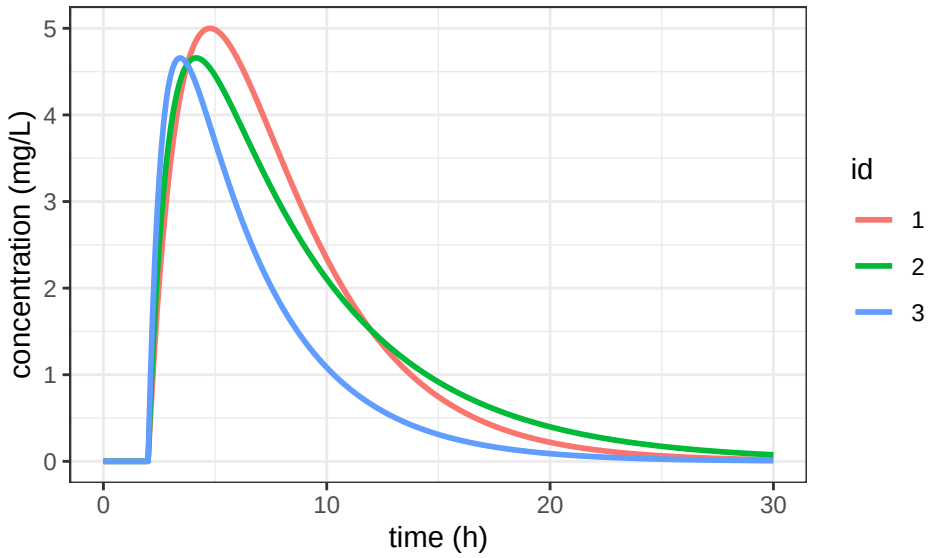
```
adm <- list(time = c(1,23,37,45), amount = c(1,0.5,2,0.3))
param <- c(Mtt=5, Ktr=1, ka=0.5, V=10, Vm=1, Km=0.6, p=0.5)
t <- seq(0, 80, by=0.1)

res <- pkmodel(t,adm,param)

print(ggplot(data=res, aes(x=time, y=cc)) + geom_line(size=1) +
  xlab("time (h)") + ylab("concentration (mg/L)"))
```



Example 3

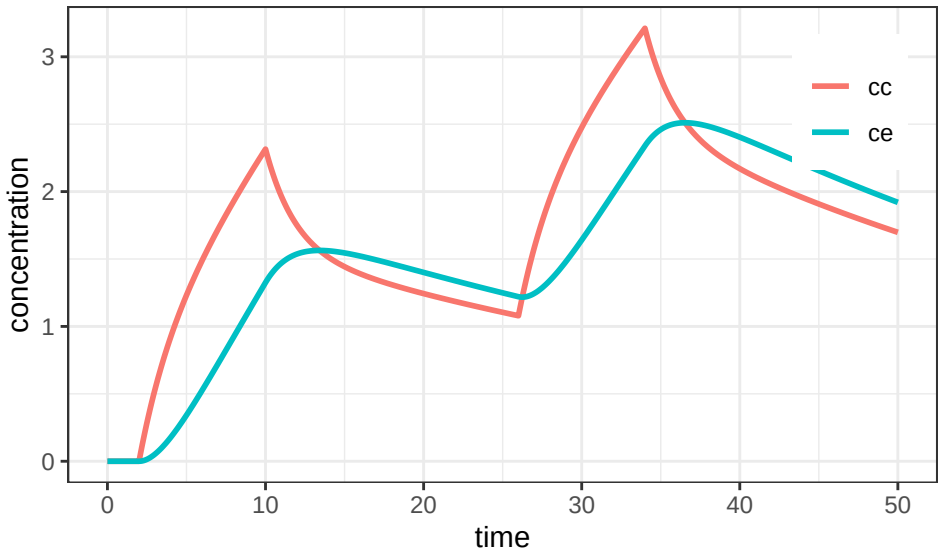
```
adm <- list( time = 2, amount = 40)
```

```
param <- inlineDataFrame("
id   ka   V    Cl
1   0.5   4     1
2     1   6     1
3   1.5   6   1.5
")

t <- seq(0, 30, by=0.1)

res <- pkmodel(t,adm,param)

print(ggplot(data=res, aes(x=time, y=cc, colour=id)) + geom_line(size=1) +
  xlab("time (h)") + ylab("concentration (mg/L)"))
```



Example 4

```
library("reshape2")

adm <- list(time=seq(2, 100, by=24), amount=40, rate=5)
param <- c(V=8, Cl=0.5, k12=0.3, k21=0.2, ke0=0.2)
t <- seq(0, 50, by=0.1)

res <- pkmodel(t,adm,param)

r <- melt(res, id='time', variable.name='c')
print(ggplot(r, aes(time,value)) + geom_line(aes(colour = c),size=1) +
        ylab('concentration') + guides(colour=guide_legend(title=NULL)) +
        theme(legend.position=c(.9, .8)))
```


concentration
time
cc
ce
0
1
2
3
0
10
20
30
40
50

simpopmlx.R

Overview

Description

Draw population parameters using the covariance matrix of the estimates.

Usage

```
simpopmlx(n = 1, project = NULL, fim = "needed", parameter = NULL, corr = NULL, kw.max = 100)
```

Arguments

`n`

- the number of vectors of population parameters (default = 1)

`project`

- a Monolix project, assuming that the Fisher information Matrix was estimated by Monolix

`fim`

- the Fisher Information Matrix estimated by Monolix. fim="sa", "lin" (default="sa")

`parameter`

- a data frame with a column pop.param (no default), a column sd (no default), and possibly a column trans (default ='N'). Only when project is not used

`corr`

- correlation matrix of the population parameters (default = identity). Only when project is not used

`kw.max`

- maximum number of trials for generating a positive definite covariance matrix (default = 100)

Examples

Using a Monolix project

We use the Monolix project `warfarinPK1_project`.

In this project, the Fisher Information Matrix was estimated both by linearization and by stochastic approximation. By default, `simpopmlx` uses the FIM estimated by stochastic approximation (`sa`).

```
project.file <- 'monolixRuns/warfarinPK1_project.mlxtran'
pop1a <- simpopmlx(n=3, project=project.file)
print(pop1a, digits=3)
```

```
##   pop ka_pop V_pop Cl_pop omega_ka omega_V omega_Cl     a     b
## 1   1  0.582  7.73  0.128    0.785   0.172    0.253 0.814 0.093
## 2   2  0.379  7.37  0.127    0.359   0.189    0.276 0.704 0.127
## 3   3  0.589  7.83  0.138    0.614   0.225    0.256 0.754 0.105
```

the FIM estimated by linearization can be used by setting `fim="lin"`

```
pop1b <- simpopmlx(n=3, project=project.file, fim="sa")
print(pop1b, digits=3)
```

```
##   pop ka_pop V_pop Cl_pop omega_ka omega_V omega_Cl     a     b
## 1   1  0.490  7.97  0.145    0.713   0.246    0.335 0.745 0.103
## 2   2  0.372  8.48  0.129    0.787   0.249    0.273 0.824 0.125
## 3   3  0.543  7.32  0.131    0.642   0.184    0.321 0.865 0.088
```

In the project `warfarinPK2_project`, $ka_{\text{ppop}} = 0.5$ (fixed) and a correlation between η_{ka} and η_V is estimated.

```
project.file <- 'monolixRuns/warfarinPK2_project.mlxtran'
pop2 <- simpopmlx(n=3, project=project.file)
print(pop2, digits=3)
```

```
##   pop ka_pop V_pop Cl_pop omega_ka omega_V omega_Cl corr_V_Cl     a      b
## 1   1    0.5  8.36  0.132    0.498   0.222    0.252     0.443 0.745 0.0474
## 2   2    0.5  7.55  0.134    0.428   0.255    0.320     0.396 0.453 0.1031
## 3   3    0.5  7.64  0.131    0.552   0.220    0.266     0.405 0.678 0.0637
```

In the project `warfarinPK3_project`, $\omega_{ka} = 0$ (fixed) and the weight is put both on V and Cl.

```
project.file <- 'monolixRuns/warfarinPK3_project.mlxtran'
pop3 <- simpopmlx(n=3, project=project.file)
print(pop3, digits=3)
```

```
##   pop ka_pop V_pop beta_V_lw70 Cl_pop beta_Cl_lw70 omega_ka omega_Cl     a      b
## 1   1    0.5  7.77       0.788  0.142        0.301    0.652    0.245 0.458 0.1142
## 2   2    0.5  7.58       0.709  0.131        0.344    0.563    0.243 0.374 0.1278
## 3   3    0.5  7.68       0.802  0.133        0.626    0.879    0.252 0.600 0.0928
```

Using a data frame

In this example, the three first parameters and the seventh one are normally distributed, the three other ones are log-normally distributed. Then, `pop.param` are the medians of these distributions while `sd` are the strandard deviations of the normal distributions.

```
param <- data.frame(pop.param=c(1.5, 0.5, 0.02, 0.4, 0.15, 0.2, 0.7),
                    sd=c(0.2, 0.05, 0.004, 0.05, 0.02, 0.02, 0.05),
                    trans=c('N','N','N','L','L','L','N'))
pop4a <- simpopmlx(n=3, parameter=param)
print(pop4a, digits=3)
```

```
##   pop    1     2      3     4     5     6     7
## 1   1 0.97 0.510 0.0192 0.308 0.150 0.204 0.786
## 2   2 1.54 0.478 0.0244 0.312 0.173 0.192 0.731
## 3   3 1.40 0.517 0.0188 0.441 0.121 0.219 0.668
```

```
param$sd=c(0.2, 0.05, 0., 0.05, 0.02, 0.02, 0.0)
pop4b <- simpopmlx(n=3, parameter=param)
print(pop4b, digits=3)
```

```
##   pop    1     2    3     4     5     6   7
## 1   1 1.45 0.538 0.02 0.361 0.134 0.192 0.7
## 2   2 1.14 0.512 0.02 0.475 0.139 0.193 0.7
## 3   3 1.32 0.479 0.02 0.358 0.131 0.182 0.7
```

Logit-normal distributions can be used with the alias 'G'. By default, the support of a logit-normal distribution is (0,1).

```
param <- data.frame(pop.param=c(1.5, 0.5, 0.02, 0.4, 0.15, 0.2, 0.7),
                    sd=c(0.2, 0.5, 0.004, 0.05, 0.02, 0.02, 0.05),
                    trans=c('N','G','N','L','L','L','N'))
```

```
pop5a <- simpopmlx(n=3, parameter=param)
print(pop5a, digits=3)
```

```
##   pop    1     2      3     4     5     6     7
## 1   1 1.26 0.583 0.0263 0.405 0.149 0.197 0.795
## 2   2 1.30 0.110 0.0227 0.414 0.136 0.195 0.686
## 3   3 1.26 0.943 0.0159 0.317 0.186 0.194 0.685
```

additional columns lim.a and lim.b should be added in order to define a logit-normal distribution on interval (a, b)

```
param$lim.a <- c(NaN, 0.2, NaN, NaN, NaN, NaN, NaN)
param$lim.b <- c(NaN, 0.7, NaN, NaN, NaN, NaN, NaN)
pop5b <- simpopmlx(n=3, parameter=param)
print(pop5b, digits=3)
```

```
##   pop    1     2      3     4     5     6     7
## 1   1 1.42 0.696 0.0152 0.441 0.161 0.185 0.689
## 2   2 1.69 0.484 0.0170 0.402 0.215 0.238 0.651
## 3   3 1.45 0.690 0.0237 0.412 0.157 0.210 0.684
```

exposure.R

Overview

Description

Compute the area under the curve (AUC), the maximum and minimum values of a function of time over a given interval, or at steady state.

The AUC over interval (a, b) is computed using the trapezoidal formula for integration

$$I(a,b) = \sum_{j=1}^{n-1} (t_{j+1} - t_j) \left(\frac{f(t_{j+1}) + f(t_j)}{2} \right).$$

where $t_1 = a < t_2 < \ldots < t_n = b$.

Cmax and Cmin are defined as

$$C_{\max} = \max_{1 \leq j \leq n} f(t_j) \quad ; \quad C_{\min} = \min_{1 \leq j \leq n} f(t_j)$$

Then, Tmax and Tmin are defined by

$$C_{\max} = f(T_{\max}) \quad ; \quad C_{\min} = f(T_{\min})$$

The times $(t_j, 1 \leq j \leq n)$ are either provided as an input to exposure, or automatically determined if exposure is computed at steady state.

Usage

```
exposure(model, output, group = NULL, treatment = NULL, parameter = NULL, data = NULL,
         project = NULL, settings = NULL, regressor = NULL, varlevel = NULL)
```

Arguments

Input arguments are the input arguments of the `simulx` function.

Specific input arguments can be also used for computing the exposure at steady state, i.e. after the administration of an "infinite" number of doses:

`output`

- a list with fields
  - `name` : a vector of output names
  - `time` : = 'steady.state'
  - `ntp` : number of time points used for computing the exposure (default=100)
  - `tol` : tolerance number, between 0 and 1, for approximating steaty-state (default=0.01)
  - `ngc` : number of doses used for estimating the convergence rate to steaty-state (default=5).

`treatment`

- a list with fields
  - `tfd` : first time of dose
  - `ii` : interdose interval
  - `amount` : amount

- `rate` : rate of infusion
- `tinf` : time of infusion

- `type` : the type of input

- `target` : the target compartment

Value

A list with several elements: * One data frame per output is created with columns `id` (if number of subject >1), `group` (if number of groups >1), `t1` (beginning of time interval), `t2` (end of time interval), `step` (time step), `auc` (area under the curve), `tmax` (time of maximum value), `cmax` (maximum value), `tmin` (time of minimum value), `cmin` (minimum value). * a list called `output` contains the values of the outputs (outputs of simulx)

Examples

Example 1: AUC for a one compartmental model

R script: doc_exposure1.R

Computing the exposure on a given interval We first compute the exposure over 24 hours for a one-compartment model after a single oral administration.

The concentration is computed between 0 and 24 h every 0.2 h.

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

p <- c(ka=0.5, V=10, Cl=1)

adm1 <- list(time=0, amount=100)
out1 <- list(name="Cc", time=seq(0, 24, by=0.2))

res1 <- exposure(model=myModel, parameter=p, output=out1, treatment=adm1)
```

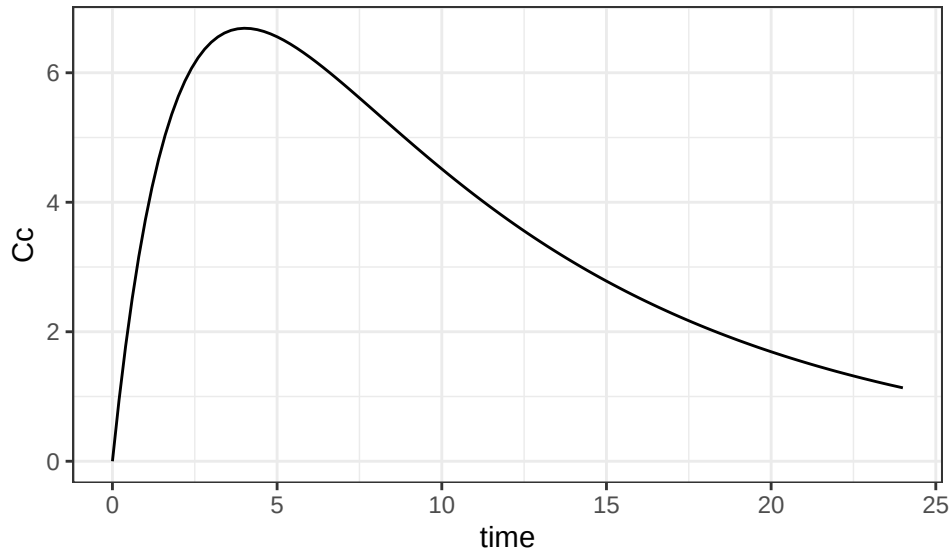
`res1` is a list with two elements:

```
names(res1)
```

```
## [1] "Cc"     "output"
```

`res1$output` is a list which contains only one data frame `Cc` with the values of the concentration computed between 0 and 24 h every 0.2 h.

```
ggplot(data=res1$output$Cc) + geom_line(aes(x=time,y=Cc))
```



`res1$Cc` is a data frame which contains several informations, such as the limits of the time interval, the time step, the area under the curve (AUC), the minimum and maximum concentration values over the interval.

```
print(res1$Cc)
```

```
##   t1 t2 step      auc tmax    cmax tmin cmin
## 1  0 24  0.2 88.64337    4 6.68731    0    0
```

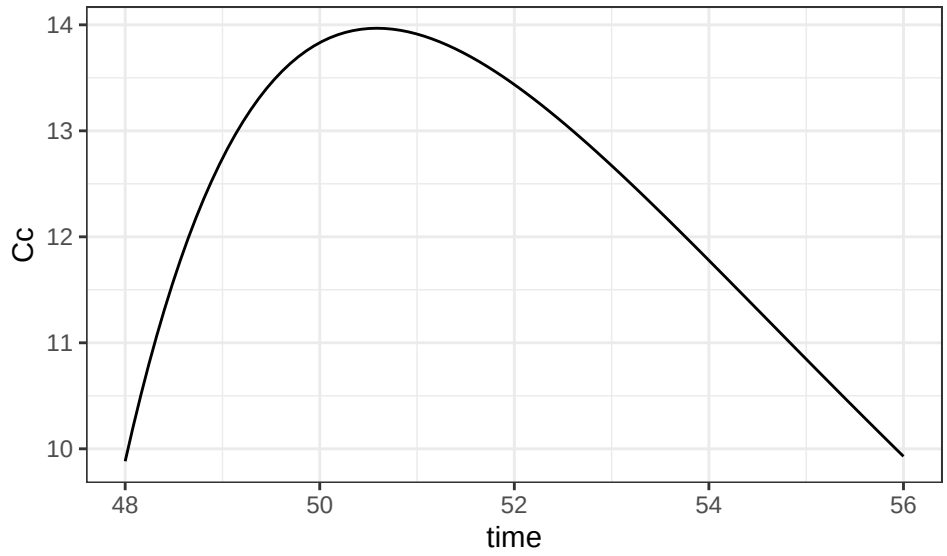
We can also compute the exposure after multiple administrations, given every 8 hours in this example.

```
adm2 <- list(time=seq(0,160,by=8), amount=100)
out2 <- list(name="Cc", time=seq(160, 168, by=0.1))

res2 <- exposure(model=myModel, parameter=p, output=out2, treatment=adm2)
print(res2$Cc)
```

```
##    t1  t2 step      auc  tmax     cmax tmin    cmin
## 1 160 168  0.1 99.99583 162.6 14.03234  160 9.96636
```

Computing the exposure at steady state When the number of doses increases, the system reaches its *steady state*: the PK profile tends to be the same between successive administrations. We can then compute the exposure between two doses by setting `time='steady.state'` in the definition of the output.

```
adm3 <- list(tfd=0, ii=8, amount=100)
out3a <- list(name="Cc", time='steady.state')

res3a <- exposure(model=myModel, parameter=p, output=out3a, treatment=adm3)

ggplot(data=res3a$output$Cc) + geom_line(aes(x=time,y=Cc))
```




```
print(res3a$Cc)
```

```
##   t1 t2       step      auc     tmax     cmax tmin     cmin
## 1 48 56 0.08080808 99.53504 50.58586 13.96767   48 9.882421
```

Steady state is approximated by a multiple dose administration. The number of dose is determined according to the desired precision of the approximation. By default, the *tolerance*, defined here as the relative error of the AUC, is 1%. Better approximation is obtained with a smaller value of `tol`.

```
out3b <- list(name="Cc", time='steady.state', tol=0.001)
res3b <- exposure(model=myModel, parameter=p, output=out3b, treatment=adm3)
print(res3b$Cc)
```

```
##   t1 t2       step      auc     tmax    cmax tmin     cmin
## 1 72 80 0.08080808 99.95535 74.58586 14.0266   72 9.958746
```

Convergence to steady state is approximately geometric: the relative error decreases as $e_j \approx \alpha^j$ where $0 < \alpha < 1$. This relationship is exact in the case of repeated iv bolus administrations, with $\alpha = 2^{-\delta/t_{1/2}}$, where δ is the inter-dose time interval and $t_{1/2}$ the half-life.

For a given tolerance `tol`, the number of doses to use is approximately $\log(\text{tol})/\log(\alpha)$.

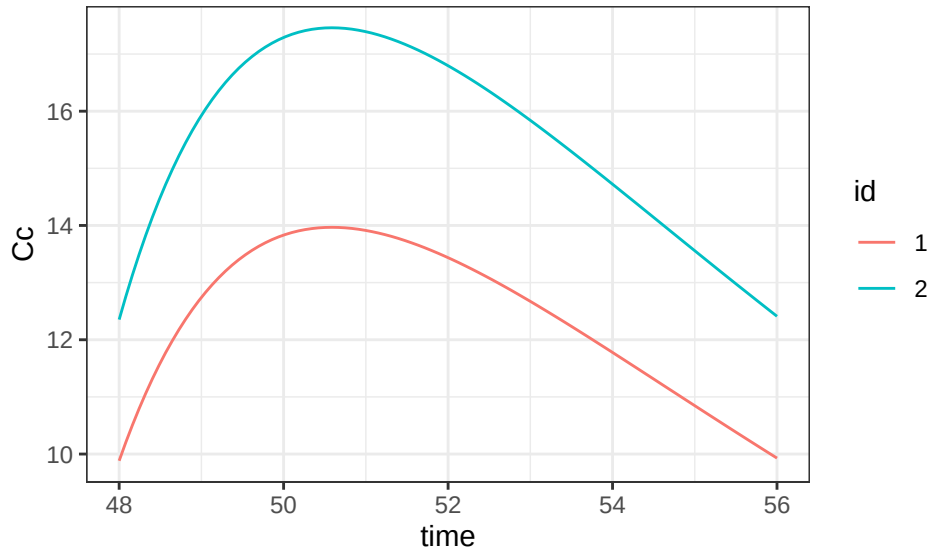
`ngc` is the number of doses used for estimating the convergence rate to steady state, i.e for estimating α. By default, `ngc=5`.

Using `exposure` with several individual and groups\ We compare in this example the exposure for two different sets of PK parameters. The input argument `parameter` is a data frame with two rows.

```
p3 <- inlineDataFrame("
 id   ka   V   Cl
  1  0.5  10    1
  2  0.5   8  0.8
")
res3c <- exposure(model=myModel, parameter=p3, output=out3a, treatment=adm3)
print(res3c$Cc)
```

```
##   id t1 t2       step       auc     tmax     cmax tmin      cmin
## 1  1 48 56 0.08080808  99.53504 50.58586 13.96767   48  9.882421
## 2  2 48 56 0.08080808 124.41880 50.58586 17.45959   48 12.353026
```

```
ggplot(data=res3c$output$Cc) + geom_line(aes(x=time,y=Cc, color=id))
```



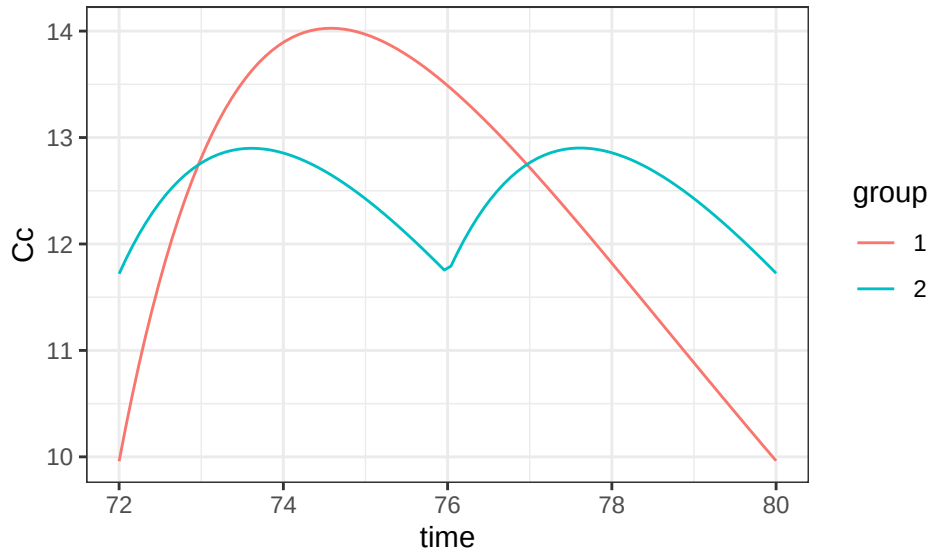
We can also compare the exposure for different dosage regimens. One group receives 100 mg every 8 hours while the other group receives 50 mg every 4 hours.

Function **exposure** automatically determines a common time interval for both groups (an interval of 8 hours in this example).

```
g1 <- list(treatment=list(ii=8, amount=100))
g2 <- list(treatment=list(ii=4, amount=50))
out4 <- list(name="Cc", time='steady.state', tol=0.001)

res4 <- exposure(model=myModel, parameter=p, output=out4, group=list(g1,g2))
print(res4$Cc)
```

```
##   id group t1 t2       step      auc     tmax     cmax tmin      cmin
## 1  1     1 72 80 0.08080808 99.95535 74.58586 14.02660   72  9.958746
## 2  2     2 72 80 0.08080808 99.94707 77.57576 12.90107   72 11.720057
```

```
ggplot(data=res4$output$Cc) + geom_line(aes(x=time,y=Cc, color=group))
```



Example 2: AUC for multiple administrations

R script: doc_exposure2.R

When, for the same individual, several administrations are combined with different times of first dose and different inter-dose time intervals, **exposure** automatically determines a common time interval. The length of this interval is the least common multiple of the time intervals associated to each administration.

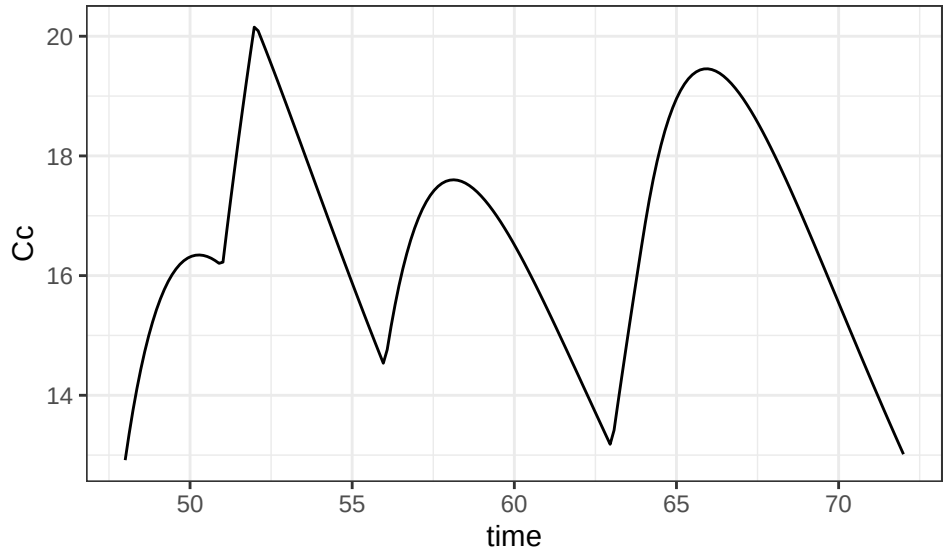
In this example, the inter-dose intervals are, respectively 8h and 12h. The PK profile obtained by combining these two administrations is therefore periodic at steady state and the period is 24h.

```
myModel <- inlineModel("
[LONGITUDINAL]
input = {ka, V, Cl}

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

EQUATION:
ddt_Ad = -ka*Ad
ddt_Ac =  ka*Ad - (Cl/V)*Ac
Cc = Ac/V
")

p <- c(ka=0.5, V=10, Cl=1)

adm1 <- list(tfd=0, ii=8, amount=100, type=1)
adm2 <- list(tfd=3, ii=12, amount=50, type=2, tinf=1)
adm <- list(adm1, adm2)
out <- list(name="Cc", time='steady.state', ntp=200)

res <- exposure(model=myModel, parameter=p, output=out, treatment=adm)
```

```
print(res$Cc)
```

```
##   t1 t2     step      auc    tmax     cmax tmin     cmin
## 1 48 72 0.120603 398.9503 51.9799 20.15223   48 12.91594
```

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



Example 3: AUC for multiple outputs and with inter individual variability

---

R script: doc_exposure3.R

We consider in this example a PKPD model, with inter individual variability on the PKPD parameters.

```
library(gridExtra)

myModel <- inlineModel("
[LONGITUDINAL]
input={ka,V,k,Imax,S0,IC50,kout}

EQUATION:
Cc  = pkmodel(ka, V, k)
Ec = Imax*Cc/(Cc+IC50)
PCA_0 = S0
ddt_PCA = kout*((1-Ec)*S0- PCA)

[INDIVIDUAL]
input={ka_pop,omega_ka,V_pop,omega_V,beta_V,k_pop,omega_k,beta_k,
       Imax_pop,omega_Imax,S0_pop,omega_S0,
       IC50_pop,omega_IC50,kout_pop,omega_kout,w,w_pop}

EQUATION:
V_pred = V_pop*(w/w_pop)^beta_V
k_pred = k_pop*(w/w_pop)^beta_k
```

```
DEFINITION:
ka  ={distribution=lognormal, prediction=ka_pop,   sd=omega_ka}
V   ={distribution=lognormal, prediction=V_pred,   sd=omega_V}
k   ={distribution=lognormal, prediction=k_pred,   sd=omega_k}
S0  ={distribution=lognormal, prediction=S0_pop,   sd=omega_S0}
IC50={distribution=lognormal, prediction=IC50_pop, sd=omega_IC50}
kout={distribution=lognormal, prediction=kout_pop, sd=omega_kout}
Imax={distribution=logitnormal, prediction=Imax_pop, sd=omega_Imax}

[COVARIATE]
input={w_pop, omega_w}

DEFINITION:
w={distribution=normal, mean=w_pop, sd=omega_w}
")
```

We provide some values for the population PKPD parameters:

```
p <- c(w_pop=70,     omega_w=10,
       ka_pop=0.5,   omega_ka=0.2,
       V_pop=10,     omega_V=0.1,    beta_V= 1,
       k_pop=0.1,    omega_k=0.15,   beta_k=-0.25,
       Imax_pop=0.8, omega_Imax=0.4,
       S0_pop=100,   omega_S0=0,
       IC50_pop=1,   omega_IC50=0.2,
       kout_pop=0.1, omega_kout=0.1)
```

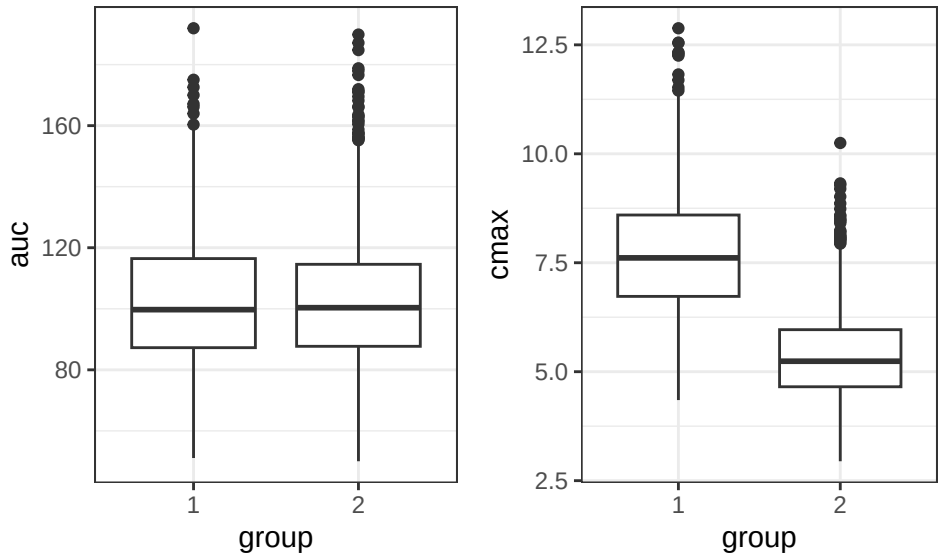
We want to compare the exposure for two different dosage regimens, taking into account the inter individual variability of the parameters. In order to investigate the distributions of the AUCs and Cmax, we simulate two arms with a large number of individuals ($N = 1000$).

Because we are interested in the PK and the PD, both outputs are defined.

```
N <- 1000
adm1 <- list(ii=24, amount=100)
adm2 <- list(ii=12, amount=50)
out <- list(name=c("Cc","PCA"), time='steady.state')

g1 <- list(treatment=adm1, size=N, level='covariate')
g2 <- list(treatment=adm2, size=N, level='covariate')

res <- exposure(model     = myModel,
                parameter = p,
                output    = out,
                group     = list(g1,g2))
```

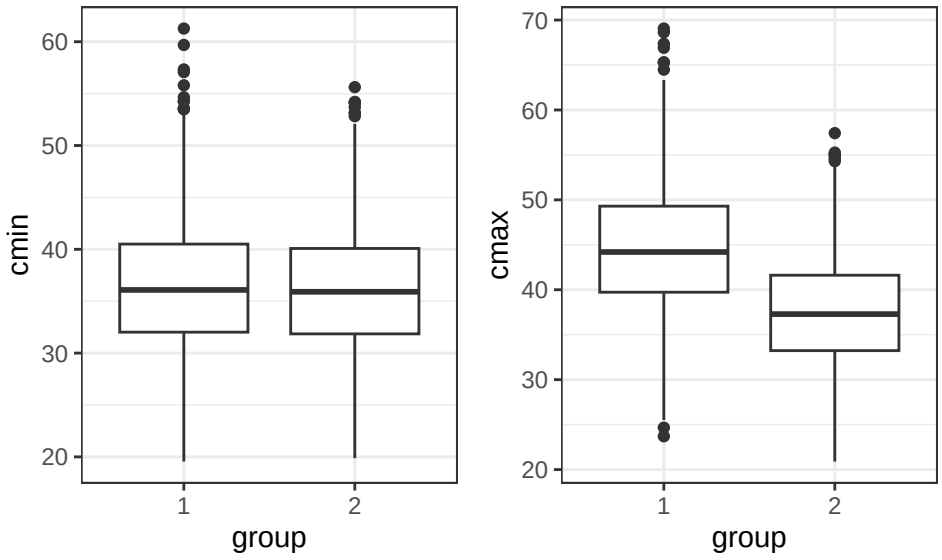
We can display the empirical distribution of the AUC and the Cmax using boxplots:

```
b11 <- ggplot(data=res$Cc)+geom_boxplot(aes(x=group,y=auc))
b12 <- ggplot(data=res$Cc)+geom_boxplot(aes(x=group,y=cmax))
grid.arrange(b11,b12,nrow=1)
```



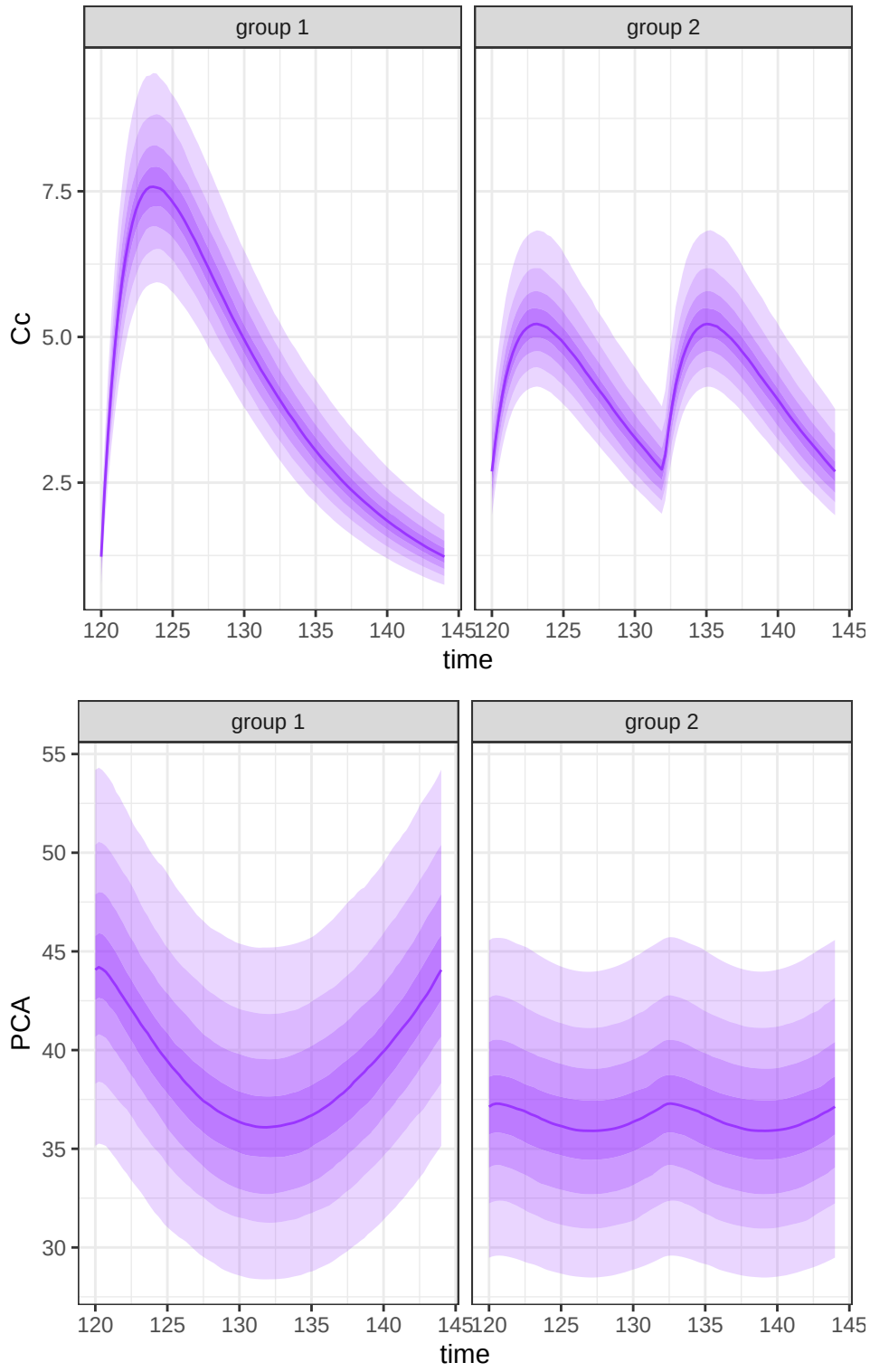
We can also display the empirical distribution of the maximum and minimum PCA values over a 24 hours period:

```
b21 <- ggplot(data=res$PCA)+geom_boxplot(aes(x=group,y=cmin))
b22 <- ggplot(data=res$PCA)+geom_boxplot(aes(x=group,y=cmax))
grid.arrange(b21,b22,nrow=1)
```



Here, `res$output`is a list with two elements: `Cc` and `PCA` computed over a 24 hours period. We can use `prctilemlx` to display the empirical distribution of these two functions of time (the 10 deciles are displayed for each distribution)

```
pl1 <- prctilemlx(res$output$Cc, labels=c("group 1", "group 2"))  +  theme(legend.position="none")
pl2 <- prctilemlx(res$output$PCA, labels=c("group 1", "group 2"))  +  theme(legend.position="none")
grid.arrange(pl1,pl2)
```



Remark: we could have used fixed time intervals instead of steady state:

```
adm1 <- list(time=seq(0, 240, by=24), amount=100)
adm2 <- list(time=seq(0, 252, by=12), amount=50)
out <- list(name=c("Cc","PCA"), time=seq(240,264,length=100))
```

monolix2simulx.R

Overview

Description

Convert a Monolix Project into an executable R script for the simulator `simulx`.

A folder containing the executable R code for `simulx` and the data files used for the simulation (parameters, observation design, dosage regimen, covariates) is automatically created.

Usage

```
monolix2simulx(project, parameter = NULL, group = NULL, open = FALSE, r.data = TRUE, fim = NULL)
```

Arguments

`project`

- the name of a Monolix project

`parameter`

- string, the type of specific parameters to use: “mode” or “mean”

`group`

- a list with the number of subjects

`open`

- load the R script created if open=TRUE

`r.data`

- read the data if r.data=TRUE

`fim`

- Fisher information matrix

Value

A list of paths to the elements used by `simulx` for the simulation

Examples

R script: doc_monolix2simulx.R

Monolix project: monolixRuns

The theophylline example (case studies) is used to illustrate how to directly simulate the model just by providing the Monolix project as an input to `simulx`.

```
project.file <- 'monolixRuns/theophylline_project.mlxtran'
res <- simulx(project = project.file)
```

We can alternatively use `monolix2simulx` to create an independent folder with a `simulx` script for simulation that can easily modified or shared with others.

```
r <- monolix2simulx(project = project.file)
```

A new folder `theophylline_project_simulx` is automatically created in the current working directory. It contains the following files:

- `theophylline_project_simulxModel.txt`: the model Mlxtran
- `individualCovariate.txt`: the individual covariates
- `treatment.txt`: the individual dosage regimens
- `output.txt`: the individual observation times
- `populationParameter.txt`: the estimated population parameters
- `originalId.txt`: the original id's
- `theophylline_project.R`: a R script

In this example, here is the R script `theophylline_project.R` automatically created:

```
# File generated automatically on 2019-06-19 18:11:02

setwd("D:/userGuide_mlxR/theophylline_project_simulx")

# model
model<-"theophylline_project_simulxModel.txt"

# treatment
trt <- read.csv("treatment.txt")

# parameters
originalId<- read.csv('originalId.txt')
populationParameter <- read.vector('populationParameter.txt')
individualCovariate <- read.csv(file='individualCovariate.txt')
list.param <- list(populationParameter,individualCovariate)
# output
name <- "y1"
time <- read.csv("output.txt",header=TRUE, sep=',')
out <- list(name=name,time=time)

# call the simulator
res <- simulx(model=model,treatment=trt,parameter=list.param,output=out)
```

It is then possible to modify this R script as any `simulx` script.

Three other demo examples, with different types of model and data, are available:

- A Monolix project for the warfarin PKPD data. In this example, the individual parameters, estimated by the conditional modes, are used for the simulation of the PKPD data. Replace "mode" by "mean" to use the conditional means instead of the conditional modes.

```
project.file <- 'monolixRuns/warfarin_project.mlxtran'
monolix2simulx(project = project.file, parameter="mode")
```

- A Monolix project for joint continuous and categorical data

```
project.file <- 'monolixRuns/pkcat_project.mlxtran'
monolix2simulx(project = project.file)
```

- A Monolix project for joint continuous and time to event data

```
project.file <- 'monolixRuns/pkrtte_project.mlxtran'
monolix2simulx(project = project.file)
```

shinymlx.R

Overview

Description

Create a Shiny application for longitudinal data model.

Usage

```
shinymlx(model, parameter=NULL, output=NULL, treatment=NULL, data=NULL, style="basic",
         appname="shinymlxApp", title=" ", settings=NULL,)
```

Arguments

`model`

- a Mlxtran or PharmML model used for the simulation.

`parameter`

- a vector, or a list of shiny widgets*

`output`

- a list with fields
  - `name` : a vector of output names,
  - `time` : a vector of times, or a vector (min, max, step).

`treatment`

- a list with fields
  - `tfd` : first time of dose,
  - `amount` : amount,
  - `nd` : number of doses,
  - `ii` : interdose interval,
  - `rate` : rate of infusion,
  - `type` : the type of input,
- Remark: Input argument of `simulx` can also be used, i.e. a list with fields `time`, `amount`, `rate`, `tinf`, `type`, `target`.

`data`

- a datafile to display with the plot

`title`

- the title of the application

`appname`

- the name of the application (and possibly its path)

`style`

- the style of the Shiny app
  - `"basic"` : basic Shiny app with a single side bar (default),
  - `"navbar1"` : navigation bar and tabPanels (including outputs),
  - `"navbar2"` : navigation bar and tabPanels (outputs separated),
  - `"dashboard1"` : Shiny dashboard,

`settings` * a list of settings * `"tabstyle"` : look of the tabs `("tabs","pills")`, * `"select.x"` : display the list of variables available for the x-axis `(TRUE,FALSE)`, * `"select.y"` : display the list of variables available for the y-axis `(TRUE,FALSE)`, * `"select.log"` : log scale option `(TRUE,FALSE)`, * `"select.ref"` : reference curve option `(TRUE,FALSE)`.

Value

A directory tempdir with files ui.R, server.R and model.txt

Examples

Example 1: introductory example

R script: doc_shinymlx1.R

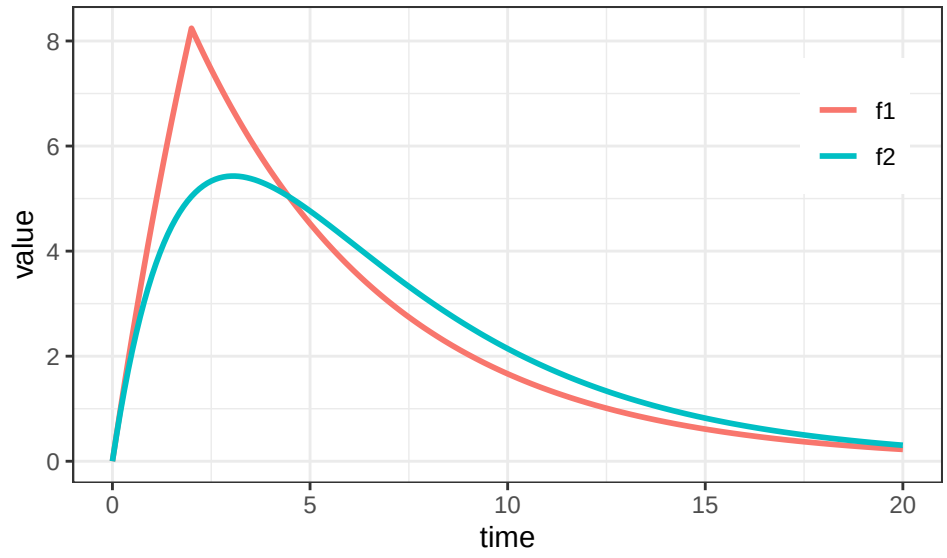
We want to compare zero-order and first-order absorption processes for a single dose administration.

The model is implemented as an inline model:

```
model1 <- inlineModel("
[LONGITUDINAL]
input = {ka, Tk0, V, k}

EQUATION:
D  = 100

if t>Tk0
  f0=D/(V*Tk0*k)*(1-exp(-k*Tk0))*exp(-k*(t-Tk0))
else
  f0=D/(V*Tk0*k)*(1-exp(-k*t))
end

f1 = f0
f2 = D*ka/(V*(ka-k))*(exp(-k*t) - exp(-ka*t))
")
```

Let us start using `simulx` for computing and plotting f_1 and f_2

```
library(reshape2)
f <- list(name = c('f1','f2'), time = seq(0, 20, 0.1))

p <- c(ka=0.5, Tk0=2, V=10, k=0.2)

r <- simulx(model = model1, parameter = p, output = f)

r <- melt(merge(r$f1,r$f2),  id='time', variable.name='f')
ggplot(r, aes(time,value)) + geom_line(aes(colour = f),size=1) +
  guides(colour=guide_legend(title=NULL)) +theme(legend.position=c(.9, .75))
```



We can now run `shinymlx` with the same input arguments. Sliders will be automatically created for the 4 parameters. By default, the minimum and maximum values of a slider are the value of the parameter multiplied, respectively, by 0.5 and 2. The step is the range (max-min) divided by 15.

```
shiny.app <- shinymlx(model = model1, parameter = p, output = f)
shiny::runApp(shiny.app)
```

It is possible to customize the widgets used for each parameters. Available widgets are `"slider"`, `"numeric"` (enter a numerical value) and `"select"` (select between possible choices). No widget will be created by defining `widget="none"`.

In the example below, A slider is created for ka, a numeric input control for $Tk0$ and a select list input control for k. $V = 10$ is constant.

By default, `style = "basic"` is used to create a basic application. Other styles are available, including "dashboard1" to create a Shiny dashboard. You can then, for instance, tick the checkbox `Reference` to display a reference curve (i.e. which is not modified when you modify the parameters values) or reset this reference curve. You can also select the outputs to be displayed, select which variable use for the x-axis, use a log scale for the y-axis, display the R codes, the model code...

```
p <- list( ka  = list(widget='slider',  value=0.5,min=0.1,max=1,step=0.1),
           Tk0 = list(widget='numeric', value=2),
           V   = list(widget='none',    value=2),
           k   = list(widget='select',  selected=0.2, choices=c(0.05,0.2,0.4))
)
shiny.app <- shinymlx(model=model1, parameter=p, output=f, style="dashboard1",
                      title="Compare PK models")
shiny::runApp(shiny.app)
```

Several display settings can be modified (see the `shinymlx` help for the list of display settings). In the example below, the list of variables available for the x-axis and the log scale option are not displayed.

Using `style = "navbar1"` will create a Shiny application with a navigation bar and 2 tabPanels: the parameters and the outputs.

```
s <- list(select.x=FALSE, select.log=FALSE)
shiny.app <- shinymlx(model=model1, parameter=p, output=f, style="navbar1", settings=s)
shiny::runApp(shiny.app)
```

Using `style = "navbar2"` will create a Shiny application with the widgets and the output options side-by-side.

```
shiny.app <- shinymlx(model=model1, parameter=p, output=f, style="navbar2", settings=s)
shiny::runApp(shiny.app)
```

Example 2: load and display data

R script: doc_shinymlx2.R

---

in this example, we want to display the predicted concentration provided by a model together with some PK data.

Data is stored in the file `data/pkdata1.txt`:

```
time  y
0.5   0.451
1     0.552
2     0.676
4     0.568
6     0.334
8     0.205
12    0.126
16    0.187
20    0.098
24    0.043
```

We can then select this datafile with the input argument `data` and plot the data.

Note that in this example, each parameter is defined as a vector of length 4: then a slider will be automatically created for each of these parameters, using the information in the vector: (value, min, max, step).

```
myModel <- inlineModel("
[LONGITUDINAL]
input = {ka, V, Cl}
EQUATION:
C = pkmodel(ka, V, Cl)
")

f <- list(name = 'C',
          time = c(0, 25, 0.5))

p <- list( ka = c(0.5,0.25,3,0.25),
           V  = c(5,1,20,1),
           Cl = c(1,0.5,3,.1))

shiny.app <- shinymlx(model     = myModel,
                      data      = 'data/pkdata1.txt',
                      treatment = list(time=c(0,12), amount=c(10,3)),
                      parameter = p,
                      output    = f)
shiny::runApp(shiny.app)
```

Example 3: PK model - combine multiple administrations

R script: doc_shinymlx3.R

In this example, we consider a basic one compartment PK model. Here, input argument `treatment` is defined as a combination of oral and iv administrations, as it would be defined for `simulx`.

We may also want to modify the dosage regimen interactively. We thus define the time of first dose (`tfd`), the number of doses (`nd`) and the interdose interval (`ii`). We create sliders for these parameters, for the amount and for the rate of infusion by defining for each a vector (value, minimum, maximum, step). We can also interactively modify the route of administration by creating a list of possible values for `type`: 1=oral, 2=iv.

Two panels are then created for the two types of administration: `adm1` and `adm2`. The parameters of the model can still be modified in the panel `param`

```
myModel <- inlineModel("
[LONGITUDINAL]
input = {F, ka, V, k}

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

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

p    <- c(F=0.7, ka=1, V=10, k=0.1)
Cc   <- list(name="Cc", time=seq(0, 100, by=0.1))

adm1 <- list(
  type   = list(widget="select", selected=1, choices=c(1,2)),
  tfd    = list(widget="slider", value=6,  min=0, max=24, step=2),
  nd     = list(widget="slider", value=3,  min=0, max=10, step=1),
  ii     = list(widget="slider", value=12, min=3, max=24, step=1),
  amount = list(widget="slider", value=40, min=0, max=50, step=5)
)
adm2 <- list(
  type   = list(widget="select", selected=2, choices=c(1,2)),
  tfd    = list(widget="slider", value=12, min=0, max=24, step=2),
  nd     = list(widget="slider", value=2,  min=0, max=10, step=1),
  ii     = list(widget="slider", value=30, min=6, max=60, step=6),
  amount = list(widget="slider", value=20, min=0, max=50, step=5),
  rate   = list(widget="slider", value=5,  min=1, max=10, step=1)
)

shiny.app <- shinymlx(model    = myModel,
                      parameter = p,
                      output    = Cc,
                      treatment = list(adm1,adm2),
                      style     = "navbar1")
shiny::runApp(shiny.app)
```

Example 4: PKPD model - display multiple outputs

R script: doc_shinymlx4.R

```
PKPDmodel <- inlineModel("
[LONGITUDINAL]
input={ka,V,Cl,ke0,Imax,IC50,S0,kout}

EQUATION:
{Cc, Ce}  = pkmodel(ka, V, Cl, ke0)

Ec = Imax*Cc/(Cc+IC50)
E1 = S0*(1 - Ec)

Ee = Imax*Ce/(Ce+IC50)
E2 = S0*(1 - Ee)

E3_0 = S0
ddt_E3 = kout*((1-Ec)*S0- E3)
")

pk.param <- c(ka=0.5, V=10, Cl=1)
pd.param <- c(ke0=0.1, Imax=0.5, IC50=0.03, S0=100, kout=0.1)

adm <- list(tfd=5, nd=15, ii=12, amount=1)

pk <- list(name = 'Cc',                time = seq(0, 250, by=1))
pd <- list(name = c('E1','E2', 'E3'),  time = seq(0, 250, by=1))


shiny.app <- shinymlx(model     = PKPDmodel,
                      treatment = adm,
                      parameter = list(pk.param, pd.param),
                      output    = list(pk, pd),
                      title     = "PKPD models",
                      style     = "dashboard1",
                      appname   = "demo4")
shiny::runApp(shiny.app)
```

mlxplore.R

Overview

Description

Explore and visualize Mlxtran and pharmML models with the Mlxplore software

Usage

```
mlxplore(model, parameter, output, treatment=NULL, group=NULL)
```

Arguments

`model`

- a Mlxtran or PharmML model.

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

`output`

- a list with fields
  - `name` : a vector of output names,
  - `time` : a vector of times.

```
o <- list(name= c('C','E'), time=seq(0,24,by=0.1))
mlxplore(..., output=o, ... )
```

`parameter`

- a vector of parameters with their names and values*

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

`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=∞),
  - `type` : the type of input (default=1),
  - `target` : the target compartment (default=`NULL`).

```
tr <- list(time=c(0,12), amount=50, rate=50)
mlxplore(...,treatment=tr, ... )
```

`group`

- a list of lists with unique field
  - `treatment` : a list.

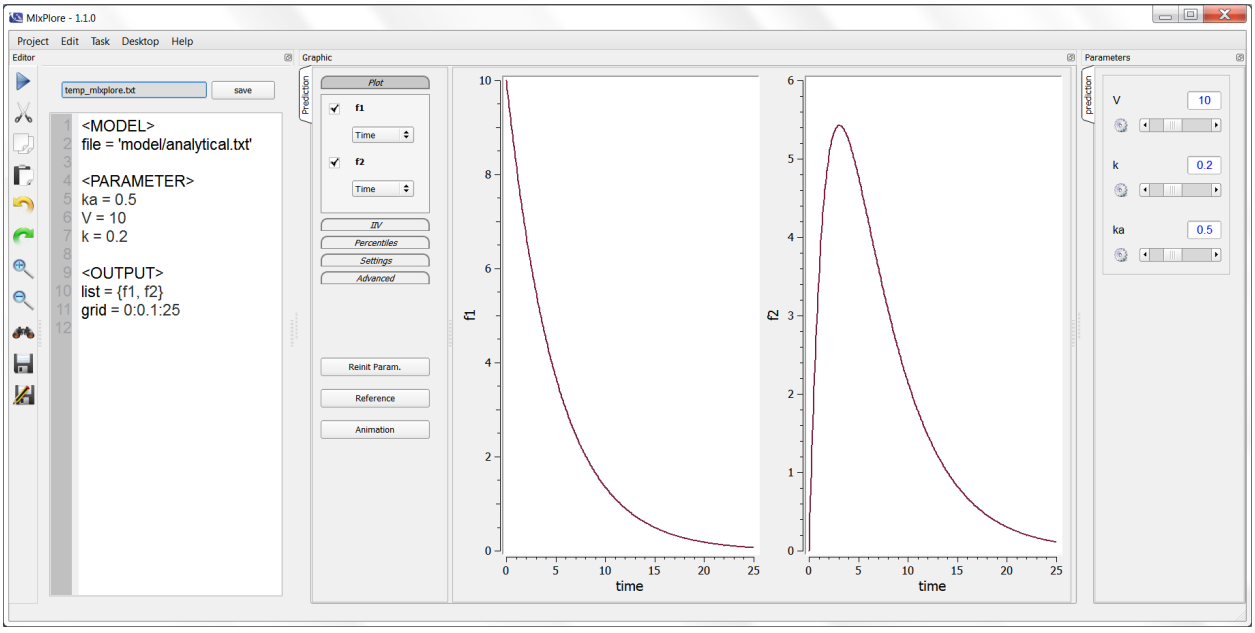
```
g1  <- list(treatment=adm1)
g2  <- list(treatment=adm2)
mlxplore(...,group=list(g1,g2), ... )
```

Examples

Example 1

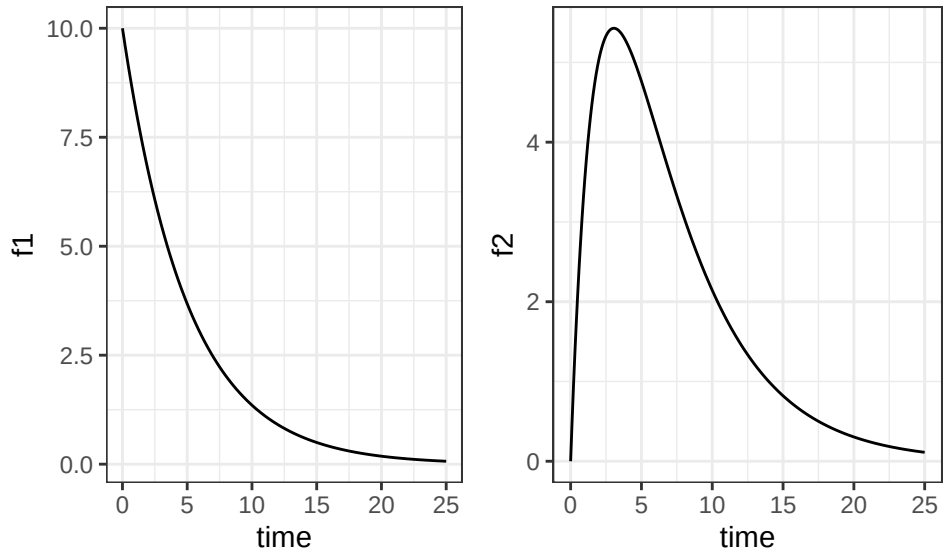
```
[LONGITUDINAL]
input = {ka, V, k}

EQUATION:
D=100
f1 = D/V*exp(-k*t)
f2 = D*ka/(V*(ka-k))*(exp(-k*t) - exp(-ka*t))
```

```
p <- c( ka=0.5, V=10, k=0.2)
f <- list(name = c('f1','f2'), time = seq(0, 25, by=0.1))

mlxplore(model     = 'model/analytical.txt',
         parameter = p,
         output    = f)
```



Remark: the inputs arguments of `mlxplore` are those of `simulx`. When could then run `simulx` with the same inputs arguments:

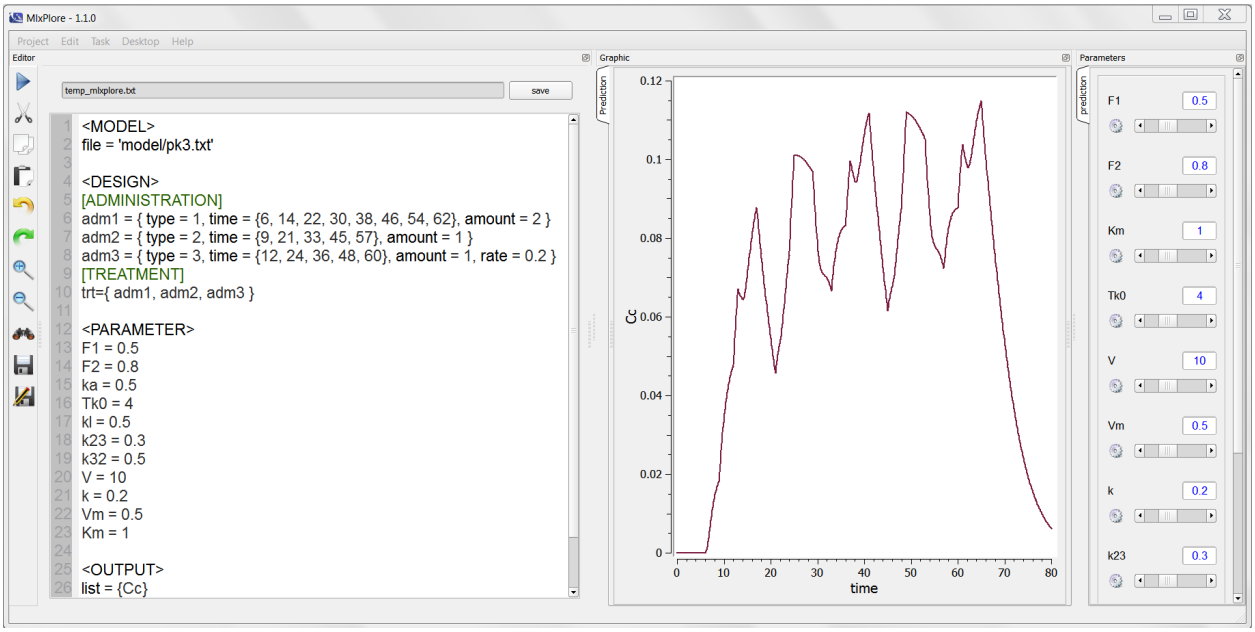
```
library(gridExtra)

res <- simulx(model     = 'model/analytical.txt',
              parameter = p,
              output    = f)
plot1 <- ggplot(data=res$f1) + geom_line(aes(x=time, y=f1))
plot2 <- ggplot(data=res$f2) + geom_line(aes(x=time, y=f2))
grid.arrange(plot1, plot2, ncol=2)
```



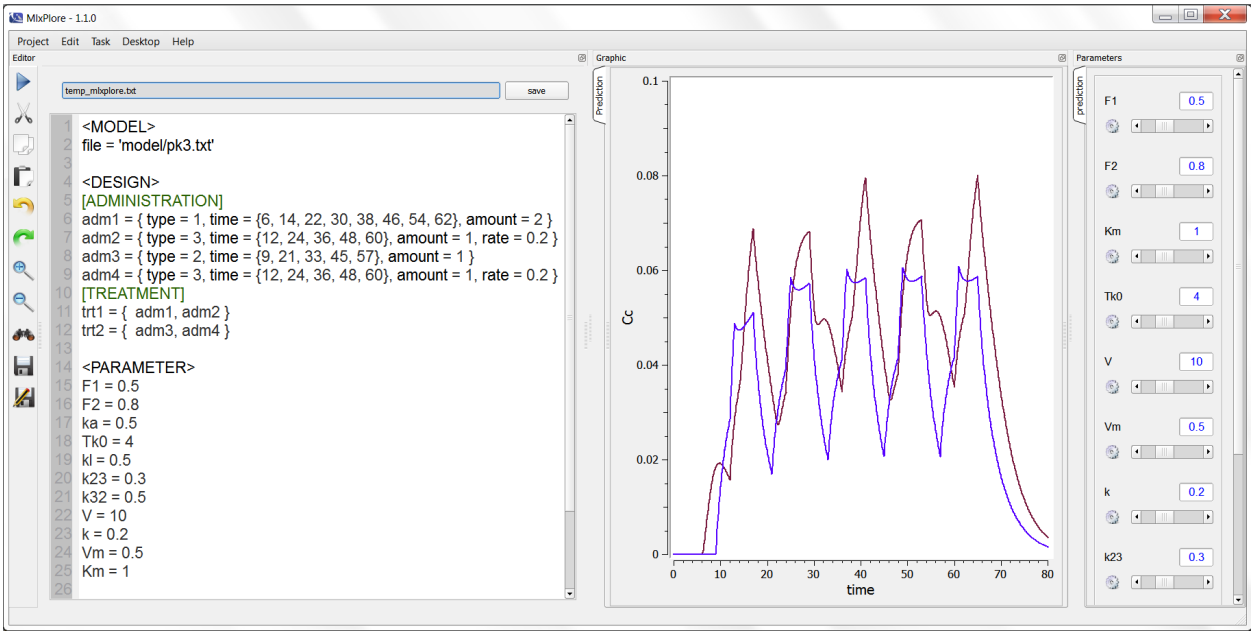
Example 2

```
[LONGITUDINAL]
input = {F1, F2, ka, Tk0, kl, k23, k32, V, k, Vm, Km}

PK:
compartment(cmt=1, amount=Al)
compartment(cmt=2, amount=Ac)
peripheral(k23,k32)
oral(type=1, cmt=1, ka,  p=F1)
oral(type=2, cmt=2, Tk0, p=F2)
iv(type=3, cmt=2)
transfer(from=1, to=2, kt=kl)
elimination(cmt=1, k)
elimination(cmt=2, Km, Vm)
Cc = Ac/V
```

```
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))

mlxplore(model     = "model/pk3.txt",
         parameter = p,
         output    = Cc,
         treatment = list(adm1, adm2, adm3))
```




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

mlxplore(model     = "model/pk3.txt",
         parameter = p,
         output    = Cc,
         group     = list(g1,g2))
```

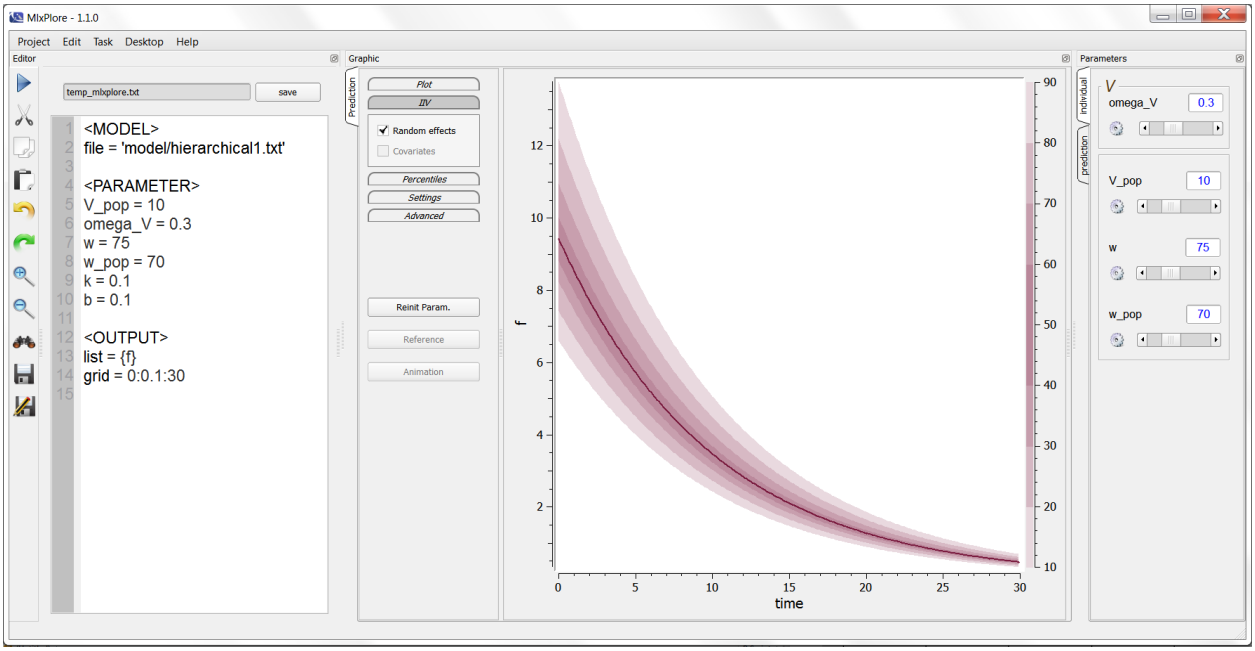


Example 3

```
[LONGITUDINAL]
input = {V, k, b}
EQUATION:
D=100
```

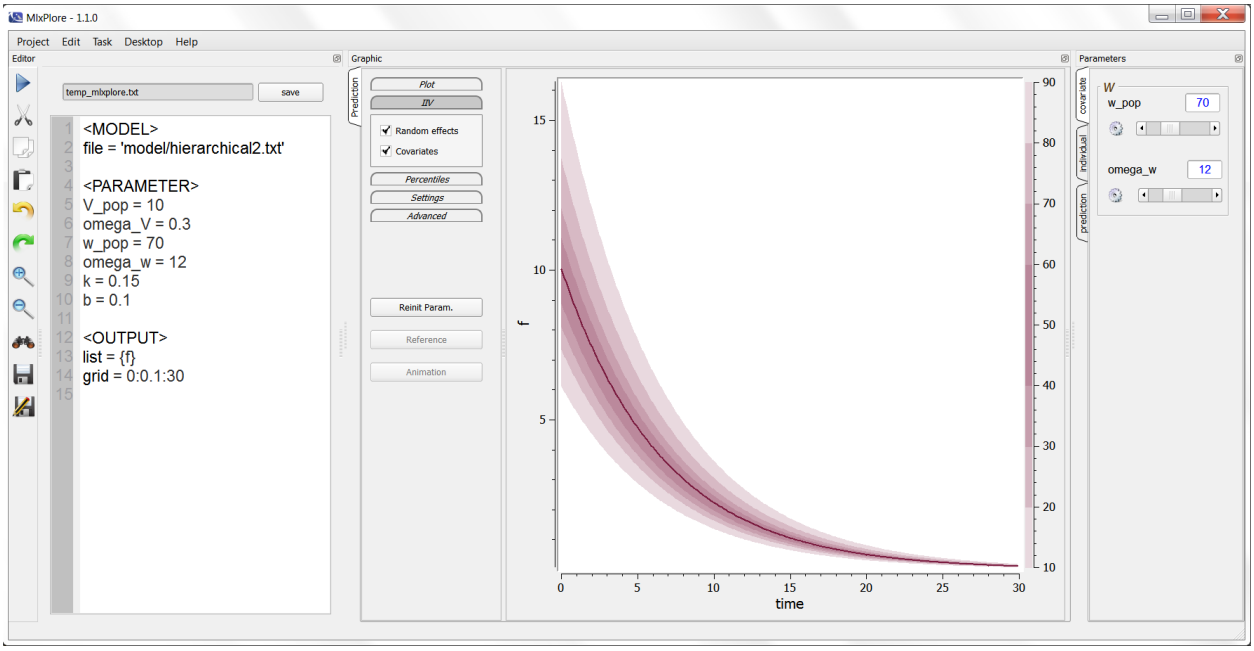
```
f = D/V*exp(-k*t)

[INDIVIDUAL]
input = {V_pop, omega_V, w, w_pop}
EQUATION:
V_pred = V_pop*(w/w_pop)
DEFINITION:
V = {distribution = lognormal, prediction = V_pred, sd = omega_V}
```

```
p <- c(V_pop=10, omega_V=0.3, w=75, w_pop=70, k=0.1, b=0.1)
f <- list(name='f', time=seq(0, 30, by=0.1))

mlxplore(model     = 'model/hierarchical1.txt',
         parameter = p,
         output    = f)
```




```
[LONGITUDINAL]
input = {V, k, b}
EQUATION:
D=100
f = D/V*exp(-k*t)

[INDIVIDUAL]
input = {V_pop, omega_V, w, w_pop}
EQUATION:
V_pred = V_pop*(w/w_pop)
DEFINITION:
V = {distribution = lognormal, prediction = V_pred, sd = omega_V}

[COVARIATE]
input = {w_pop, omega_w}
DEFINITION:
w = {distribution = normal, mean = w_pop, sd = omega_w}
```

```
p <- c(V_pop=10, omega_V=0.3, w_pop=70, omega_w=12, k=0.15, b=0.1)
f <- list(name='f', time=seq(0, 30, by=0.1))

mlxplore(model     = 'model/hierarchical2.txt',
         parameter = p,
         output    = f)
```



prctilemlx.R

Overview

Description

Compute and display percentiles of the empiricial distribution of longitudinal data.

Usage

```
prctilemlx(r,col=NULL, number=8, level=80, plot=TRUE, color="#9a35ff",
           group=NULL, facet=TRUE, labels=NULL, band=NULL)
```

Arguments

`r`

- a data frame with a column `id`, a column `time` and a column with values. The times should be the same for each individual.

`col`

- a vector with the three column indexes for `id`, `time` and `y`. Default = c(1, 2, 3).

`number`

- the number of intervals (i.e. the number of percentiles minus 1).

`level`

- the largest interval (i.e. the difference between the lowest and the highest percentile).

`plot`

- if `TRUE` the empirical distribution is displayed, if `FALSE` values are returned in a data frame.

`color`

- a color to be used for the plots (default="#9a35ff").

`col`

- variable to be used for defining groups (by default, `group` is used when it exists).

`facet`

- makes subplots for different groups if `TRUE`.

`labels`

- vector of strings.

`band`

- deprecated (use number and level instead) ; a list with fields `number` and `level`.

Value

If `plot=TRUE`: a `ggplot` object

If `plot=FALSE`: a list with fields

`proba`

- a vector of probabilities of length `band$number+1`,

color

- a vector of colors used for the plot of length band$number,

y

- a data frame with the values of the empirical percentiles computed at each time point.

Examples

Some basic examples

We simulate longitudinal data for $N = 2000$ individuals using simulx

```
myModel <- inlineModel("
[LONGITUDINAL]
input = {ka, V, Cl}
EQUATION:
C = pkmodel(ka,V,Cl)

[INDIVIDUAL]
input = {ka_pop, V_pop, Cl_pop, omega_ka, omega_V, omega_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, sd=omega_Cl}
")

N=2000

pop.param   <- c(
  ka_pop  = 1,    omega_ka  = 0.5,
  V_pop   = 10,   omega_V   = 0.4,
  Cl_pop  = 1,    omega_Cl  = 0.3)

res <- simulx(model     = myModel,
              parameter = pop.param,
              treatment = list(time=0, amount=100),
              group     = list(size=N, level='individual'),
              output    = list(name='C', time=seq(0,24,by=0.5)))
```

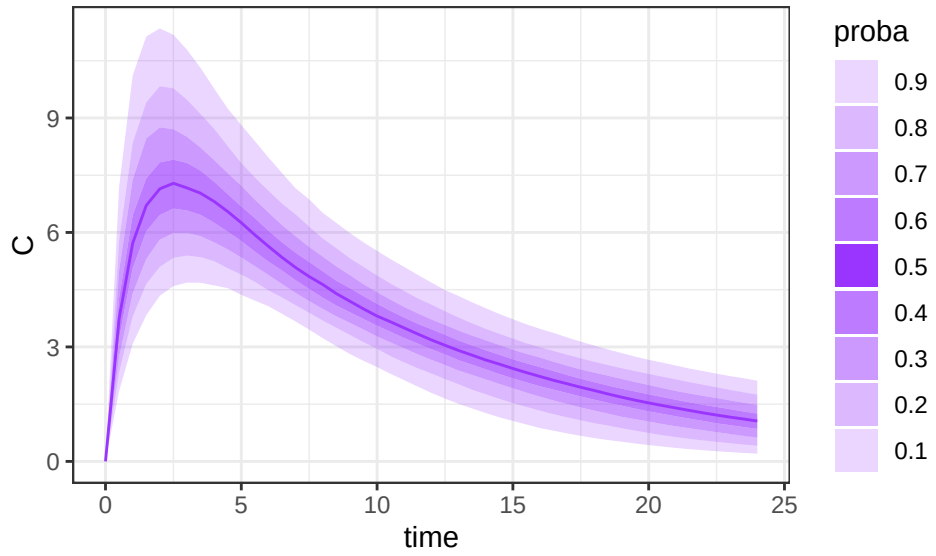
res$C is a dataframe with $2000 \times 241 = 482000$ rows and 3 columns

```
res$C[1:10,]
```

```
##    id time        C
## 1   1  0.0 0.000000
## 2   1  0.5 3.366413
## 3   1  1.0 5.685185
## 4   1  1.5 7.220590
## 5   1  2.0 8.173877
## 6   1  2.5 8.698035
## 7   1  3.0 8.909124
## 8   1  3.5 8.894955
## 9   1  4.0 8.721758
## 10  1  4.5 8.439282
```

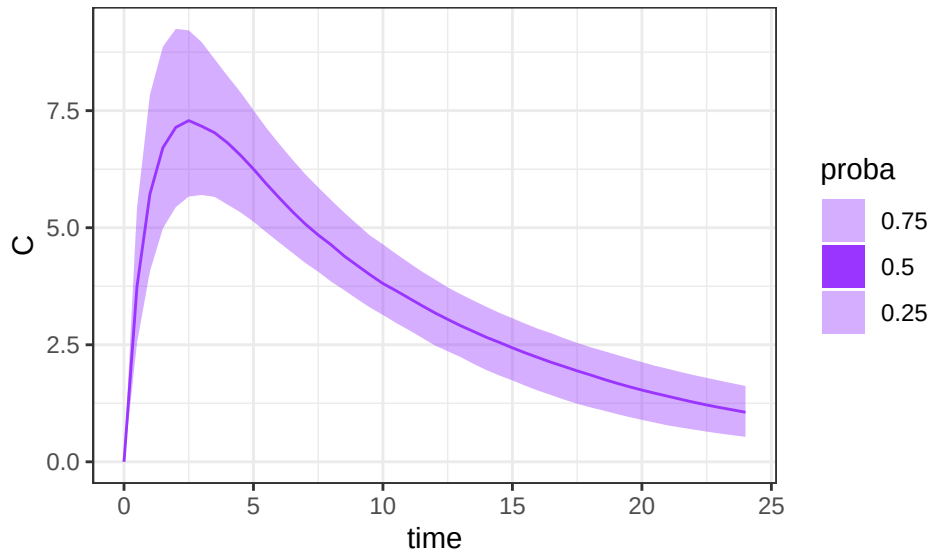
we can compute and display the empirical percentiles of C using the default settings (i.e. percentiles of order 10%, 20%, ... 90%)

```
p1   <- prctilemlx(res$C)
print(p1)
```



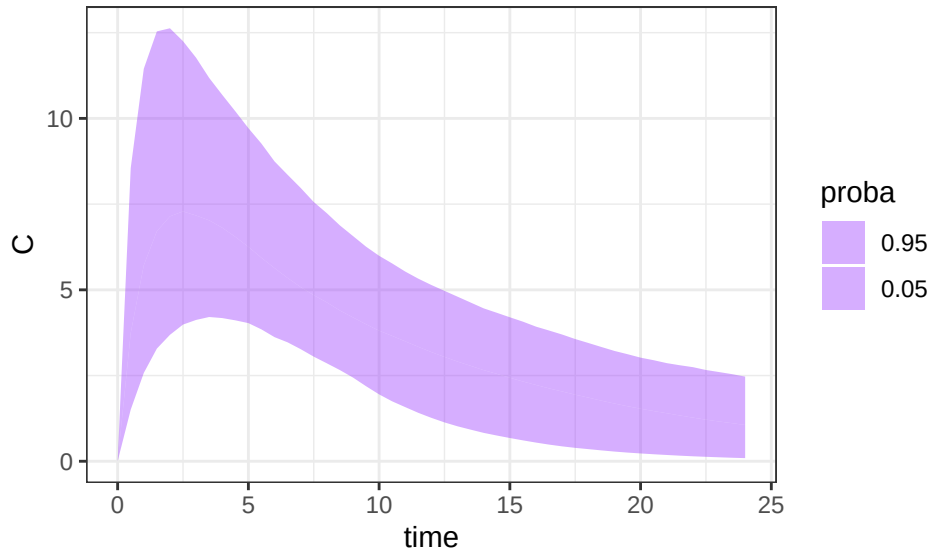
The 3 quartiles (i.e. percentiles of order 25%, 50% and 75%) are displayed by selecting a 50% interval splitted into 2 subintervals

```
p2 <- prctilemlx(res$C, number=2, level=50)
print(p2)
```



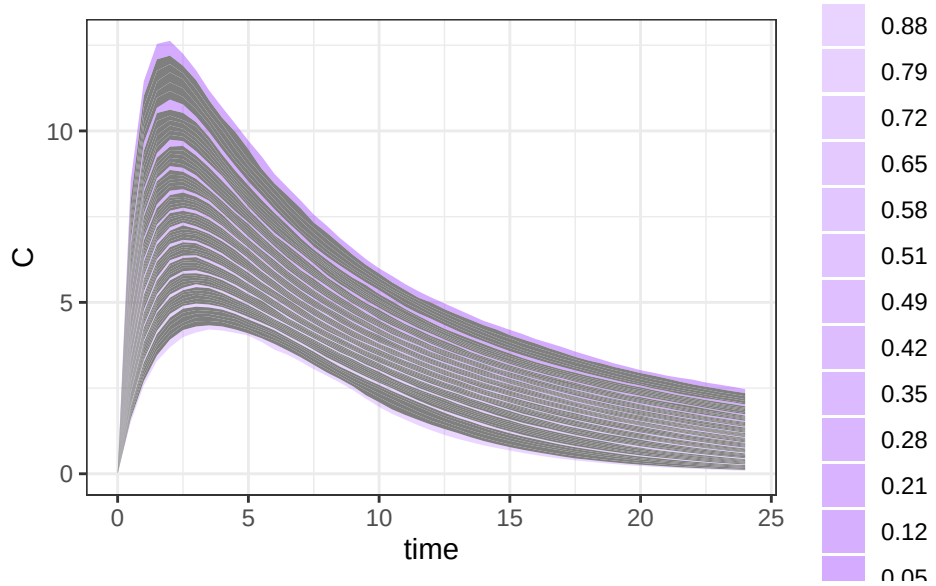
A one 90% interval can be displayed using only one interval

```
p3 <- prctilemlx(res$C, number=1, level=90)
print(p3)
```



or 75 subintervals in order to better represent the continuous distribution of the data within this interval

```
p4 <- prctilemlx(res$C, number=75, level=90)
print(p4)
```



A List of data frames is returned and the percentiles are not plotted by setting `plot=FALSE`

```
p5 <- prctilemlx(res$C, number=4, level=80, plot=FALSE)
print(names(p5))
```

```
## [1] "proba" "color" "y"
```

```
print(p5$proba)
```

```
## [1] 0.1 0.3 0.5 0.5 0.7 0.9
```

```
print(p5$color)
```

```
## [1] "#9A35FF33" "#9A35FF80" "#9A35FFFF" "#9A35FF80" "#9A35FF33"
```

```
print(p5$y[1:5,])
```

```
##   time      10%      30%      50%      50%      70%      90%
## 1  0.0 0.000000 0.000000 0.000000 0.000000 0.000000  0.00000
## 2  0.5 1.846845 2.737048 3.740409 3.740409 5.006423  7.19461
## 3  1.0 3.078062 4.397786 5.717513 5.717513 7.379957 10.11182
## 4  1.5 3.830286 5.302974 6.707726 6.707726 8.462473 11.13611
## 5  2.0 4.340331 5.809902 7.143543 7.143543 8.753283 11.34672
```

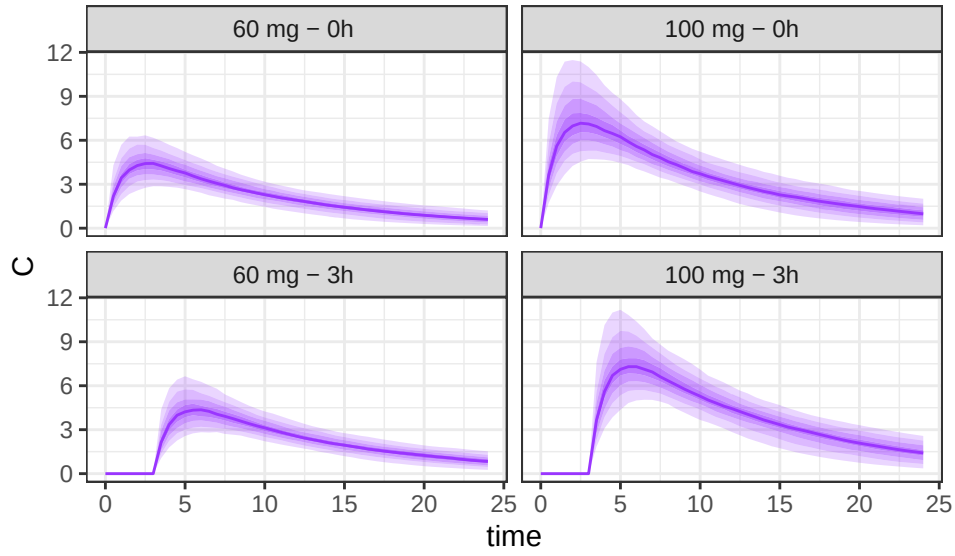
Using prctilemlx with groups

We now define 4 treatment groups:

```
Ng=400
g1 <- list(size=Ng, level='individual', treatment = list(time=0, amount=60))
g2 <- list(size=Ng, level='individual', treatment = list(time=0, amount=100))
g3 <- list(size=Ng, level='individual', treatment = list(time=3, amount=60))
g4 <- list(size=Ng, level='individual', treatment = list(time=3, amount=100))

t.out <- seq(0,24,by=0.5)
res <- simulx(model     = myModel,
              parameter = pop.param,
              group     = list(g1, g2, g3, g4),
              output    = list(name='C', time=t.out))
```

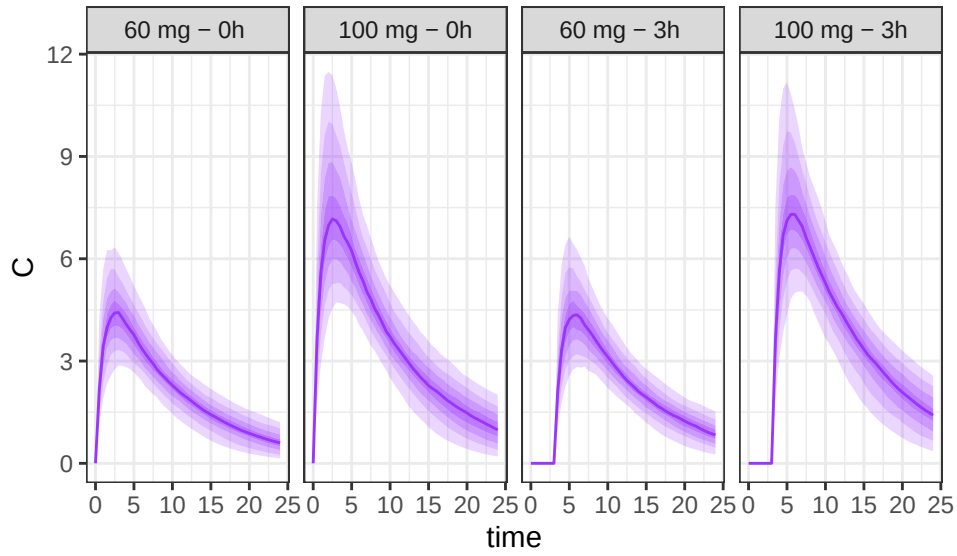
By default, column "group" is used - if it exists - to make subplots

```
labels <- c("60 mg - 0h ", "100 mg - 0h", "60 mg - 3h ", "100 mg - 3h")
resC <- res$C
prctilemlx(resC, label=labels) + theme(legend.position = "none")
```



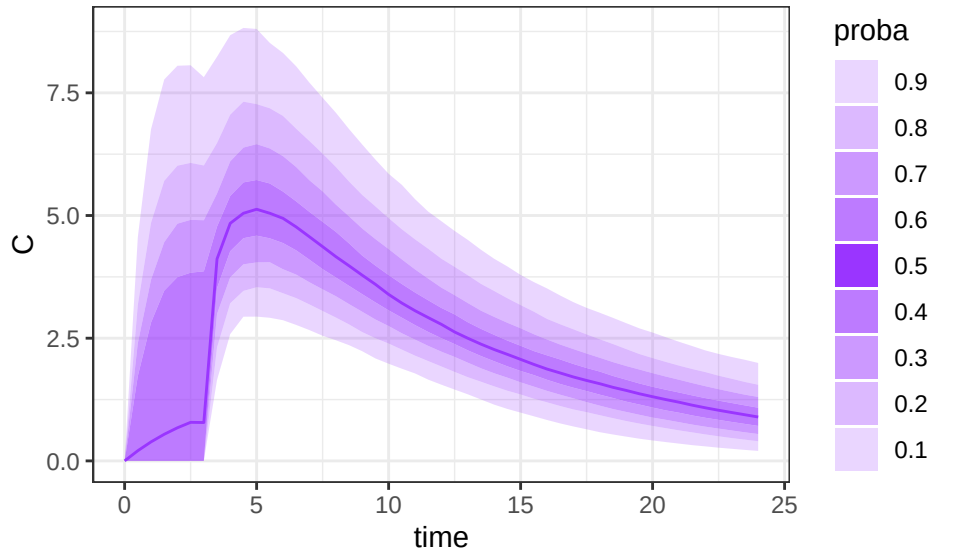
`facet_wrap` is used by `prctilemlx` with the default settings to produce these subplots. These default settings can be modified using `facet_wrap` explicitely

```
prctilemlx(resC, label=labels) + theme(legend.position = "none")  + facet_wrap(~ group, nrow=1)
```



A single plot is produced with group="none" as input argument

```
prctilemlx(resC, group="none")
```

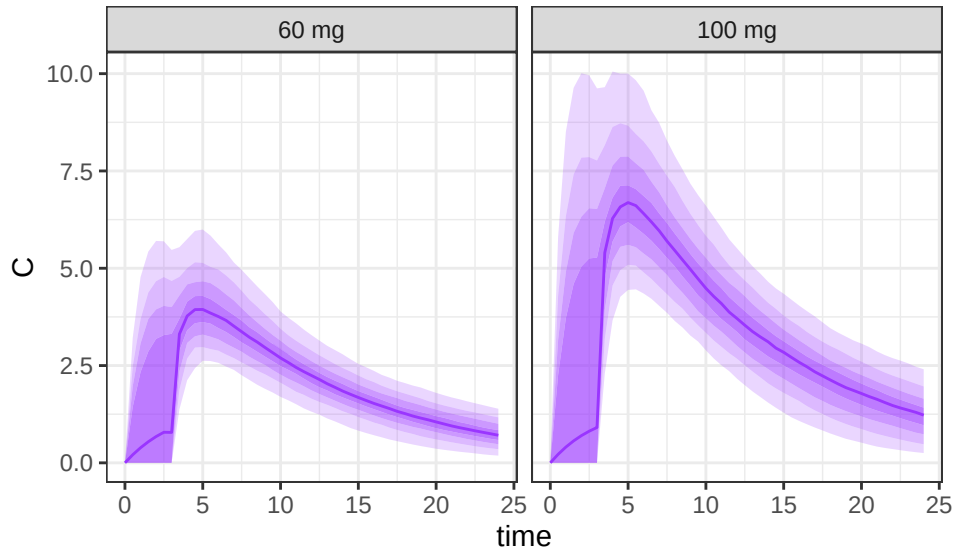


Covariates, such as the time of dose, or the dose amount can be added to the data frame resC and used as categorical covariates to define groups.

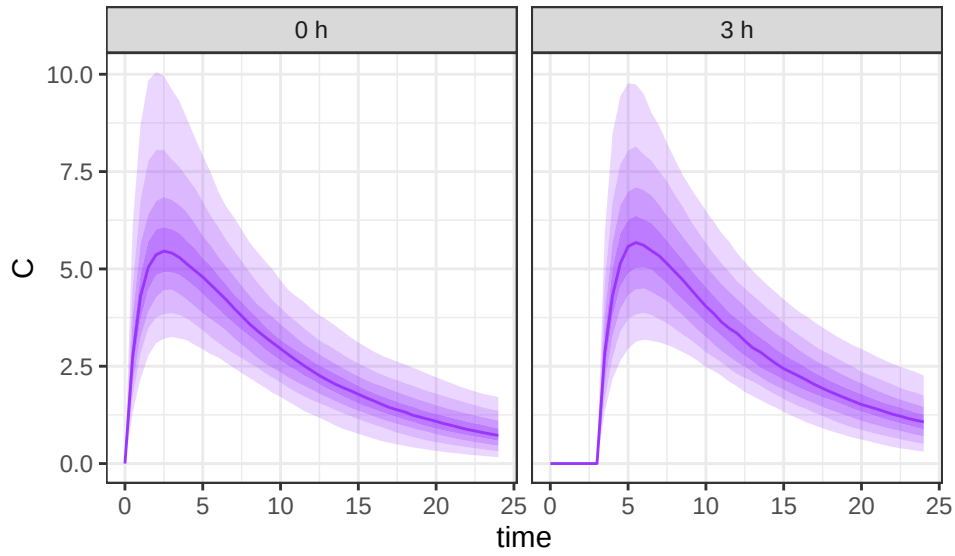
```
resC$doseTime <- 0
resC$doseTime[res$C$group %in% c(3,4)] <- 3
resC$doseAmount <- 60
resC$doseAmount[res$C$group %in% c(2,4)] <- 100
```

These covariates can now be used separately as factors to make subplots

```
prctilemlx(resC, group="doseAmount", labels=c("60 mg", "100 mg")) + theme(legend.position = "none")
```

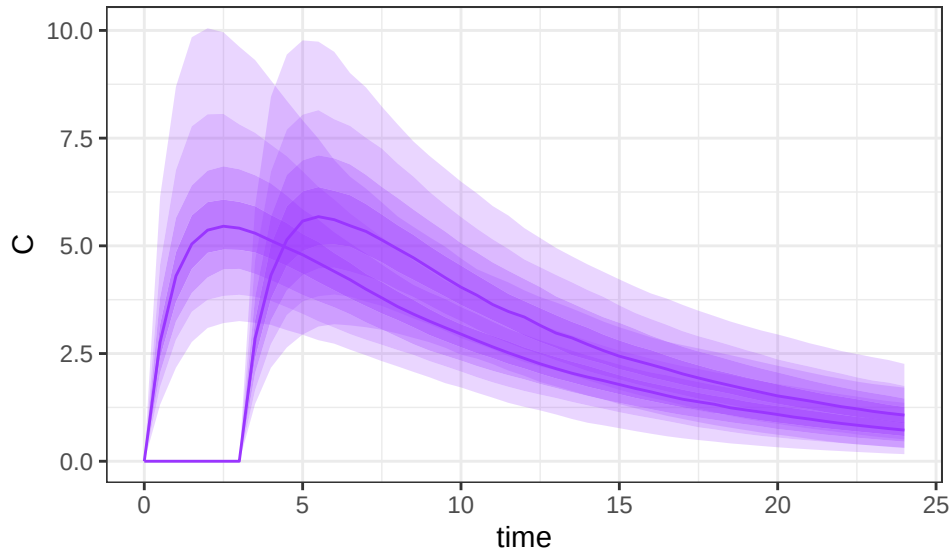



```
prctilemlx(resC, group="doseTime", labels=c("0 h", "3 h"))  +
  theme(legend.position = "none")
```



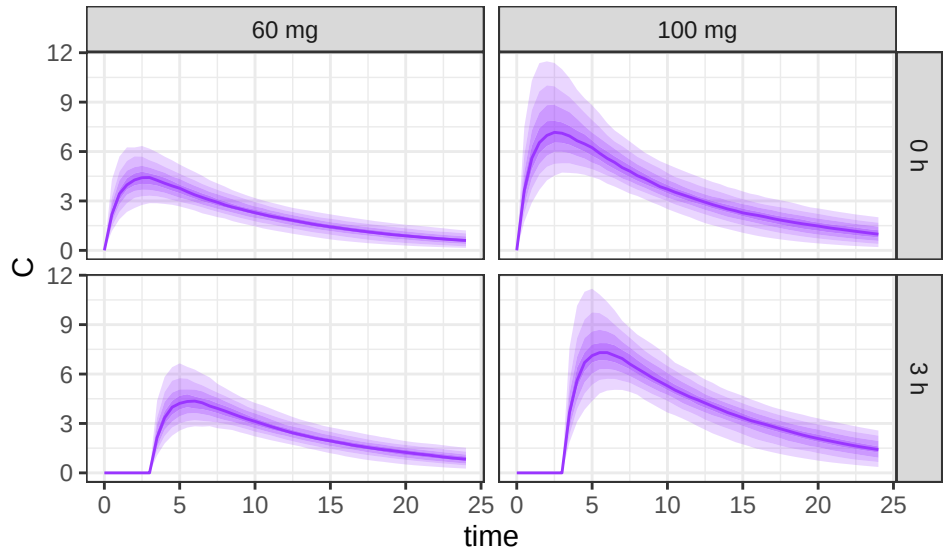
or for plotting the different predictive distributions on a same plot using `facet=FALSE`

```
prctilemlx(resC, group="doseTime", facet=FALSE)  + theme(legend.position = "none")
```



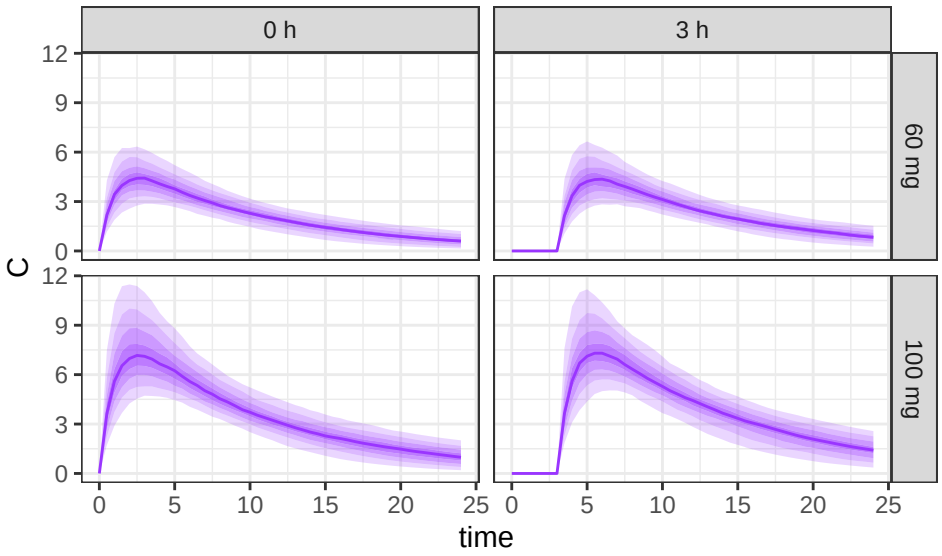
These covariates can also be combined to make a panel of subplots

```
prctilemlx(resC, group=c("doseTime", "doseAmount"),
           labels=list(c("0 h", "3 h"),c("60 mg", "100 mg"))) +
  theme(legend.position = "none")
```



Default settings of `facet_grid` which are used can be modified

```
prctilemlx(resC, group=c("doseTime", "doseAmount"),
           labels=list(c("0 h", "3 h"),c("60 mg", "100 mg"))) +
  facet_grid(doseAmount~doseTime) + theme(legend.position = "none")
```



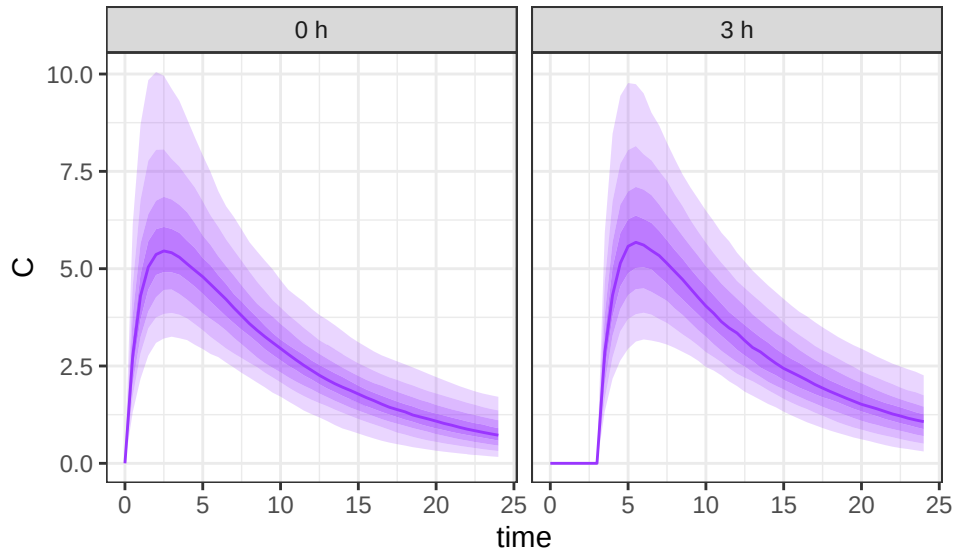
It is also possible to use a data.frame of (categorical) covariates for `group`. This data frame should have a column id and one or two columns of covariates used for making the subplots.

Example with one covariate:

```
N <- nlevels(res$C$id)
dose.time <- rep(c(0,3),each=N/2)
cov <- data.frame(id=levels(res$C$id), doseTime=dose.time)
head(cov)
```

```
##   id doseTime
## 1  1        0
## 2  2        0
## 3  3        0
## 4  4        0
## 5  5        0
## 6  6        0
```

```
prctilemlx(res$C, group=cov[c("id","doseTime")], labels=c("0 h", "3 h")) +
  theme(legend.position = "none")
```

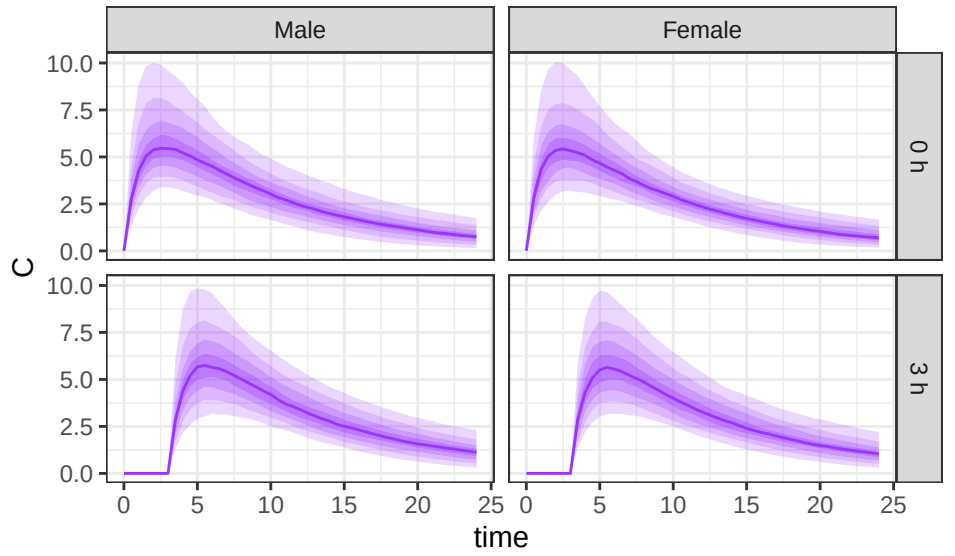


Example with two covariates:

```
gender.sim <- c("M","F")[sample(2,size=N,replace = TRUE)]
cov$gender <- gender.sim
head(cov)
```

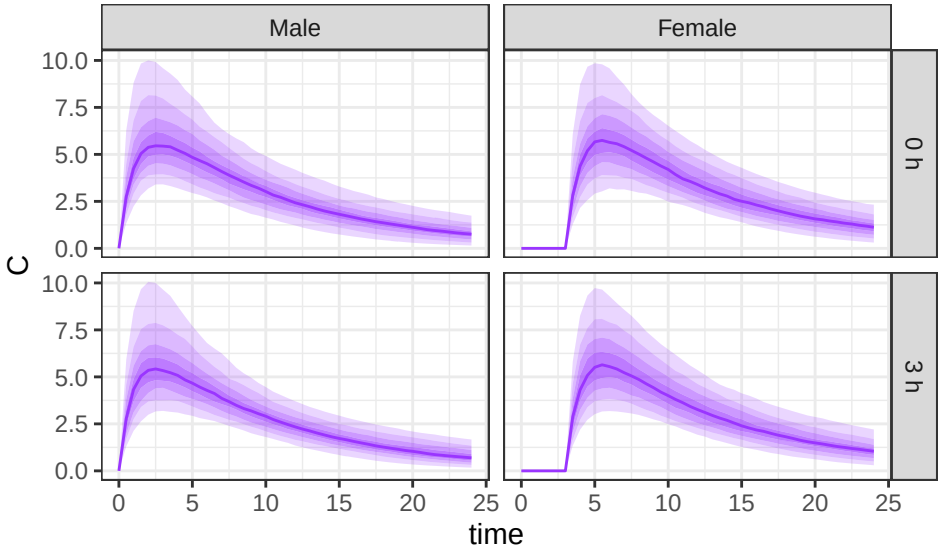
```
##   id doseTime gender
## 1  1        0      M
## 2  2        0      F
## 3  3        0      M
## 4  4        0      M
## 5  5        0      M
## 6  6        0      M
```

```
prctilemlx(res$C, group=cov, labels=list(c("0 h", "3 h"),c("Male", "Female"))) +
  theme(legend.position = "none")
```



Remark: this is equivalent to add a column `gender` and a column `doseTime` to the data frame `resC`.

```
resC$gender <- rep(cov$gender,each=length(t.out))
prctilemlx(resC, group= c("gender","doseTime"), labels=list(c("0 h", "3 h"),c("Male", "Female")))  + the
```



kmplotmlx.R

Overview

Description

Plot empirical survival functions using the Kaplan-Meier estimator.

Usage

```
kmplotmlx(r, index=1, level=NULL, time=NULL, cens=TRUE, plot=TRUE,
          color="#e05969", group=NULL, facet=TRUE, labels=NULL)
```

Arguments

`r`

- a data frame with a column `id`, a column `time`, a column with values and possibly a column `group`.

`index`

- an integer: `index=k` means that the survival function for the `k`-th event is displayed. Default is `index=1`.

`level`

- a number between 0 and 1: confidence interval.

`time`

- a vector of time points where the survival function is evaluated.

`cens`

- display censoring times `TRUE/FALSE` (default=`TRUE`).

`plot`

- plot the estimated survival function `TRUE/FALSE` (default=`TRUE`)

`color`

- color to be used for the plots (default="#e05969").

`group`

- variable to be used for defining groups (by default, `group` is used when it exists).

`facet`

- makes subplots for different groups `TRUE/FALSE` (default=`TRUE`)

`labels`

- vector of strings.

Value

a `ggplot` object if `plot=TRUE`. Otherwise, a list with fields:

`surv`

- a data frame with columns `T` (time), `S` (survival), possibly `(S1, S2)` (confidence interval) and possibly `group`,

`cens`

- a data frame with columns `T0` (time), `S0` (survival) and possibly `group`.

Examples

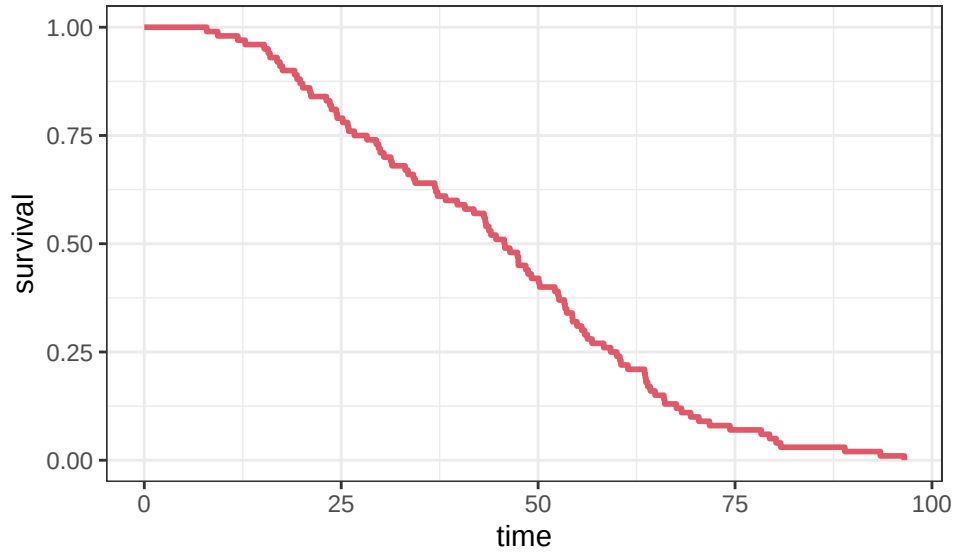
Single event

Survival function\ We simulate a single event for $N = 100$ individuals using a Weibull model. Here, the time to event is exactly known.

```
tteModelA <- inlineModel("
[LONGITUDINAL]
input = {beta,lambda}
EQUATION:
h=(beta/lambda)*(t/lambda)^(beta-1)
DEFINITION:
e = {type=event, maxEventNumber=1, hazard=h}
")

p1   <- c(beta=2.5,lambda=50)
e    <- list(name='e', time=0)
resA1 <- simulx(model     = tteModelA,
                parameter = p1,
                output    = e,
                group     = list(size=100))
```

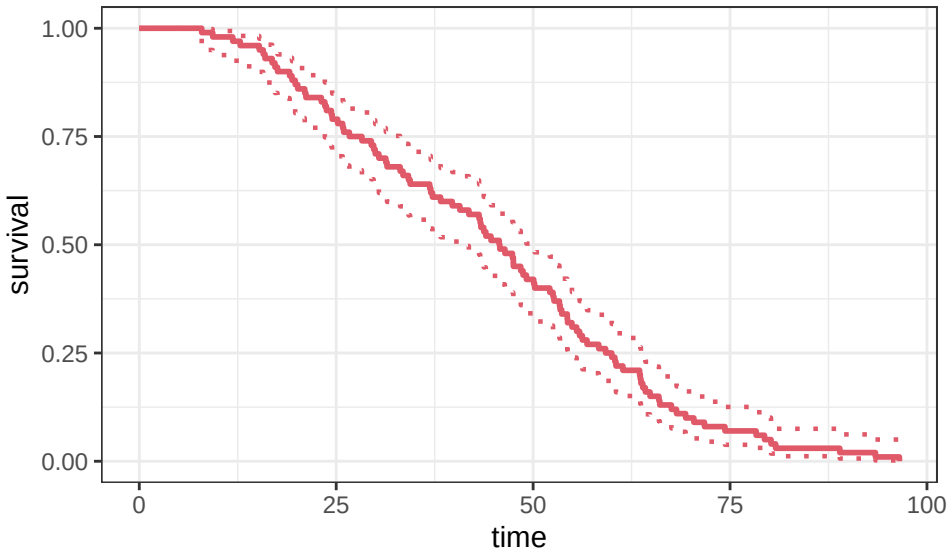
Compute and display the empirical estimate of the survival function:

```
plA1a  <- kmplotmlx(resA1$e)
print(plA1a)
```



Confidence interval\ Display a 90% confidence interval:

```
plA1b  <- kmplotmlx(resA1$e, level=0.90)
print(plA1b)
```



Using groups and replicates

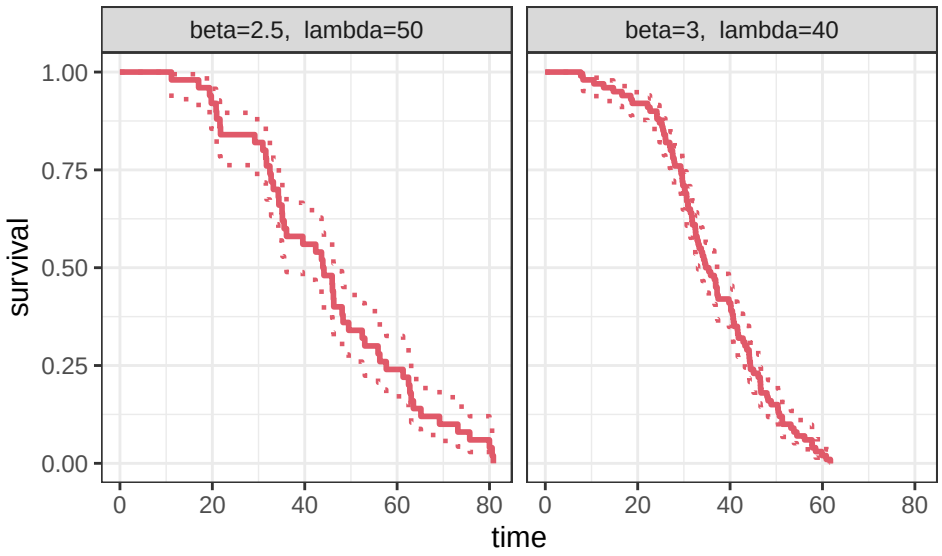
Simulate events for two groups of individuals, using different parameters:

```
p2   <- c(beta=3,lambda=40)
g1   <- list(size=50,  parameter=p1)
g2   <- list(size=100, parameter=p2)
resA2 <- simulx(model    = tteModelA,
                output   = e,
                group    = list(g1,g2),
                settings = list(seed=1234))
```

Here, `res2$e` is a data frame with an additional column `group`.

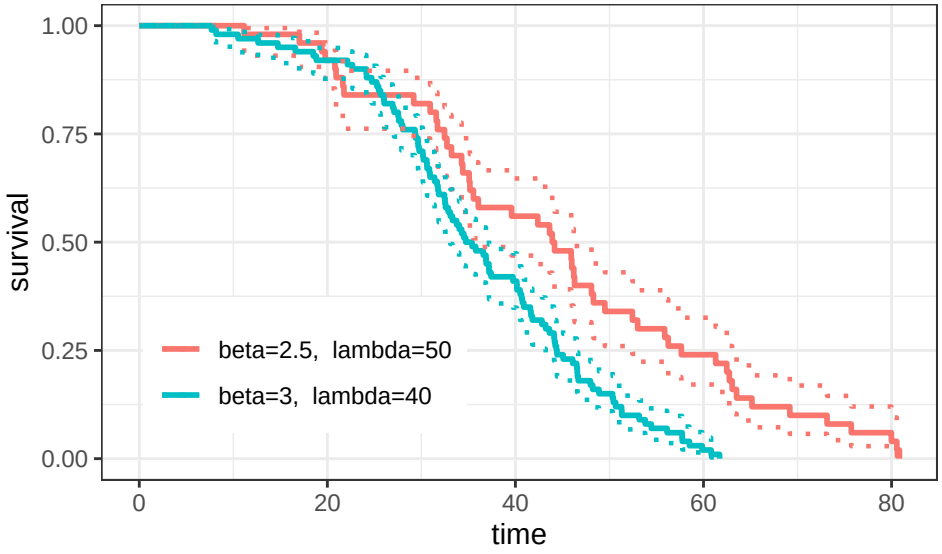
By default, `kmplotmlx` displays the two survival functions on 2 separated subplots:

```
group.labels <- c("beta=2.5,  lambda=50", "beta=3,  lambda=40")
plA2a  <- kmplotmlx(resA2$e, level=0.80, labels=group.labels)
print(plA2a)
```



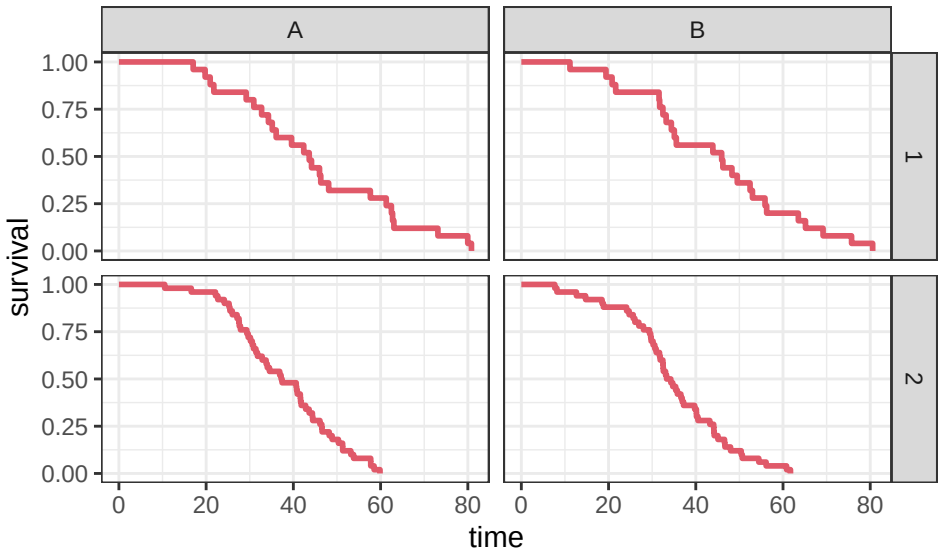
Use `facet=FALSE` if you want a unique panel.

```
plA2b  <- kmplotmlx(resA2$e, level=0.80, facet=FALSE, labels=group.labels) +
           theme(legend.title=element_blank())
print(plA2b)
```



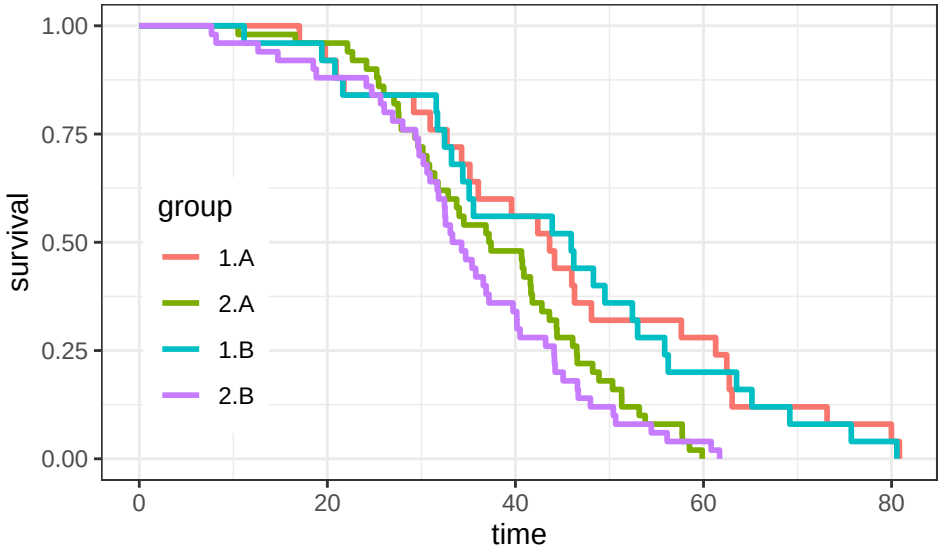
If `group` is defined by two factors, `kmplotmlx` will automatically layout panels in a grid (using `facet_grid`)

```
cov <- data.frame(id=(1:150), cov1=c(rep(1,50),rep(2,100)), cov2=rep(c("A","B"),75))
plA2c  <- kmplotmlx(resA2$e, group=cov)
print(plA2c)
```



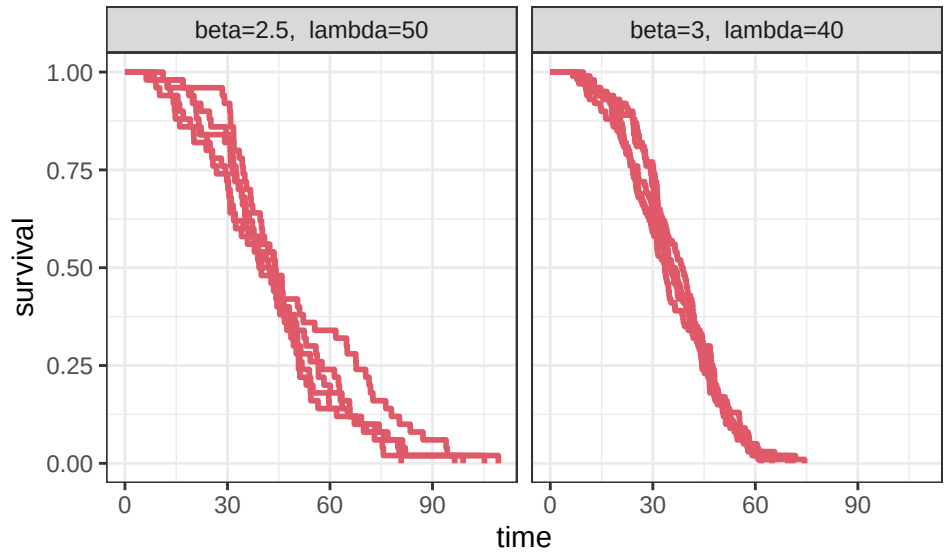
Use `facet=FALSE` if you want a unique panel.

```
plA2d  <- kmplotmlx(resA2$e, group=cov, facet=FALSE)
print(plA2d)
```

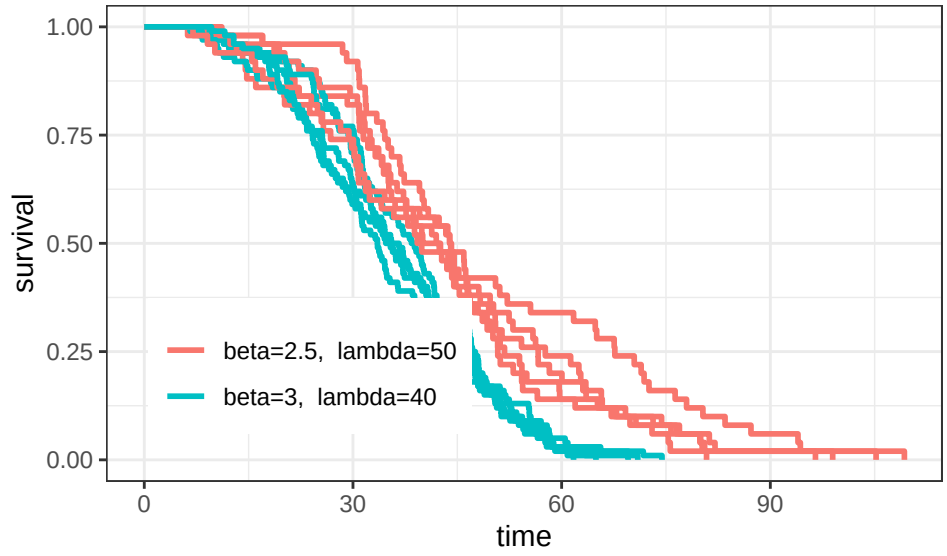


Replicates of the same experiment are displayed on the same subplots.

```
resA3 <- simulx(model    = tteModelA,
                output   = e,
                group    = list(g1,g2),
                nrep     = 5,
                settings = list(seed=1234))
plA3a <- kmplotmlx(resA3$e, labels=group.labels)
plot(plA3a)
```




```
plA3b <- kmplotmlx(resA3$e, facet=FALSE, labels=group.labels) +
           theme(legend.title=element_blank())
plot(plA3b)
```



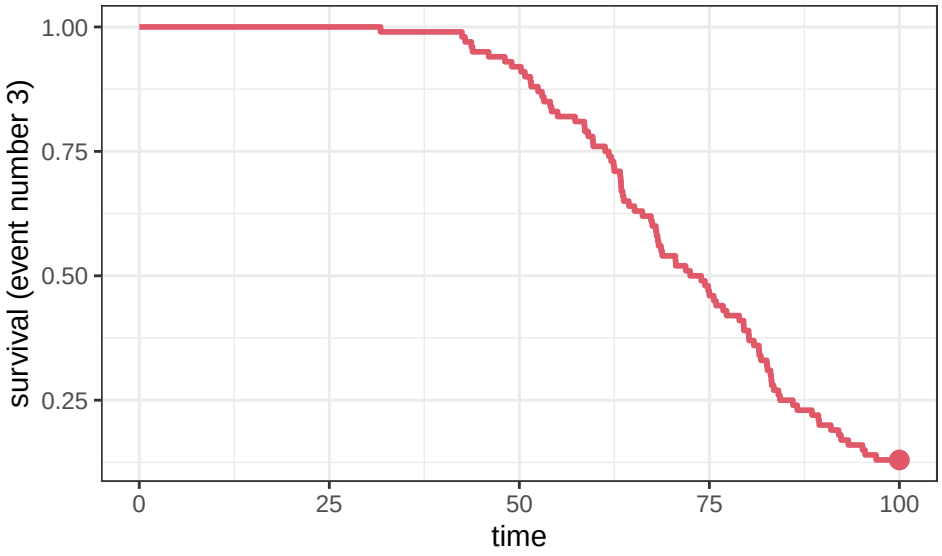
Repeated events

```
tteModelB <- inlineModel("
[LONGITUDINAL]
input = {beta,lambda}
EQUATION:
h=(beta/lambda)*(t/lambda)^(beta-1)
DEFINITION:
e = {type=event, rightCensoringTime=100, hazard=h}
")

resB1 <- simulx(model     = tteModelB,
```

```
                parameter = p1,
                output    = e,
                group     = list(size=100))
```

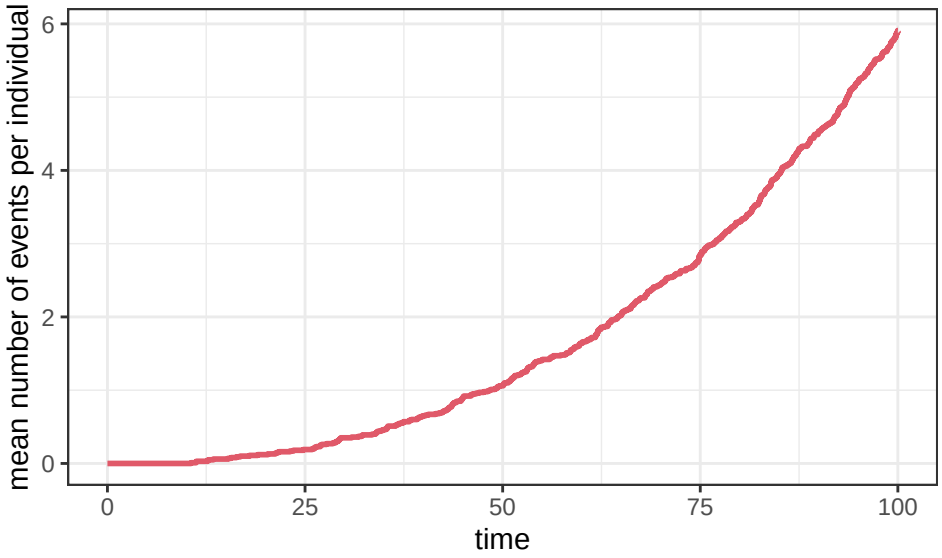
Kaplan-Meier plot for the third event using `index=3`

```
plB1a <- kmplotmlx(resB1$e, index=3)
print(plB1a)
```



Plot the mean number of events per individual using `index="numberEvent"`

```
plB1b <- kmplotmlx(resB1$e, index="numberEvent")
print(plB1b)
```

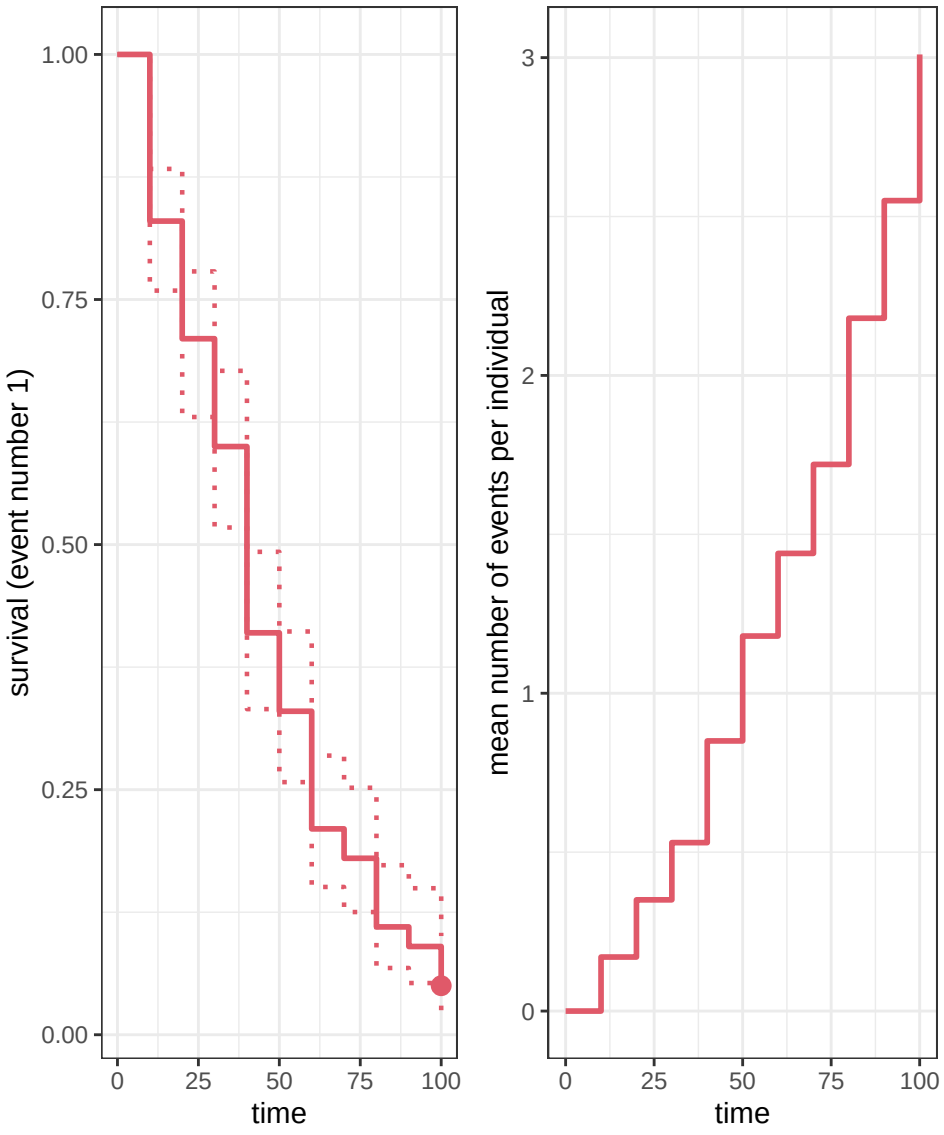


Interval censored events

Let us simulate (repeated) interval censored events

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

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

DEFINITION:
  e = {type=event, eventType=intervalCensored,
       intervalLength=10, rightCensoringTime=100,
       hazard=h}
")

p <- c(beta=1.5,lambda=50)
e <- list(name='e', time=0)
g <- list(size=100)
resC <- simulx(model     = tteModelC,
               parameter = p,
               output    = e,
               group     = g)
```

```
library(gridExtra)
plC1 <- kmplotmlx(resC$e,level=0.90)
plC2 <- kmplotmlx(resC$e,index="numberEvent")
grid.arrange(plC1, plC2, nrow=1)
```


survival (event number 1)
1.00
0.75
0.50
0.25
0.00
0
25
50
75
100
time
mean number of events per individual
3
2
1
0
0
25
50
75
100
time

catplotmlx.R

Overview

Description

Plot the empirical distribution of categorical longitudinal data.

Usage

```
catplotmlx(r, breaks = NULL)
```

Arguments

`r`

- a data frame with a column `id`, a column `time`, a column with values and possibly a column `group`.

`breaks`

- one of:
  - a vector giving the breakpoints,
  - a single number giving the number of segments.

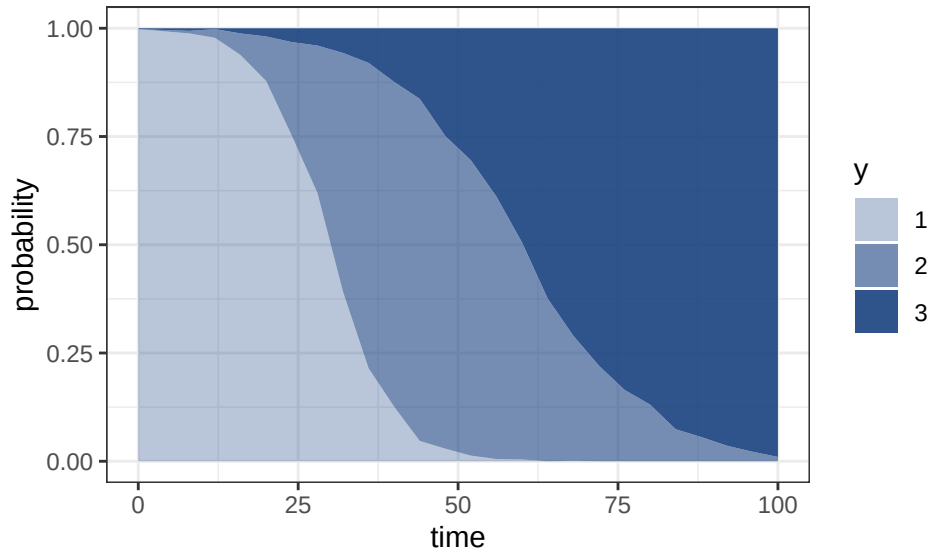
Value

a `ggplot` object.

Examples

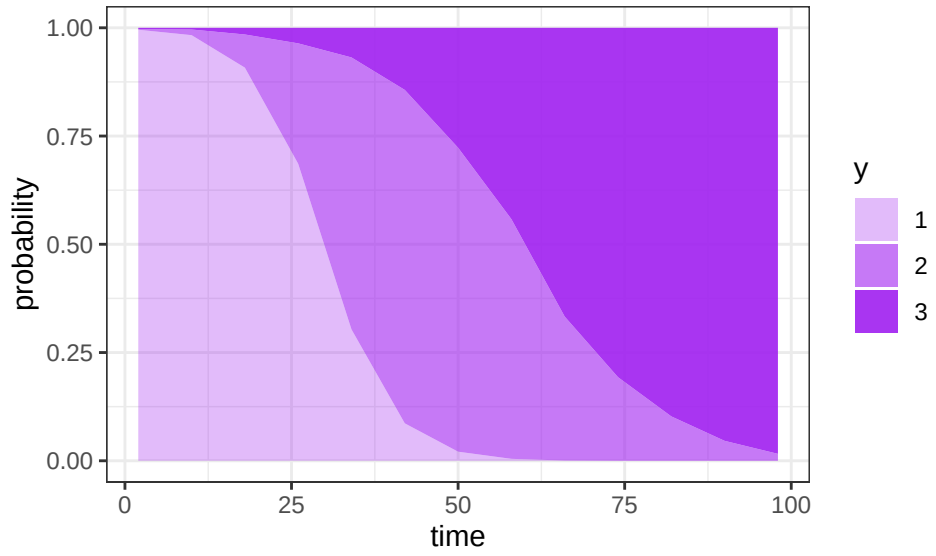
Basic examples

```
catModel <- inlineModel("
[LONGITUDINAL]
input =  {a,b}
EQUATION:
lp1=a-b*t
lp2=a-b*t/2

p1 = 1/(1+exp(-lp1))
p2 = 1/(1+exp(-lp2)) - p1
DEFINITION:
y = {type=categorical, categories={1, 2, 3},
     P(y=1)=p1, P(y=2)=p2}
")

out  <- list(name='y', time=seq(0, 100, by=4))

Ng  <- 1000
g1 <- list(size=Ng, parameter=c(a=6,b=0.2))
res <- simulx(model=catModel, output=out, group=g1)
plot1 <- catplotmlx(res$y)
print(plot1)
```



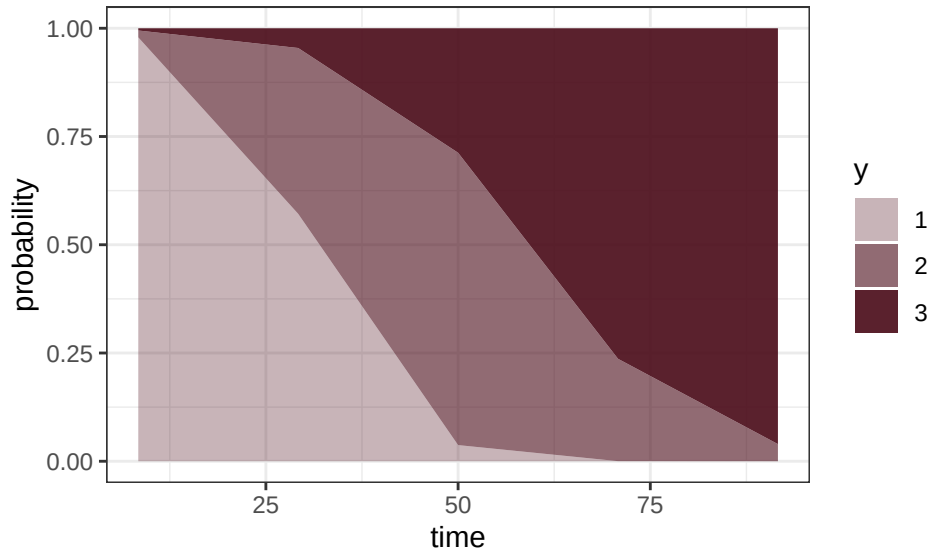
You can modify the location of the breaks and/or the color

```
plot2 <- catplotmlx(res$y, breaks=seq(-2,102,by=8), color="purple")
print(plot2)
```



or define the number of breaks instead of the location,

```
plot3 <- catplotmlx(res$y,breaks=5, color="#490917")
print(plot3)
```



`catplotmlx` returns a list of tables with `plot=FALSE`

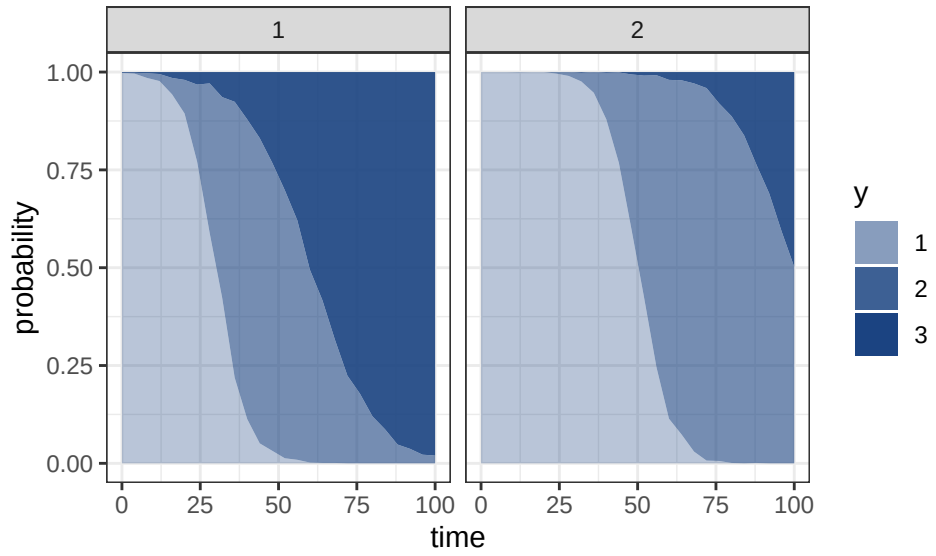
```
cat.out <- catplotmlx(res$y, color="#490917", breaks=5, plot=FALSE)
print(cat.out)
```

```
## $color
## [1] "#4909174D" "#49091799" "#490917E6"
##
## $y
##   time baseline          1      2 3
## 1  8.4        0 0.97900000 0.9948 1
## 2 29.2        0 0.57140000 0.9544 1
## 3 50.0        0 0.03733333 0.7125 1
## 4 70.8        0 0.00020000 0.2366 1
## 5 91.6        0 0.00000000 0.0396 1
```

Using catplotmlx with groups

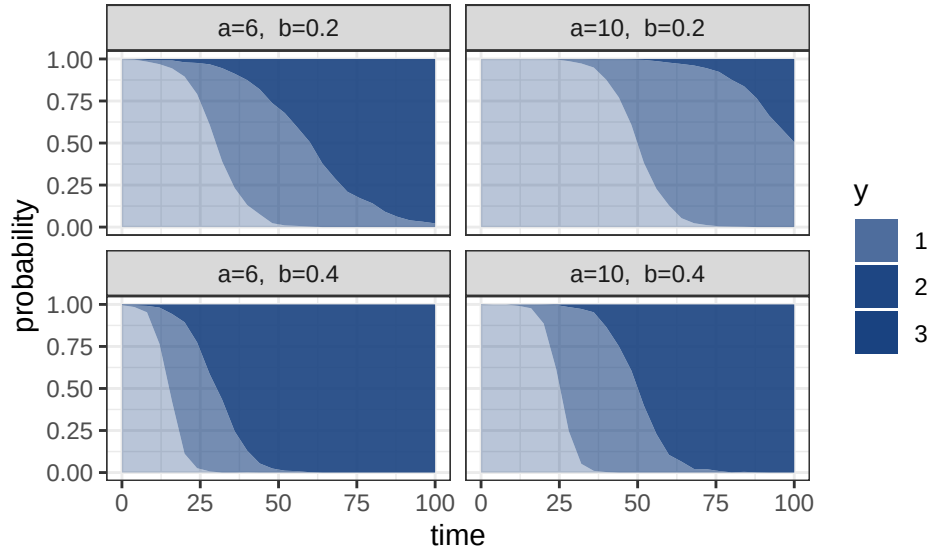
If a column `group` exists in the data, then `catplotmlx` will creat a graph per group.

```
g2 <- list(size=Ng, parameter=c(a=10,b=0.2))
res <- simulx(model=catModel, output=out, group=list(g1,g2))
plot4 <- catplotmlx(res$y)
print(plot4)
```



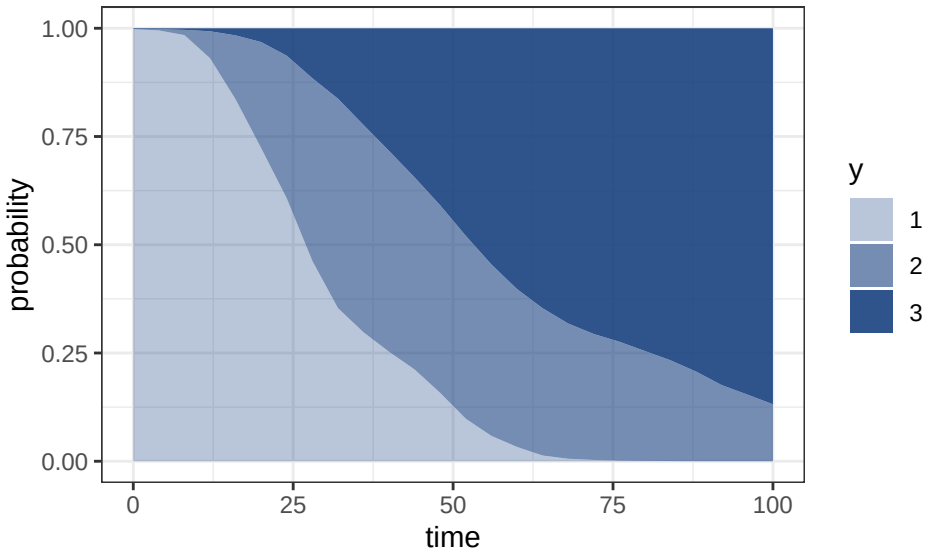
The labels of the subplots can be defined by the user.

```
g3 <- list(size=Ng, parameter=c(a=6,b=0.4))
g4 <- list(size=Ng, parameter=c(a=10,b=0.4))
res <- simulx(model=catModel, output=out, group=list(g1,g2,g3,g4))
group.labels <- c("a=6,  b=0.2", "a=10,  b=0.2", "a=6,  b=0.4", "a=10,  b=0.4")
plot5 <- catplotmlx(res$y, labels=group.labels)
print(plot5)
```



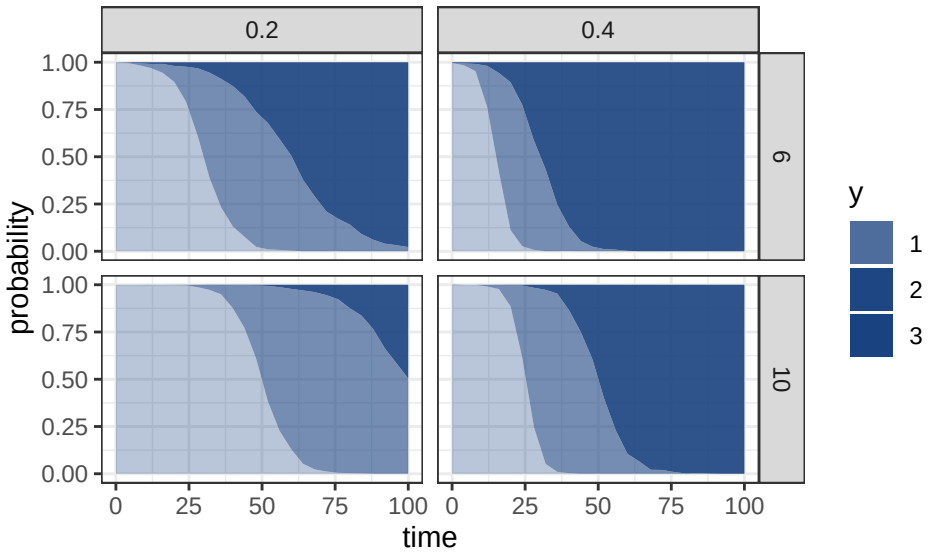
A unique graph is produced by ignoring the group

```
plot6 <- catplotmlx(res$y, group="none")
print(plot6)
```



If group is defined by two factors, `catplotmlx` will automatically layout panels in a grid (using `facet_grid`)

```
cov <- data.frame(id=levels(res$y$id), a=rep(c(6,10,6,10), each=Ng),
                  b=rep(c(0.2,0.2,0.4,0.4), each=Ng) )
plot7 <- catplotmlx(res$y, group=cov)
print(plot7)
```



Customized plots for groups

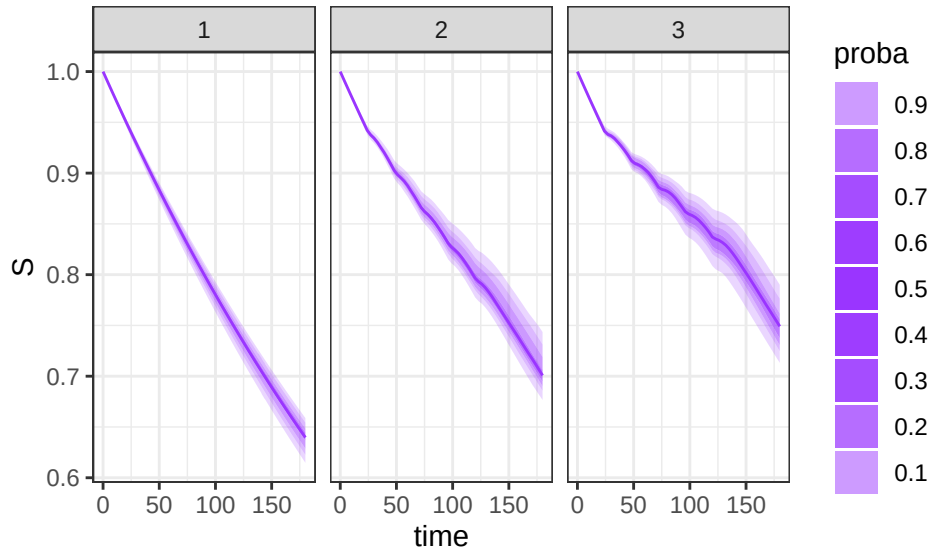
Plotting several prediction distributions on a same plot

Simulate first data from a joint model (PK + TTE) defining 3 treatment arms:

```
jointModel <- inlineModel("
[LONGITUDINAL]
input={Tk0, V, Cl, alpha, beta, a}

EQUATION:
C = pkmodel(Tk0, V, Cl)
h = exp(-alpha - beta*C)
H_0 = 0
ddt_H = h
S = exp(-H)

DEFINITION:
y = {distribution=lognormal, prediction=C, sd=a}
e = {type=event, maxEventNumber=1, rightCensoringTime=200, hazard=h}

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

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}
alpha = {distribution=lognormal,   prediction=alpha_pop,sd=omega_alpha}
")

#-----------------------------------------------
N=100
t.time <- seq(24,120,by=24)
g1 <- list(size=N, level='individual', treatment=list(time=t.time, amount=0))
g2 <- list(size=N, level='individual', treatment=list(time=t.time, amount=25))
g3 <- list(size=N, level='individual', treatment=list(time=t.time, amount=50))

pop.param   <- c(
  Tk0_pop   = 3,    omega_Tk0   = 0.3,
  V_pop     = 10,   omega_V     = 0.3,
  Cl_pop    = 1,    omega_Cl    = 0.3,
  alpha_pop = 6,    omega_alpha = 0.01,
  beta      = 0.4,  a           = 0.1
)

f <- list(name=c('C','S'), time=seq(0,to=180,by=3))
e <- list(name="e", time=0)

res1 <- simulx(model     = jointModel,
               parameter = pop.param,
               group     = list(g1, g2, g3),
               output    = list(f,e))
```

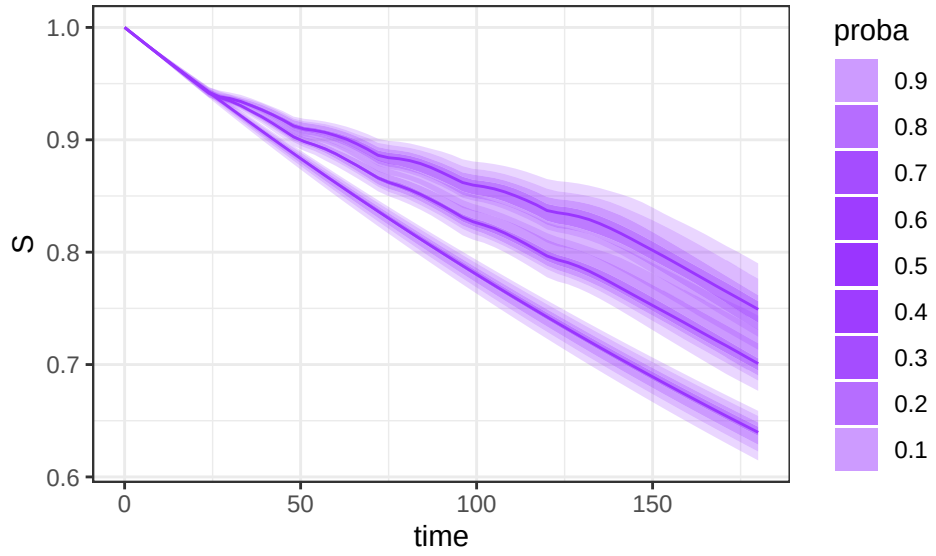
`prctilemlx` displays the 3 prediction intervals on 3 subplots

```
print(prctilemlx(res1$S))
```



or on the same plot:

```
print(prctilemlx(res1$S, facet=FALSE))
```



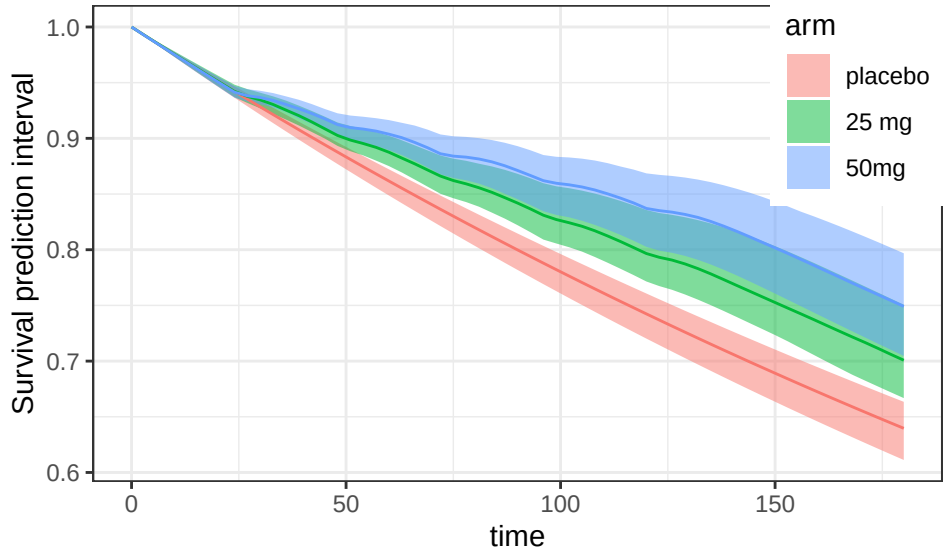
Alternatively, the function `plot.prediction.intervals` can be used if we want to display these prediction intevals with different colors.

Remark: the `R` function `plot.prediction.intervals` is not part of the `mlxR` package. This is just an example of what can be done: this function can be freely used/modified by anyone.

```
plot.prediction.intervals <- function(r, plot.median=TRUE, level=90, labels=NULL,
                                      legend.title=NULL, colors=NULL) {
  P <- prctilemlx(r, number=1, level=level, plot=FALSE)
  if (is.null(labels))  labels <- levels(r$group)
  if (is.null(legend.title))  legend.title <- "group"
  names(P$y)[2:4] <- c("p.min","p50","p.max")
```

```
  pp <- ggplot(data=P$y)+ylab(NULL)+
    geom_ribbon(aes(x=time,ymin=p.min, ymax=p.max,fill=group),alpha=.5)
  if (plot.median)
    pp <- pp + geom_line(aes(x=time,y=p50,colour=group))

  if (is.null(colors)) {
    pp <- pp + scale_fill_discrete(name=legend.title,
                                   breaks=levels(r$group),
                                   labels=labels)
    pp <- pp + scale_colour_discrete(name=legend.title,
                                     breaks=levels(r$group),
                                     labels=labels,
                                     guide=FALSE)
  } else {
    pp <- pp + scale_fill_manual(name=legend.title,
                                 breaks=levels(r$group),
                                 labels=labels,
                                 values=colors)
    pp <- pp + scale_colour_manual(name=legend.title,
                                   breaks=levels(r$group),
                                   labels=labels,
                                   guide=FALSE,values=colors)
  }
  return(pp)
}
```

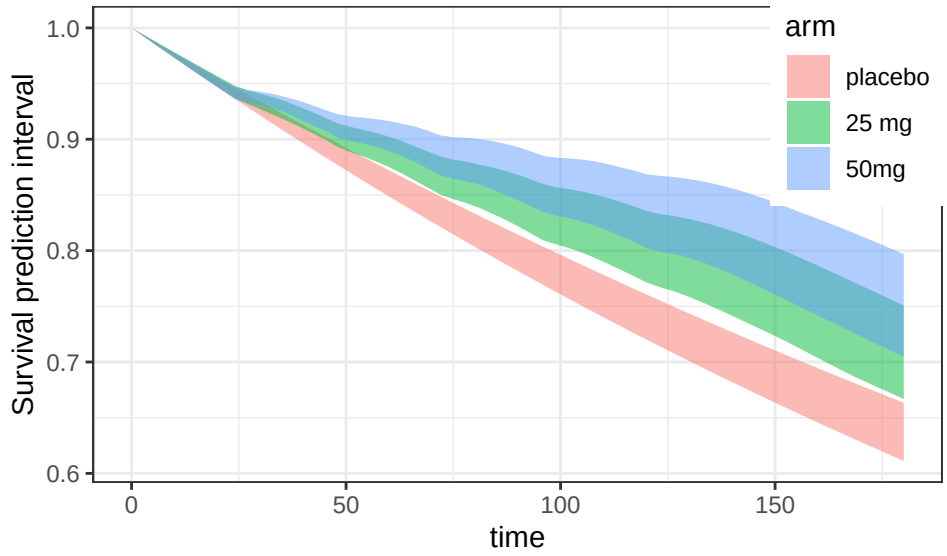
```
plot.S1 <- plot.prediction.intervals(res1$S,
                                     labels       = c("placebo","25 mg","50mg"),
                                     legend.title = "arm")
plot.S1 <- plot.S1  + ylab("Survival prediction interval") + theme(legend.position=c(0.9,0.8))
print(plot.S1)
```



The median can be removed

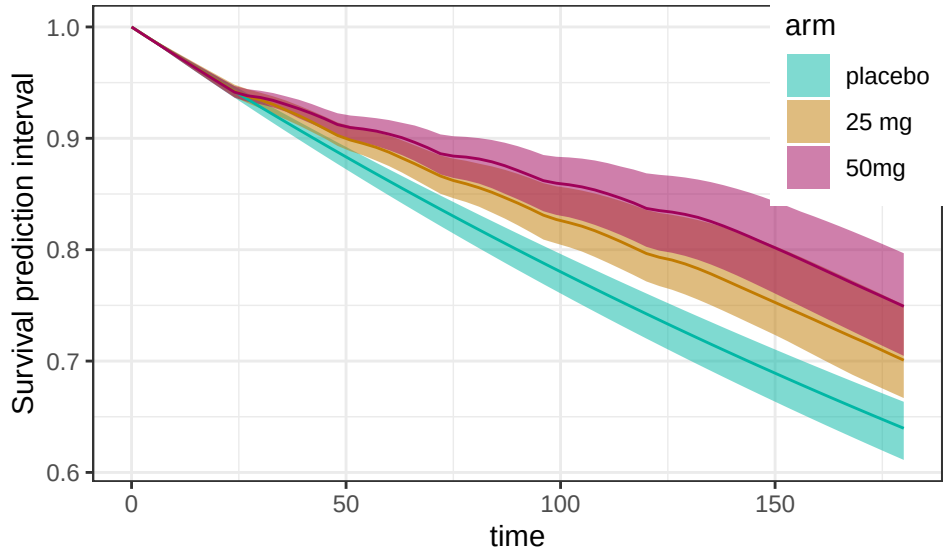
```
plot.S2 <- plot.prediction.intervals(res1$S,
                                     labels       = c("placebo","25 mg","50mg"),
```

```
                                    legend.title = "arm",
                                    plot.median  = FALSE)
plot.S2 <- plot.S2  + ylab("Survival prediction interval") + theme(legend.position=c(0.9,0.8))
print(plot.S2)
```



and the colors can be defined by the user

```
plot.S3 <- plot.prediction.intervals(res1$S,
                                    labels       = c("placebo","25 mg","50mg"),
                                    legend.title = "arm",
                                    colors       = c('#01b7a5', '#c17b01', '#a00159'))
plot.S3 <- plot.S3  + ylab("Survival prediction interval") + theme(legend.position=c(0.9,0.8))
print(plot.S3)
```



statmlx.R

Overview

Description

Compute statistical summaries (mean, quantile, variance, survival rate,...).

Usage

```
statmlx(r, FUN = "mean", probs = c(0.05, 0.5, 0.95), surv.time = NULL)
```

Arguments

`r`

- a data frame with a column `id`

`FUN`

- a string, or a vector of strings, with the name of the functions to apply to the result of the simulation

`probs`

- a vector of quantiles between 0 and 1. Only used when "quantile" has been defined in FUN

`surv.time`

- a scalar or a vector of times. Only used when "event" has been defined in type

Value

a data frame.

Example

```
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}
")

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)
out.y <- list(name=c('y'), time=seq(0,to=25,by=5))
out.e <- list(name=c('e'), time=0)
out.p <- c("V", "Cl")
out   <- list(out.y, out.e, out.p)
g <- list(size=100, level='individual')
res1 <- simulx(model=modelPK, treatment=adm, parameter=p, output=out, group=g)

statmlx(res1$parameter, FUN = "mean", probs = c(0.05, 0.5, 0.95))
```

```
##     V.mean  Cl.mean
## 1 10.09505 1.023188
```

```
statmlx(res1$parameter, FUN = "quantile", probs = c(0.05, 0.5, 0.95))
```

```
##       V.p5    V.p50    V.p95     Cl.p5   Cl.p50   Cl.p95
## 1 7.064748 9.920791 14.02987 0.7246583 1.004425 1.392795
```

```
statmlx(res1$parameter, FUN = c("sd", "quantile"), probs = c(0.05, 0.95))
```

```
##       V.sd     Cl.sd     V.p5    V.p95     Cl.p5   Cl.p95
## 1 2.066184 0.1961092 7.064748 14.02987 0.7246583 1.392795
```

```
statmlx(res1$y, FUN = c("mean", "sd", "quantile"), probs = c(0.05, 0.95))
```

```
##   time     y.mean      y.sd      y.p5     y.p95
## 1    0 10.4296457 2.3080859 7.6014789 15.203610
## 2    5  5.9842670 1.0441123 4.5885689  7.812777
## 3   10  3.5189957 0.7320124 2.3738216  4.665897
## 4   15  2.2141548 0.7196505 1.1379914  3.463530
## 5   20  1.3714524 0.6101297 0.5768636  2.631927
## 6   25  0.8494559 0.4614625 0.2557196  1.716282
```

```
statmlx(res1$e, surv.time=c(10,20))
```

```
##   nbEv.mean S10.mean S20.mean
## 1      0.54     0.75     0.58
```

```
res2 <- simulx(model=modelPK, treatment=adm, parameter=p, output=out, group=g, nrep=3)
statmlx(res2$parameter, FUN = c("sd", "quantile"), probs = c(0.05, 0.95))
```

```
##   rep     V.sd     Cl.sd     V.p5    V.p95     Cl.p5   Cl.p95
## 1   1 1.956987 0.1873184 7.670040 13.67523 0.8321865 1.401149
## 2   2 2.100450 0.2142459 7.406265 13.62250 0.7698099 1.495370
## 3   3 2.210193 0.1953296 6.801339 13.69480 0.7527140 1.339947
```

```
statmlx(res2$y, FUN = c("mean", "sd", "quantile"), probs = c(0.05, 0.95))
```

```
##    rep time     y.mean      y.sd      y.p5     y.p95
## 1    1    0  9.7008325 2.1047035 7.0423117 13.390844
## 6    1    5  5.7340060 0.8997047 4.4894696  7.267318
## 2    1   10  3.3222626 0.6387232 2.2356943  4.209951
## 3    1   15  2.0416497 0.6605197 1.0785720  3.163436
## 4    1   20  1.2363056 0.4948520 0.5377879  1.988325
## 5    1   25  0.7652786 0.3928911 0.2108409  1.451873
## 7    2    0 10.1834469 2.2401588 7.1258095 14.100451
## 12   2    5  6.0238848 1.0877527 4.5551293  8.080962
## 8    2   10  3.5852785 0.8040753 2.2982555  4.743856
```

```
## 9    2   15  2.1430816 0.6987445 0.9114754  3.266766
## 10   2   20  1.3278161 0.5847167 0.4888474  2.323114
## 11   2   25  0.8222328 0.4422644 0.1816241  1.582306
## 13   3    0 10.5313108 2.3864855 6.8882294 14.803442
## 18   3    5  6.0429475 1.1047404 4.1134649  7.774528
## 14   3   10  3.5472738 0.8048637 2.3879437  4.641677
## 15   3   15  2.0865997 0.6669426 1.0964079  3.353405
## 16   3   20  1.2938832 0.5682448 0.5170885  2.405244
## 17   3   25  0.7832306 0.4425990 0.2084871  1.617373
```

```
statmlx(res2$e, surv.time=c(10,20,30))
```

```
##   rep nbEv.mean S10.mean S20.mean S30.mean
## 1   1      0.54     0.63     0.51     0.46
## 2   2      0.52     0.70     0.59     0.48
## 3   3      0.68     0.59     0.40     0.32
```

readDatamlx.R

Overview

Description

Read data in a Monolix/NONMEM format and Monolix project

Usage

```
readDatamlx(project = NULL, datafile = NULL, header = NULL, infoProject = NULL, addl.ss = 10)
```

Arguments

`project`

- a Monolix project

`datafile`

- a formatted data file

`header`

- a vector of strings (mandatory if datafile is used)

`infoProject`

- an xmlfile

`addl.ss`

- number of additional doses to use for steady-state (default=10)

Value

A list of data frame.

Reading a data file

The header

Let us use the warfarin data file in a Monolix format:

```
head(read.table('data/warfarin_data.txt', header=TRUE, sep="\t"))
```

```
##    id time amt  dv dvid   wt age sex mdv
## 1 100  0.0 100   .    1 66.7  50   1   1
## 2 100  0.5   .   0    1 66.7  50   1   1
## 3 100  1.0   . 1.9    1 66.7  50   1   1
## 4 100  2.0   . 3.3    1 66.7  50   1   1
## 5 100  3.0   . 6.6    1 66.7  50   1   1
## 6 100  6.0   . 9.1    1 66.7  50   1   1
```

`readDatamlx(datafile= ..., header=...)` reads a Monolix data file and create several data frames. A header should be provided, using the following reserved key-words:

`id, time, amt, rate, tinf, y, ytype, x, cov, cat, occ, mdv, evid, addl, ss, ii, ignore`

In this example, `wt` and `age` are continuous covariates while `sex` is categorical

```
d <- readDatamlx(datafile = 'data/warfarin_data.txt',
                 header   = c('id','time','amt','y','ytype','cov','cov','cat'))
names(d)
```

```
## [1] "treatment" "y1"        "y2"        "covariate" "id"        "N"         "catNames"
```

```
head(d$treatment)
```

```
##    id time amount
## 1 100    0    100
## 2   1    0    100
## 3   2    0    100
## 4   3    0    120
## 5   4    0     60
## 6   5    0    113
```

```
head(d$covariate)
```

```
##    id   wt age sex
## 1 100 66.7  50   1
## 2   1 66.7  50   1
## 3   2 66.7  31   1
## 4   3 80.0  40   1
## 5   4 40.0  46   0
## 6   5 75.3  43   1
```

```
head(d$y1)
```

```
##    id time   y1
## 1 100  0.5  0.0
## 2 100  1.0  1.9
## 3 100  2.0  3.3
## 4 100  3.0  6.6
## 5 100  6.0  9.1
## 6 100  9.0 10.8
```

```
warfarin.data <- list(dataFile = 'data/warfarin_data.txt',
                      headerTypes   = c('id','time','amt','y','ytype','cov','cov','cat'))
d <- readDatamlx(data = warfarin.data)
```

Some columns can be ignored:

```
d <- readDatamlx(datafile = 'data/warfarin_data.txt',
                 header   = c('id','time','amt','y','ytype','cov','ignore','ignore'))
head(d$covariate)
```

```
##    id   wt
## 1 100 66.7
## 2   1 66.7
## 3   2 66.7
## 4   3 80.0
## 5   4 40.0
## 6   5 75.3
```

EVID and MDV columns

Column `EVID` can be used instead of dots (".") in the data file

```
head(read.table('data/warfarin_data_evid.txt', header=TRUE, sep="\t"))
```

```
##    id time amt  dv dvid   wt age sex evid
## 1 100  0.0 100   0    1 66.7  50   1    1
## 2 100  0.5   0   0    1 66.7  50   1    0
## 3 100  1.0   0 109    1 66.7  50   1    0
## 4 100  2.0   0 303    1 66.7  50   1    0
## 5 100  3.0   0 606    1 66.7  50   1    0
## 6 100  6.0   0 901    1 66.7  50   1    0
```

```
d <- readDatamlx(datafile = 'data/warfarin_data_evid.txt',
                 header   = c('id','time','amt','y','ytype','cov','cov','cat','evid'))
head(d$treatment)
```

```
##     id time amount
## 1  100    0    100
## 20   1    0    100
## 34   2    0    100
## 54   3    0    120
## 72   4    0     60
## 91   5    0    113
```

Column `MDV` allows one to ignore some observations

```
head(read.table('data/warfarin_data_mdv.txt', header=TRUE, sep="\t"))
```

```
##    id time amt  dv dvid   wt age sex mdv
## 1 100  0.0 100   .    1 66.7  50   1   1
## 2 100  0.5   .   0    1 66.7  50   1   1
## 3 100  1.0   . 1.9    1 66.7  50   1   1
## 4 100  2.0   . 3.3    1 66.7  50   1   1
## 5 100  3.0   . 6.6    1 66.7  50   1   1
## 6 100  6.0   . 9.1    1 66.7  50   1   1
```

```
d <- readDatamlx(datafile = 'data/warfarin_data_mdv.txt',
                 header   = c('id','time','amt','y','ytype','cov','cov','cat','mdv'))
head(d$y1)
```

```
##     id time  y1
## 1  100   24 5.6
## 3  100   36 4.0
## 5  100   48 2.7
## 7  100   72 0.8
## 12   1   24 9.2
## 14   1   36 8.5
```

SS and ADDL columns

A column `SS` may be used in the data file in order to indicate that the system is at *steady-state*. Column `II` (interdose interval) is the time between two successive doses.

```
head(read.csv('data/dataSS.csv'))
```

```
##   time        y amount ss ii
## 1    0        .    100  1 12
## 2    1  3.95484      .  .  .
## 3    3 10.28948      .  .  .
## 4    5  5.74431      .  .  .
```

```
## 5    7  3.70618      .  .  .
## 6    9  2.60192      .  .  .
```

By default, `readDatamlx` adds 10 doses to reach steady state.

```
d <- readDatamlx(datafile = "data/dataSS.csv",
                 header   = c('time','y','amt','ss','ii'))
d$treatment
```

```
##     time amount
## 1.9 -120    100
## 1.8 -108    100
## 1.7  -96    100
## 1.6  -84    100
## 1.5  -72    100
## 1.4  -60    100
## 1.3  -48    100
## 1.2  -36    100
## 1.1  -24    100
## 11   -12    100
## 1      0    100
```

This number of additional doses to reach steady-state may be modified with the input argument `addl.ss`

```
d <- readDatamlx(datafile = "data/dataSS.csv",
                 header   = c('time','y','amt','ss','ii'),
                 addl.ss  = 5)
d$treatment
```

```
##     time amount
## 1.9 -120    100
## 1.8 -108    100
## 1.7  -96    100
## 1.6  -84    100
## 1.5  -72    100
## 1.4  -60    100
## 1.3  -48    100
## 1.2  -36    100
## 1.1  -24    100
## 11   -12    100
## 1      0    100
```

Any number of additional doses may be considered, before the current dose using a negative number (here, 5 doses are added before t=0),

```
head(read.csv('data/dataADDL1.csv'))
```

```
##   time        y amount addl ii
## 1    0        .    100   -5 12
## 2    1  3.95484      .    .  .
## 3    3 10.28948      .    .  .
## 4    5  5.74431      .    .  .
## 5    7  3.70618      .    .  .
## 6    9  2.60192      .    .  .
```

```
d <- readDatamlx(datafile = "data/dataADDL1.csv",
                 header   = c('time','y','amt','addl','ii'))
d$treatment
```

```
##     time amount
## 1.4  -60    100
## 1.3  -48    100
## 1.2  -36    100
## 1.1  -24    100
## 11   -12    100
## 1      0    100
```

or after the current dose with a positive number of doses (here, 5 doses are added after t=-60),

```
head(read.csv('data/dataADDL2.csv'))
```

```
##   time        y amount addl ii
## 1  -60        .    100    5 12
## 2    1  3.95484      .    .  .
## 3    3 10.28948      .    .  .
## 4    5  5.74431      .    .  .
## 5    7  3.70618      .    .  .
## 6    9  2.60192      .    .  .
```

```
d <- readDatamlx(datafile = "data/dataADDL2.csv",
                 header   = c('time','y','amt','addl','ii'))
d$treatment
```

```
##     time amount
## 1    -60    100
## 11   -48    100
## 1.1  -36    100
## 1.2  -24    100
## 1.3  -12    100
## 1.4    0    100
```

##Reading a Monolix project

```
project.file <- 'monolixRuns/theophylline_project.mlxtran'
```

`readDatamlx(project)` reads a Monolix project and creates several data frames

```
theo.data <- readDatamlx(project = project.file)
names(theo.data)
```

```
## [1] "populationParameters" "treatment"            "y"
## [4] "covariate"            "id"                   "N"
```

```
head(theo.data$y)
```

```
##   id time     y
## 1  1 0.25  2.84
## 2  1 0.57  6.57
## 3  1 1.12 10.50
## 4  1 2.02  9.66
## 5  1 3.82  8.58
## 6  1 5.10  8.36
```

`readDatamlx(project, out.data=TRUE)` returns a list with the original data and information about the project

```
theo.data <- readDatamlx(project = project.file, out.data=TRUE)
names(theo.data)
head(theo.data$data)    #  original data
```

```
theo.data$infoProject$datafile #  path & name of the data file
theo.data$infoProject$dataheader # header/keywords used by Monolix
theo.data$infoProject$resultFolder  # result folder
```

writeDatamlx.R

Overview

Description

Write data contained in a list of dataframes in a single file (NONMEM/Monolix format) or in several files as tables.

Usage

```
writeDatamlx(r, result.file = NULL, result.folder = NULL, sep = ",", ext = NULL,
             digits = 5, app.file = F, app.dir = F)
```

Arguments

`r`

- a list of dataframes

`result.file`

- a string with the name of the file

`result.folder`

- a string with the name of the folder

`sep`

- (default = ",")

`ext`

- a string with the extension of the file names

`digits`

- (default = 5)

`app.file`

- TRUE/FALSE (default=FALSE) append to file

`app.dir`

- TRUE/FALSE (default=FALSE) append to dir

Writing simulated data

Single output

```
model1 <- inlineModel("
[LONGITUDINAL]
input = {V, Cl, a1}

EQUATION:
Cc = pkmodel(V, Cl)

DEFINITION:
y1 ={distribution=lognormal, prediction=Cc, sd=a1}
")
```

```
adm  <- list(amount=100, time=seq(0,50,by=24))
p <- c(V=10, Cl=1, a1=0.1)
y1 <- list(name='y1', time=seq(5,to=50,by=5))

res1a <- simulx( model     = model1,
                 treatment = adm,
                 parameter = p,
                 output    = y1,
                 group     = list(size=5),
                 settings  = list(seed = 32323))
```

Use `writeDatamlx` to write the results of the simulation in a `csv` file using the standard format used by Monolix,

```
writeDatamlx(res1a, result.file = "res1a.csv")
head(read.csv("res1a.csv"))
```

```
##   id time       y amount
## 1  1    0       .    100
## 2  1    5 5.75182      .
## 3  1   10 3.50217      .
## 4  1   15 2.05825      .
## 5  1   20 1.43786      .
## 6  1   24       .    100
```

in a `txt` file,

```
writeDatamlx(res1a, result.file = "res1a.txt", sep="\t")
head(read.table("res1a.txt", header=TRUE, sep="\t"))
```

```
##   id time       y amount
## 1  1    0       .    100
## 2  1    5 5.75182      .
## 3  1   10 3.50217      .
## 4  1   15 2.05825      .
## 5  1   20 1.43786      .
## 6  1   24       .    100
```

or in several files in a folder

```
writeDatamlx(res1a, result.folder = "res1a")
list.files(path="res1a")
```

```
## [1] "treatment.csv" "y1.csv"
```

Data can be written both in a single file and in separated files in a folder

```
writeDatamlx(res1a, result.file = "res1a.csv",result.folder="res1a")
```

Remark: instead of using `writeDatamlx`, it is possible to ask `simulx` to write the results of the simulation in a file and/or in a folder.

```
res1b <- simulx( model         = model1,
                 treatment     = adm,
                 parameter     = p,
                 output        = y1,
                 group         = list(size=5, level="individual"),
                 result.file   ="res1b.csv",
```

```
                result.folder ="res1b",
                settings      = list(seed = 32323))

head(read.csv("res1b.csv"))
```

```
##   time      y amount
## 1    0      .    100
## 2    5 5.7518      .
## 3   10 3.5022      .
## 4   15 2.0583      .
## 5   20 1.4379      .
## 6   24      .    100
```

```
list.files(path="res1b")
```

```
## [1] "treatment.csv" "y1.csv"
```

Groups and replicates

A column `group` and a column `rep` are added to the data file when several groups and several replicates are defined

```
g1 = list(size=5, treatment=list(amount=100, time=seq(0,50,by=12)))
g2 = list(size=3, treatment=list(amount=50,  time=seq(0,50,by=12)))
res1c <- simulx( model     = model1,
                 parameter = p,
                 group     = list(g1,g2),
                 nrep      = 10,
                 output    = y1)

writeDatamlx(res1c, result.file = "res1c.csv")
head(read.csv("res1c.csv"))
```

```
##   rep id group time       y amount
## 1   1  1     1    0       .    100
## 2   1  1     1    5 7.20051      .
## 3   1  1     1   10  3.9863      .
## 4   1  1     1   12       .    100
## 5   1  1     1   15 9.26273      .
## 6   1  1     1   20 5.79117      .
```

Censored data

A column `cens` and a column `limit` are added when (left/right/interval) censored data are simulated

```
y1 <- list(name='y1', time=seq(5,to=50,by=5) , lloq=3, limit=0)
res1d <- simulx( model     = model1,
                 treatment = adm,
                 parameter = p,
                 group     = list(size=5),
                 output    = y1)

writeDatamlx(res1d, result.file = "res1d.csv")
head(read.csv("res1d.csv"))
```

```
##   id time       y cens limit amount
## 1  1    0       .    .     .    100
```

```
## 2  1    5 7.49322    0     .      .
## 3  1   10 4.15957    0     .      .
## 4  1   15       3    1     0      .
## 5  1   20       3    1     0      .
## 6  1   24       .    .     .    100
```

Multiple outputs

```
model2 <- 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)
y2 <- list(name='y2', time=seq(2,to=50,by=6))

res2a <- simulx( model     = model2,
                 treatment = adm,
                 parameter = p,
                 group     = list(size=5, level="individual"),
                 output    = list(y1,y2))
```

A column `ytype` is created when several outputs are written in a single data file

```
writeDatamlx(res2a, result.file = "res2a.csv")
head(read.csv("res2a.csv"))
```

```
##   id time        y cens limit ytype amount
## 1  1    0        .    .     .     .    100
## 2  1    2 79.14571    0     .     2      .
## 3  1    5  7.30899    0     .     1      .
## 4  1    8 70.93016    0     .     2      .
## 5  1   10  5.58433    0     .     1      .
## 6  1   14 60.84051    0     .     2      .
```

One file per output is created when the data is written into a folder

```
writeDatamlx(res2a, result.folder = "res2a")
list.files(path="res2a")
```

```
## [1] "treatment.csv" "y1.csv"        "y2.csv"
```

Using a Monolix project

Using a simplified format

Let us simulate some data using a Monolix project,

```
project.file <- 'monolixRuns/theophylline_project.mlxtran'
sim.res1  <- simulx(project = project.file , setting=list(seed=123456))
```

We can then write the results in a data file. Using `writeDatamlx(..., result.file=...)`, information about the project is not used,

```
writeDatamlx(sim.res1, result.file="theosim.csv")
head(read.csv("theosim.csv"))
```

```
##   id time       y amount
## 1  1 0.00       .    320
## 2  1 0.25 3.22263      .
## 3  1 0.57 5.76049      .
## 4  1 1.12 7.02616      .
## 5  1 2.02 8.06874      .
## 6  1 3.82 6.80563      .
```

Information about the project can be used by defining the Monolix project as an input argument of `writeDatamlx`. In this case, a file with the simulated data is created in the result folder, with the prefix "sim_" added to the name of the original data file,

```
writeDatamlx(sim.res1, project=project.file)
head(read.table("monolixRuns/theophylline_project/sim_theophylline_data.txt",header=TRUE))
```

```
##   ID AMT AMT.KG TIME    CONC WEIGHT SEX
## 1  1 320   4.02 0.00       .   79.6   M
## 2  1   .      . 0.25 3.22263   79.6   M
## 3  1   .      . 0.57 5.76049   79.6   M
## 4  1   .      . 1.12 7.02616   79.6   M
## 5  1   .      . 2.02 8.06874   79.6   M
## 6  1   .      . 3.82 6.80563   79.6   M
```

Another file name can be used

```
writeDatamlx(sim.res1, project=project.file, result.file="theosim.txt")
head(read.table("theosim.txt",header=TRUE))
```

```
##   ID AMT AMT.KG TIME    CONC WEIGHT SEX
## 1  1 320   4.02 0.00       .   79.6   M
## 2  1   .      . 0.25 3.22263   79.6   M
## 3  1   .      . 0.57 5.76049   79.6   M
## 4  1   .      . 1.12 7.02616   79.6   M
## 5  1   .      . 2.02 8.06874   79.6   M
## 6  1   .      . 3.82 6.80563   79.6   M
```

Writing the results in several files

The results can also be saved in separated files in a folder

```
writeDatamlx(sim.res1, project=project.file, result.folder="theo")
dir("theo")
```

```
## [1] "originalId.csv" "treatment.csv"  "y.csv"
```

Using the original format of the data

Note that the original format can be used by setting format.original=TRUE in the settings of simulx:

```
res2 <- simulx(project=project.file, result.file="theo_test.txt",settings=list(format.original=TRUE))
print(head(read.table("theo_test.txt", sep="\t", header=TRUE)))
```

```
##   ID AMT AMT.KG TIME    CONC WEIGHT SEX
## 1  1 320   4.02 0.00       .   79.6   M
## 2  1   .      . 0.25 0.60046   79.6   M
## 3  1   .      . 0.57 0.83014   79.6   M
## 4  1   .      . 1.12  2.3177   79.6   M
## 5  1   .      . 2.02  3.5167   79.6   M
## 6  1   .      . 3.82  4.5711   79.6   M
```