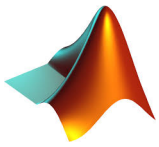
Mlxm 1.1



A toolbox for the simulation and visualization of mixed effects models

Marc Lavielle and Fazia Bellal,
Inria, Popix team
Saclay, France

June 2014

With contributions by
Elodie Maillot, Laura Brocco and Célia Barthélémy
Inria, Popix team

Contents

Chapter 1

Introduction

1.1 Description

Mlxm is a set of MATLAB functions for performing various tasks using MLXTRAN or PharmML models:

- simulx: compute predictions and simulate data from mixed effects models,
- pkmodel: compute predicted concentrations using compartmental PK models,
- mlxplore: visualize models.
- plotmlx: plot outputs of simulx and pkmodel (longitudinal data)
- kmplotmlx: Kaplan Meier plot for outputs of simulx (time-to-event data)
- displaymlx: display outputs of simulx (individual parameters)

MLXTRAN is a declarative language designed for encoding hierarchical models, including complex mixed effects models. MLXTRAN is also a particularly powerful solution for encoding dynamical systems represented by a system of ordinary differential equations. MLXTRAN is used by several software tools including MONOLIX for modeling and mlxplore for model exploration.

PharmML stands for *Pharmacometrics Markup Language* – a new standard for encoding of models, associated tasks and their annotation as used in pharmacometrics (http://www.pharmml.org). PharmML is being developed by the DDMoRe consortium, a European Innovative Medicines Initiative project (http://www.ddmore.eu).

Mlxm requires Mlxlibrary, a C++ based library developed by Lixoft. Mlxlibrary includes the model computation engine Mlxcompute that allows to solve efficiently complex systems of ordinary differential equations (ODEs) and delayed differential equations (DDEs).

Mlxm was developed by Inria and is governed by the CeCILL-B license. You can use, modify and/ or redistribute the software under the terms of the CeCILL license as circulated by CEA, CNRS and INRIA at the following URL

http://www.cecill.info/index.en.html

1.2 Installing and running Mlxm

1.2.1 Installation

– install Mlxlibrary from the Lixoft website

`http://download.lixoft.com?software=mlxlibrary`

– download and unzip `Mlxm110.zip`

`https://team.inria.fr/popix/mlxtoolbox/`

1.2.2 Running Mlxm

– start MATLAB

– change the working directory to `Mlxm110`

– run `initMlxm`
`>>initMlxm`

You can then run any demos or create your own MATLAB script.

Chapter 2

simulx

2.1 Introduction

2.1.1 Overview

simulx is a MATLAB function for computing predictions and sampling data from MLXTRAN and PharmML models.

simulx takes advantage of the modularity of hierarchical models for simulating different components of a model: models for population parameters, individual covariates, individual parameters and longitudinal data. Furthermore, simulx allows to draw different types of longitudinal data, including continuous, count, categorical, and time-to-event data.

We give here a very short overview of the input and output arguments lists. More details are provided with the examples in the next sections of the documentation.

Usage

res = **simulx**('model','mlxt.txt','parameter',par,'output',out) simulates the outputs defined in out using model mlxt.txt with parameters values defined in par and returns the results in the cell array res.

res = **simulx**('model', ... ,'treatment',tr) uses the *source terms* (i.e. the doses for pharmacometrics applications) defined in tr.

res = **simulx**('model', ... ,'group',gr) creates groups using the information in gr.

res = **simulx**('model', ... ,'settings',s) uses the settings defined in the structure s.

[res, dataIn] = **simulx**(...) returns the input argument dataIn of the mex-file mlxCompute.

dataOut = **simulx**('data', dataIn) directly calls the mex-file mlxCompute with the input argument dataIn. simulx returns dataOut, the output of mlxCompute. This usage of simulx can be much faster than the standard one.

Input arguments

model	a model MLXTRAN or PharmML used for the simulation

parameter	a structure with fields	
	name	a cell array of parameters names
	value	a vector of parameters values
output	a a structure with fields	
	name	a cell array of outputs names
	time	a vector of times (for the longitudinal outputs)
treatment (*opt.*)	a a structure with fields	
	time	a vector of inputs times
	amount	a scalar or a vector of amounts
	rate (*opt.*)	a scalar or a vector of infusion rates
	type (*opt.*)	an integer (in the case of multiple types of administration)
group (*opt.*)	a structure with optional fields	
	size	size of the group (default=1)
	level	level of randomization (default="longitudinal")
	parameter	used for different parameters per group
	output	used for different outputs per group
	treatment	used for different treatments per group
settings (*opt.*)	a structure with optional settings	
	recordFile	name of the datafile where the simulated data is stored
	seed	initialization of the random number generator (integer)

Output arguments

res a cell array with the outputs defined in the input argument output and their values

Extension

Remark: simulx can use models implemented both in MLXTRAN and PharmML. For the sake of clarity, most of the examples provided in this documentation use models implemented with the MLXTRAN language: see the documentation for more information about MLXTRAN (Lixoft, 2014a,b). Examples of the use of simulx with PharmML are proposed in Section 2.9.

2.1.2 Introduction examples

The demos used in this section are in /demos/1_introduction

EXAMPLE 1.1 We consider the function f solution of

$$\dot{f}(t) = \frac{u\, f(t)}{v + f(t)} \tag{2.1}$$

for $t \geq 0$ and $f(t) = 10$ for $t < 0$.

MLXTRAN script for encoding this model:

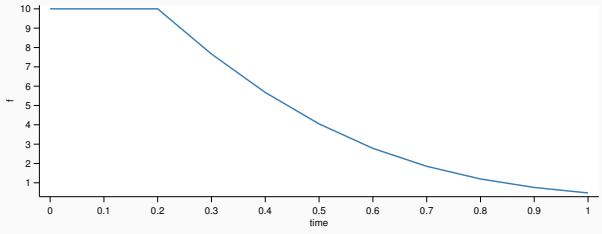
intro1_model.txt

```
[LONGITUDINAL]
input = {u, v, a}

EQUATION:
t0=0.2
f_0=10
ddt_f = u*f/(v+f)
```

`simulx` script for computing $(f(t_j)$ at times $t_j = 0, 0.1, 0.2, \ldots, 0.9, 1$, with $u = 50$, $v_{\text{pop}} = 10$,

intro1_simulx.m

```
f.name = 'f' ;
f.time = 0:0.1:1 ;

p.name  = {'u','v'} ;
p.value = [50,10] ;

res = simulx( 'model','intro1_model.txt',...
              'parameter',p,...
              'output',f);
plotmlx(res)
```



EXAMPLE 1.2 We consider now the following nonlinear mixed effects model:

$$y_{ij} = f(t_{ij}; u, v_i) + a\varepsilon_{ij} \ ; \ 1 \leq i \leq N \ , \ 1 \leq j \leq n_i$$

where f is defined in (2.1) and where

$$\begin{aligned} \varepsilon_{ij} &\underset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1) \\ \log(v_i) &\underset{\text{i.i.d.}}{\sim} \mathcal{N}(\log(v_{\text{pop}}), \omega_v^2) \end{aligned}$$

The MLXTRAN script for encoding this model now has a section `[LONGITUDINAL]` for the model for longitudinal data (y_{ij}) and a section `[INDIVIDUAL]` for the model for individual parameters (v_i)

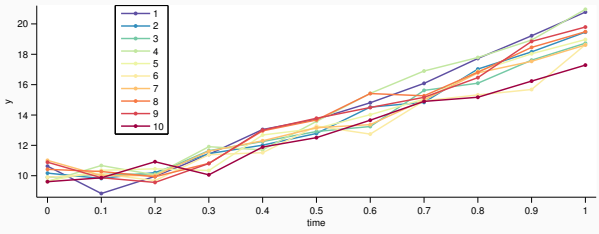
intro2_model.txt

```
[LONGITUDINAL]
input = {u, v, a}

EQUATION:
t0=0.2
f_0=10
ddt_f = u*f/(v+f)

DEFINITION:
y = {distribution=normal, prediction=f, sd=a}

;----------------------------------------
[INDIVIDUAL]
input = {v_pop, omega_v}

DEFINITION:
v = {distribution=logNormal, mean=log(v_pop), sd=omega_v}
```

simulx script for simulating $(y_{ij}, 1 \leq i \leq 10)$ at times $t_{ij} = 0, 0.1, 0.2, \ldots, 0.9, 1$, with $u = 50$, $v_{\text{pop}} = 10$, $\omega_v = 0.2$ and $a = 0.5$:

intro2_simulx.m

```
y.name  = 'y';
y.time  = 0:0.1:1;

p.name  = {'u','v_pop','omega_v','a'};
p.value = [20,10,0.2,0.5];

g.size  = 10;
g.level = 'individual';

res = simulx( 'model', 'intro2_model.txt',...
              'parameter', p,...
              'output', y,...
              'group', g);

opt = struct('symbol','.-');
plotmlx(res,'option',opt)
```



2.2 Simulating longitudinal data

The demos used in this section are in /demos/2_longitudinal

The model for longitudinal data is implemented in the [LONGITUDINAL] section of a script MLXTRAN.

Functions of time can be defined in a block EQUATION and can include ordinary differential equations (ODE) and delayed differential equations (DDE). We consider parametric functions of time of the form $f(t;\phi)$, where ϕ is a vector of structural parameters. We will see Section 2.4 how to define f as the solution of a dynamical system with source terms (e.g. doses for pharmacokinetics models).

The probability distribution of longitudinal data can then be defined in a DEFINITION block, using the functions of time defined in the EQUATION block. We consider parametric distributions that depend on a vector of parameters ξ.

The [LONGITUDINAL] section starts with the definition of the vector of input parameters $\psi = (\phi, \xi)$ in a list input.

Then, the simulx script should provide

- the model which is used (i.e. the name of the MLXTRAN or PharmML script),
- the values of the components of ψ,
- the list of desired outputs and the times at which they should be evaluated (if they are functions) or simulated (if they are data).

The results are returned as a cell array.

2.2.1 Evaluating functions of time

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

prediction_model.txt

```
[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))
```

We may want to evaluate both f_1 and f_2 at the same time points:

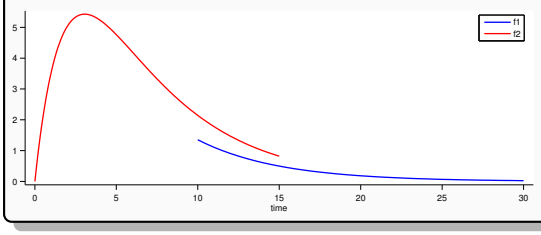
prediction1_simulx.m

```
p = struct('name',{{'ka','V','k'}},'value',[0.5,10,0.2]);
f = struct('name',{{'f1','f2'}},'time',0:0.1:30);
res = simulx('model','prediction_model.txt','parameter',p,'output',f);
```

It is also possible to evaluate f_1 and f_2 at different time points. It is necessary in this case to define first the two outputs in different structures and then the list of outputs as a cell array:

prediction2_simulx.m

```
p = struct('name',{{'ka','V','k'}},'value',[0.5,10,0.2]);
f1 = struct('name','f1','time',10:0.5:30);
f2 = struct('name','f2','time',0:0.1:15);
f = {f1, f2};
res = simulx('model','prediction_model.txt','parameter',p,'output',f);

opt = struct;
opt.output{1} = {'f1','f2'};
opt.symbol{1} = {'b-','r-'};
plotmlx(res,'option',opt)
```



2.2.2 Simulating longitudinal data

MLXTRAN allows us to define different types of longitudinal data in a DEFINITION block.

Continuous data

Possible distributions for continuous data are: normal, log-normal, probit-normal and logit-normal[1].

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

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

where f and g are defined in the EQUATION block and where $\varepsilon_j \sim_{\text{i.i.d.}} \mathcal{N}(0,1)$. Here, $f(t_j;\psi)$ is the *prediction* of y_j.

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

continuous_model.txt

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

EQUATION:
D=100
f1 = D/V*exp(-k*t)
f2 = D*ka/(V*(ka-k))*(exp(-k*t) - exp(-ka*t))
g1=a+b*f1

DEFINITION:
y1 = {distribution=normal, prediction=f1, sd=g1}
y2 = {distribution=logNormal, prediction=f2, sd=b}
```

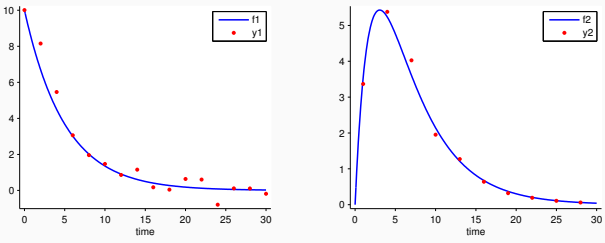
In this example, the sampling times for $(y_j^{(1)})$ and $(y_j^{(2)})$ are different.

We can use the plotmlx function to display the predictions and the simulated data.

[1] An extensive library of distributions will be available in a next release

continuous_simulx.m

```
p   = struct('name',{{'ka','V','k','a','b'}},'value',[0.5,10,0.2,0.3,0.1]);
f   = struct('name',{{'f1','f2'}},'time',0:0.1:30);
y1 = struct('name','y1','time',0:2:30);
y2 = struct('name','y2','time',1:3:30);
out={f,y1,y2};
res = simulx('model','continuous_model.txt','parameter',p,'output',out);

opt.output = {{'f1','y1'} , {'f2','y2'}};
opt.symbol = {{'b-','r.'},{'b-','r.'}};
plotmlx(res,'option',opt)
```



Count data

The data's distribution is defined by explicitly providing the probability mass function of y_j.

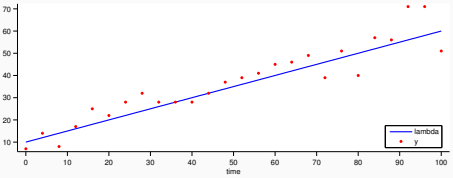
Here, the Poisson distribution is used for defining the distribution of y_j. The Poisson intensity is function of time:

count_model.txt

```
[LONGITUDINAL]
input =  {a,b}

EQUATION:
lambda=a +b*t

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

count_simulx.m

```
p = struct('name',{{'a','b'}},'value',[10,0.5]);
l = struct('name','lambda','time',0:1:100);
y = struct('name','y','time',0:4:100);
res = simulx('model','count_model.txt','parameter',p,'output',{l,y});

opt.output={{'lambda','y'}};
opt.symbol={{'b-','r.'}};
plotmlx(res,'option',opt)
```



Categorical data

The data's distribution is defined by giving explicitly the probability of each category for each y_j.

In this example, y_j takes its values in $\{1, 2, 3\}$. The cumulative logits define the distribution of y_j.

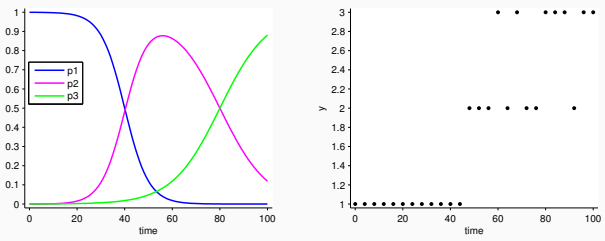
categorical_model.txt

```
[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
p3=1-p1-p2

DEFINITION:
y = {type=categorical, categories={1,2,3},
logit(P(y<=1))=lp1,logit(P(y<=2))=lp2}
```

The probabilities of each categories are not necessary for sampling the y_j's from this distribution. Nevertheless, they can also be computed and used in graphics for instance.

categorical_simulx.m

```
p = struct('name',{{'a','b'}},'value',[8,0.2]);
pr = struct('name',{{'p1','p2','p3'}},'time',0:1:100);
y = struct('name','y','time',0:4:100);
res = simulx('model','categorical_model.txt','parameter',p,'output',{pr,y});

opt.output={{'p1', 'p2', 'p3'}, 'y'};
opt.symbol={{'b-','m-','g-'},{'k.'}};
plotmlx(res,'option',opt)
```



Time-to-event data

The distribution of time-to-event data (or *survival* data) is defined with the *hazard* function. Some additional information should be provided in the model (maximum number of events, right censoring time, interval censoring, ...).

We consider a Weibull model for a single event in this example, with a right censoring time equal to 100.

event1_model.txt

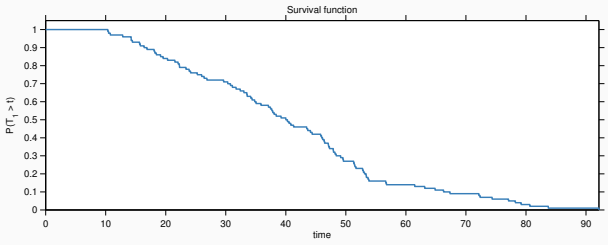
```
[LONGITUDINAL]
input = {beta,lambda}

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

DEFINITION:
e = {type=event, maxEventNumber=1, rightCensoringTime=100, hazard=h}
```

We simulate 100 individuals with the same model. An additional arguments `group` is thus necessary, with the number of individuals defined in the field `size` (see Section 2.5 for more detail about the use of `group`).

It is mandatory to state explicitly when the experiment starts (at time $t = 0$ in this example):

event1_simulx.m

```
p = struct('name',{{'beta','lambda'}},'value',[2.5,50]);
h = struct('name','h','time',0.1:0.1:30);
e = struct('name','e','time',0);
g = struct('size',100);
res = simulx('model','event1_model.txt','parameter',p,'output',{h,e},'group',g);

kmplotmlx(res(2))
```



We can use the same `simulx` script with a new model, assuming now that the events are repeated and interval censored:

event2_model.txt

```
[LONGITUDINAL]
input = {beta,lambda}

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

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

2.2.3 Joint longitudinal and time-to-event data model

We consider a joint model for PK and repeated events. The hazard is function of the predicted concentration.

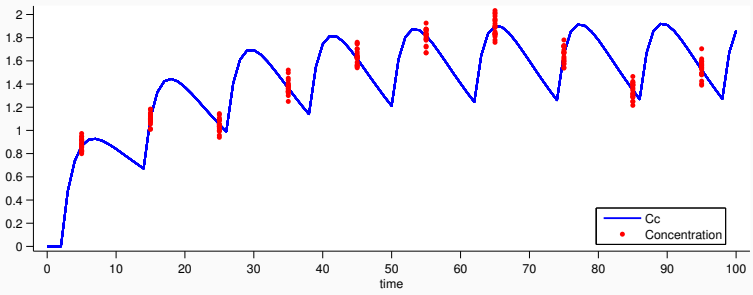
joint_model.txt

```
[LONGITUDINAL]
input = {ka, V, Cl, u, v, b}

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

DEFINITION:
Concentration = {distribution=normal, prediction=Cc, sd=b*Cc}
Hemorrhaging  = {type=event, hazard=h}
```

We may want to evaluate the predicted concentration and hazard for a given set of parameter values and simulate longitudinal data (i.e. concentrations and repeated events) for several subjects:

_simulx.m

```
a = struct('time',2:12:120,'amount',10);
p = struct('name',{{'ka','V','Cl','u','v','b'}},'value',[0.5,8,0.5,0.002,2,0.05]);
f = struct('name',{{'Cc', 'h'}},'time',0:100);
c = struct('name','Concentration','time',5:10:100);
e = struct('name','Hemorrhaging','time',0);
out = {f, c, e};
g = struct('size',20);
res = simulx('model','joint_model.txt','treatment',a,'group',g,...
             'parameter',p,'output',out);

kmplotmlx(res(4))
```



2.3 Simulation of hierarchical models

The demos used in this section are in `/demos/3_hierarchical`

Instead of considering only one individual, we will now consider N subjects in a population context. Then, the models we are interested with are hierarchical models that involves different types of variables:

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

Considering these variables as random variables, the joint probability distribution of $\boldsymbol{y}$, $\boldsymbol{\psi}$, $\boldsymbol{c}$ and θ can be decomposed into a product of conditional distributions (Lavielle, 2014):

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

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

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

```
[LONGITUDINAL]

[INDIVIDUAL]

[COVARIATE]

[POPULATION]
```

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

2.3.1 Simulating individual parameters

The model for the individual parameters is implemented in the `[INDIVIDUAL]` section.

hierarchical1_model.txt

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

[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}
```

Values of the population parameters are provided in the `simulx` script as well as the parameters with no variability.

hierarchical1a_simulx.m

```
p = struct('name',{{'V_pop','omega_V','w','w_pop','k','b'}},...
           'value',[10,0.2,75,70,0.2,0.2]);
f = struct('name','f','time',0:0.1:30);
y = struct('name','y','time',1:3:30);
V = struct('name','V');
out = {V,f,y};
res = simulx('model','hierarchical1_model.txt','parameter',p,'output',out);
```

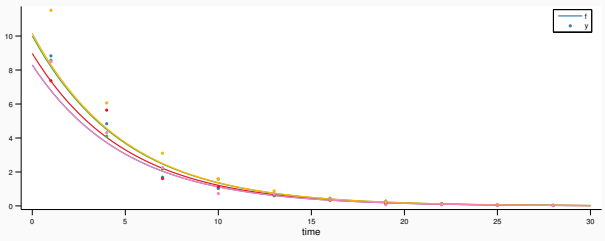
Remark It would be equivalent in this example to define k_{pop} in the `simulx` script and set $k = k_{\text{pop}}$ in the `[INDIVIDUAL]` section of the MLXTRAN script.

It is possible to simulate several individual using `group`. 5 individual are simulated in the next example.

`'level','individual'` means that each individual has its own set of (simulated) individual parameters.

hierarchical1b_simulx.m

```
>>g = struct('size',5,'level','individual');
>>res = simulx('model','hierarchical1_model.txt','parameter',p,'output',out,'group',g);
>>displaymlx(res)

    id          V
     1      10.55
     2       7.83
     3       9.50
     4      12.55
     5      11.11

>>opt.output = {{'f','y'}};
>>opt.symbol = {{'b-','r.'}};
>>plotmlx(res,'option',opt)
```



2.3.2 Simulating individual covariates

Model for the individual covariates is implemented in the `[COVARIATE]` section:

hierarchical2_model.txt

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

[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}
```

Individual covariates and individual parameters can be defined as outputs.

hierarchical2a_simulx.m

```
p = struct('name',{{'V_pop','omega_V','w_pop','omega_w','k','b'}},...
           'value',[10,0.2,70,12,0.2,0.2]);
f = struct('name','f','time',0:0.1:30);
y = struct('name','y','time',1:3:30);
ind = struct('name',{{'w','V'}});
out={ind,f,y};
res = simulx('model','hierarchical2_model.txt','parameter',p,'output',out);
```

Again, it is possible to simulate several individual using `group`. 5 individual are simulated in the next example.

`'level','covariate'` means that each individual has its own set of (simulated) individual covariates. Then, one vector of individual parameters (V in this example) and one vector of observations per individual are simulated.

```
>>g = struct('size',5,'level','covariate');
>>res = simulx('model','hierarchical2_model.txt','parameter',p,'output',out,'group',g);
>>displaymlx(res)

    id          w          V
     1      68.27       8.38
     2      58.56       7.32
     3      58.22       8.84
     4      65.85       8.55
     5      49.57       4.63
```

Remark: Using instead `'level','individual'` would mean that only one set of covariates (w in this example) is simulated. Then, the 5 individuals would share the same covariates.

2.3.3 Simulating population parameters

Model for the population parameters is implemented in the `[POPULATION]` section:

hierarchical3_model.txt

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

[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}

[POPULATION]
input = {ws, gw, Vs, gV}
DEFINITION:
w_pop = {distribution=normal, mean=ws, sd=gw}
V_pop = {distribution=logNormal, mean=log(Vs), sd=gV}
```

The `simulx` script now includes the values of the *hyperparameters* that define the distribution of the population parameters.

hierarchical3a_simulx.m

```
p = struct('name',{{'ws','gw','Vs','gV','omega_w','omega_V','k','b'}},...
           'value',[70,10,10,0.1,12,0.15,0.2,0.2]);
f = struct('name','f','time',0:0.1:30);
y = struct('name','y','time',1:3:30);
ind = struct('name',{{'w','V'}});
pop = struct('name',{{'w_pop','V_pop'}});
out={pop,ind,f,y};
res = simulx('model','hierarchical3_model.txt','parameter',p,'output',out);
```

We can now simulate 6 individuals with this model:

```
>>g = struct('size',[2,3],'level',{{'population','covariate'}});
>>res = simulx('model','hierarchical3_model.txt','parameter',p,'output',out,'group',g);
>>displaymlx(res)

    id      w_pop      V_pop          w          V
     1      62.54      10.48      62.99       9.55
     2      62.54      10.48      62.64      10.75
     3      62.54      10.48      51.96       8.21
     4      47.81       9.49      21.69       4.59
     5      47.81       9.49      58.32      11.18
     6      47.81       9.49      49.74       9.11
```

`'size',[2,3],'level',{{'population','covariate'}}` means that 2 populations and 3 set of individual covariates (w here) in each population are simulated. Then, one vector of individual parameters (V in this example) and one vector of observations per individual are simulated.

2.4 Pharmacokinetics Models

The demos used in this section are in `/demos/4_pk`

One of the most appealing property of `simulx` is its ability to easily encode complex pharmacokinetics (PK) models. We will consider here compartmental PK models: the human body is described by a series of compartments in which the drug is distributed.

The depot compartment is the site at which a drug is deposited: the gut for oral administration, skin for the application of dermal patches, the bloodstream for intravenous administration, etc. The central compartment consists of blood and highly perfused organs and peripheral compartments of less perfused tissue.

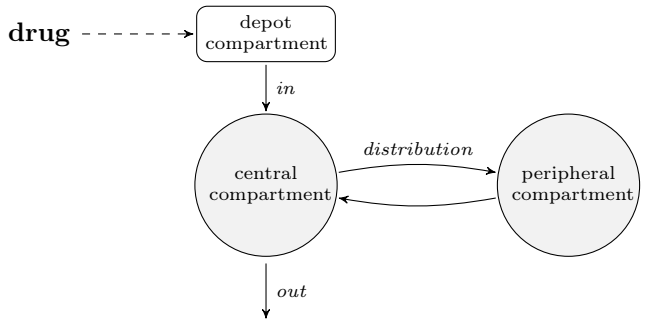
PK compartmental models assume that drug concentration is perfectly homogeneous in each compartment of the body at all times. Then, describing quantitatively a PK model turns into describing how the drug amount varies in each compartment. This means we typically need to describe three components:

. The *rate in* describes how the drug moves from the depot compartment to the central compartment.

. The *rate of distribution* describes exchanges of the drug between the central and peripheral compartments.

. The *rate out* describes how the drug is eliminated from the central compartment.

The final model will then bring together these three components into one model.



,

2.4.1 Representing PK models with dynamical systems

A PK model is a dynamical system mathematically represented by a system of ordinary differential equations (ODEs) which describe transfers between compartments and elimination from the central compartment. Let us look as several examples of dynamical systems for different types of administration.

Intravenous administration.

Consider first a system for iv administration with only one central compartment. The only process that needs to be described is elimination.

Linear elimination (or first-order elimination) means that the rate of elimination is directly proportional to the amount:

$$\dot{A}_c(t) = -k_e\, A_c(t), \tag{2.2}$$

where $A_c(t)$ and $\dot{A}_c(t)$ are respectively the drug amount in the central compartment and its derivative at time t. The proportionality constant k_e is the *elimination rate constant*.

Saturable elimination means that above a certain drug concentration, the elimination rate tends to reach a maximal value. Such capacity-limited elimination is best explained by the Michaelis-Menten equations:

$$\dot{A}_c(t) = -\frac{V_m}{V\,K_m + A_c(t)}\,A_c(t). \tag{2.3}$$

Let us next introduce a peripheral compartment to the model and assume linear transfer between the central compartment and it. Assuming linear elimination from the central compartment, the mathematical representation of this model now consists of a system of two ODEs:

$$\begin{aligned}\dot{A}_c(t) &= k_{21}A_p(t) - k_{12}A_c(t) - k_eA_c(t)\\ \dot{A}_p(t) &= -k_{21}A_p(t) + k_{12}A_c(t),\end{aligned} \tag{2.4}$$

where A_p is the amount in the peripheral compartment and k_{12} and k_{21} the distribution rate constants.

A three-compartment model is characterized by three ODEs:

$$\begin{aligned}\dot{A}_c(t) &= k_{21}A_p(t) + k_{31}A_q(t) - (k_{12} + k_{13} + k_e)A_c(t)\\ \dot{A}_p(t) &= -k_{21}A_p(t) + k_{12}A_c(t)\\ \dot{A}_q(t) &= -k_{31}A_q(t) + k_{13}A_c(t).\end{aligned}$$

Oral administration.

A *zero-order absorption process* assumes that a drug is transferred from the depot compartment with constant rate R_0:

$$\dot{A}_d(t) = -R_0\mathbb{1}_{A_d(t)>0} \tag{2.5}$$

where $\mathbb{1}_U = 1$ if U is true and $\mathbb{1}_U = 0$ otherwise.

A *first-order absorption process* assumes that the absorption rate is proportional to the drug amount in the depot compartment:

$$\dot{A}_d(t) = -k_a\,A_d(t). \tag{2.6}$$

PK models for oral administration consist of a model each for absorption, distribution and elimination. For example, a one-compartment model with a first-order absorption process and linear elimination combines (2.6) and (2.2):

$$\begin{aligned}\dot{A}_d(t) &= -k_aA_d(t)\\ \dot{A}_c(t) &= k_aA_d(t) - k_eA_c(t).\end{aligned}$$

A two-compartment model with a zero-order absorption process and nonlinear elimination combines (2.5), (2.4) and (2.3):

$$\begin{aligned}\dot{A}_d(t) &= -R_0\mathbb{1}_{A_d(t)>0}\\ \dot{A}_c(t) &= R_0\mathbb{1}_{A_d(t)>0} + k_{21}A_p(t) - \left(k_{12} + \frac{V_m}{V\,K_m + A_c(t)}\right)A_c(t)\\ \dot{A}_p(t) &= -k_{21}A_p(t) + k_{12}A_c(t).\end{aligned}$$

2.4.2 Implementing PK models with MLXTRAN

There are three main ways to implement PK models using MLXTRAN: first, the `pkmodel` function for defining standard PK models; second, PK macros for complex administrations and/or complex PK models; and third, equations for implementing any dynamical system. These options offer a lot of flexibility. Furthermore, the same implementation can be used for performing several tasks such as modeling and simulation.

1) Equations can be used if we wish to represent a dynamical system by a system of ODEs and "deposit" the drug in any compartment, i.e., define administrations as source terms for any component of the system of ODEs.

EXAMPLE 4.1 One compartment model for iv administration with linear elimination. This PK model can be implemented with MLXTRAN using equations:

pk1_model.txt

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

PK:
depot(target=Ac)

EQUATION:
ddt_Ac = -k*Ac
Cc=A/V
```

`pk1_model` is now "ready" to receive iv bolus or iv infusions.

EXAMPLE 4.2 Combination of oral and iv administrations. Only a fraction F of the oral dose is absorbed. Inputs of type 1 (`type=1`) are oral administrations and inputs of type 2 (`type=2`) are iv administrations.

pk2_model.txt

```
[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
```

`pk2_model` is now "ready" to receive any combination of oral and iv doses.

2) `pkmodel` should be used when there is only one type of administration. It plays the role of PK model library; a model is selected by the choice of function arguments.

EXAMPLE 4.3 One-compartment model for oral administration with lag-time, zero-order absorption and linear elimination. The model parameters are bioavailability F, lag-time $Tlag$, duration of absorption $Tk0$, volume of distribution V and clearance Cl.

pk3_model.txt

```
[LONGITUDINAL]
input = {F, Tlag, Tk0, V, Cl}

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

Each of p, $Tlag$, $Tk0$, V and Cl are reserved keywords automatically recognized by `pkmodel`; p is the fraction of the dose which is absorbed.

EXAMPLE 4.4 Two-compartment model for iv administration, nonlinear elimination, including an effect compartment. The model parameters are distribution rate constants k_{12} and k_{21}, volume of distribution V, Michaelis-Menten parameters V_m and K_m, and transfer rate constant ke_0.

pk4_model.txt

```
[LONGITUDINAL]
input = {k12, k21, V, Vm, Km, ke0}

EQUATION:
{Cc, Ce} = pkmodel(k12, k21, V, Vm, Km, ke0)
```

See Section **??** how to use the MATLAB function `pkmodel` which does not require any MLXTRAN script.

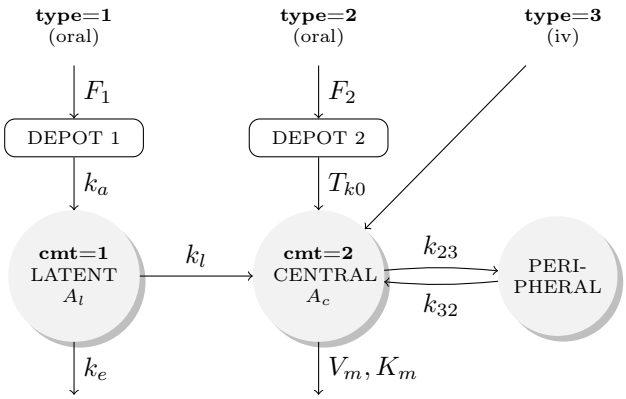
3) PK macros can be used to define the components of a compartmental model (compartments, absorption, distribution, elimination, etc.).

EXAMPLE 4.5 Sequential zero-order/first-order absorption process. A fraction F_0 of the dose is first absorbed with a zero-order absorption process, then the remaining $1 - F_0$ is absorbed with a first-order absorption process. The drug is eliminated from the central compartment by a linear process.

pk5_model.txt

```
[LONGITUDINAL]
input = {F0, Tk0, ka, V, Cl}

PK:
compartment(cmt=1, amount=Ac)
oral(cmt=1, Tk0, p=F0)
oral(cmt=1, ka, Tlag=Tk0, p=1-F0)
elimination(cmt=1, Cl)
Cc = Ac/V
```

EXAMPLE 4.6 Multiple administrations and multiple absorption processes. In this example, one type of dose is administered orally (`type=1`) and absorbed into a latent compartment following a first-order absorption process, a second type is administered orally (`type=2`) and absorbed into the central compartment following a zero-order absorption process, and a third type is directly administered intravenously to the central compartment (`type=3`). The transfer from the latent to the central compartment is linear. A peripheral compartment is linked to the central compartment. The drug is eliminated by a linear process from the latent compartment and a nonlinear process from the central one. Here, A_l and A_c are the drug amounts in the latent and central compartments.



pk6_model.txt

```
[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
```

Remark: It is possible with MLXTRAN to combine equations and PK macros.

See the MLXTRAN documentation for more information about the use of MLXTRAN for implementing PK models:

2.4.3 Putting doses into a system

PK models do not describe how a drug is administered. Administered doses are *source terms*, i.e., *inputs* that dynamically modify the state of a system. Then, the same PK model can be used for different dose regimens.

The plasmatic concentration predicted by a model is the solution of a system of ODEs for a given series of inputs and initial conditions (we suppose in general that all compartments are empty before the first dose).

In the case of iv bolus, a drug is administered intravenously over a negligible period of time and achieves instantaneous distribution throughout the central compartment. Let D be the drug amount administered at time τ. Then,

$$A_c(\tau^+) = A_c(\tau^-) + D,$$

where τ^- and τ^+ are the times just before and after drug administration. If the compartment is empty before τ, the solution is:

$$C_c(t) = \frac{D}{V}\, e^{-k_e\,(t-\tau)}\, \mathbb{1}_{t>\tau}.$$

If K doses $D_1, D_2, \ldots, D_K$ are administered at times $\tau_1, \tau_2, \ldots, \tau_K$, a superposition principle

applies since the system is linear:

$$C_c(t) = \frac{1}{V}\sum_{k=1}^{K} D_k e^{-k_e\,(t-\tau_k)}\,\mathbb{1}_{t>\tau_k}.$$

In the case of iv infusion, a dose is administered with a constant rate $R_{\rm inf}$ during a certain infusion time period of length $T_{\rm inf} = D/R_{\rm inf}$. Assuming a unique dose is administered at time τ, the solution is:

$$C_c(t) = \begin{cases} 0 & \text{if } t < \tau \\ R/(V\,k_e)(1 - e^{-k_e\,(t-\tau)}) & \text{if } \tau \le t < \tau + T_{\rm inf} \\ R/(V\,k_e)(1 - e^{-k_e\,T_{\rm inf}})e^{-k_e\,(t-\tau-T_{\rm inf})} & \text{if } t \ge \tau + T_{\rm inf} \end{cases}$$

Note again that the same model was used for all these different examples; it is the changing inputs – here doses – which generate different solutions.

Consider now oral administration and let A_d be the drug amount in the depot compartment. If an amount D is instantaneously deposited at time τ (imagine for instance that the dose is swallowed all at once), then, denoting F the bioavailability, state variable A_d is modified at time τ:

$$A_d(\tau^+) = A_d(\tau^-) + F\,D.$$

If the central and depot compartments are empty before τ, the solution is:

$$C_c(t) = \frac{F\,D\,k_a}{V(k_a - k_e)}\left(e^{-k_e\,(t-\tau)} - e^{-k_a\,(t-\tau)}\right)\mathbb{1}_{t>\tau}.$$

2.4.4 Using `simulx` for evaluating PK models

In order to evaluate these types of models, we need inputs, i.e., a dose regimen. For each dose being administered, we need specific information:

- the administration time
- the drug amount administered
- the rate (or duration) of infusion if this is the type of administration
- the administration types if there are several of them.

Amounts and times must always be provided while rates and types are needed only in certain cases.

Single type of administration

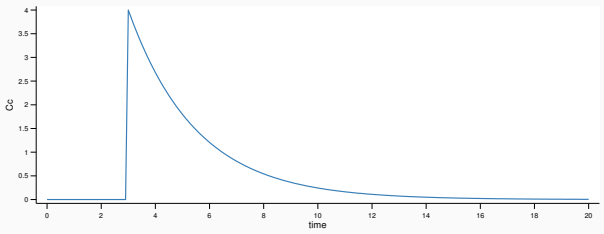
Let us start with the PK model `pk1_model.txt` of Example 2.42.1 for iv administration.

- Here `type` is not required since there is only one type of administration.
- If the administration is an iv bolus, `rate` is not required either.

Consider first a unique administration of 40 units at time 0:

pk1_simulx.m

```
adm = struct('time',3, 'amount',40);
Cc = struct('name','Cc','time',0:0.1:20);
p = struct('name',{{'V','k'}},'value',[10,0.4]);
res = simulx('model','pk1_model.txt','parameter',p,'output',Cc,'treatment',adm);

plotmlx(res)
```



If we consider multiple administrations, a vector of times is required:

```
adm = struct('time',[6,18,30,42], 'amount',40)
```

or, equivalently

```
adm = struct('time',[6:12:42], 'amount',40)
```

Different amounts can be administered

```
adm = struct('time',[6,18,30,42], 'amount',[40,30,40,50])
```

`rate` is required for an input with constant rate (e.g. iv infusion)

```
adm = struct('time',[6,18,30,42], 'amount',40, 'rate',10)
```

An iv bolus is an infusion with an infinite rate

```
adm = struct('time',[6,18,30,42], 'amount',40, 'rate',Inf)
```

`rate` and/or `amount` can be vectors

```
adm = struct('time',[6,18,30,42], 'amount',40, 'rate',[10,20,10,20])
```

Multiple types of administration

We will use now the PK model `pk2_model.txt` of Example 2.42.2 for iv and oral administrations.

Here `type` is required since there are two possible types of administration. According to model `pk2_model.txt`, we use `type=1` for oral administration

```
adm = struct('time',[6,18,30,42], 'amount',40, 'type',1)
```

and `type=2` for iv administration

```
adm = struct('time',[6,18,30,42], 'amount',40, 'type',2)
```

A constant rate can be used for any type of administration, including oral administration

```
adm = struct('time',[6,18,30,42], 'amount',40, 'rate',10, 'type',1)
```

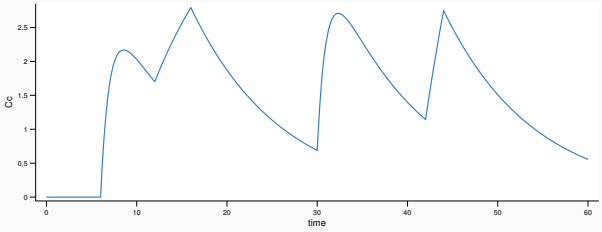
It is possible to combine different types of administration in a single list

```
adm = struct('time',[6,18,30,42], 'amount',[40,20,40,20],...
             'rate',[Inf,5,Inf,10], 'type',[1,2,1,2])
```

It is possible to define equivalently several types of administrations in several lists, and a treatment as a list of administrations:

pk2a_simulx.m

```
adm1 = struct('time',[6 30],'amount',40,'type',1);
adm2 = struct('time',[12 42],'amount',20,'type',2,'rate',[5, 10]);
trt = {adm1, adm2};
Cc = struct('name','Cc','time',0:0.1:60);
p = struct('name',{{'F','ka','V','k'}},'value',[0.7,1,10,0.1]);
res = simulx('model','pk2_model.txt','parameter',p,'output',Cc,'treatment',trt);

plotmlx(res)
```



It is also possible to define the target compartments (i.e. the targets of the source terms) directly in the MATLAB script instead of the model. That can be useful for testing different different administration schemes for example, without modifying the model which only represents the dynamical system:

pk2b_model.txt

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

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

pk2b_simulx.m

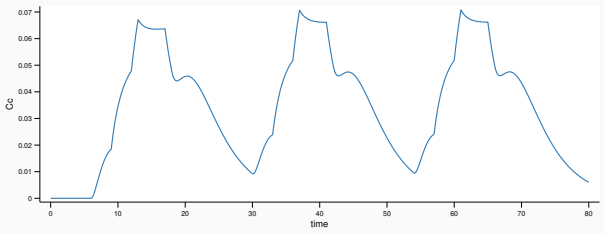
```
adm1 = struct('time',[6 30],'amount',40,'target','Ad');
adm2 = struct('time',[12 42],'amount',20,'target','Ac','rate',[5, 10]);
trt = {adm1, adm2};
Cc = struct('name','Cc','time',0:0.1:60);
p = struct('name',{{'ka','V','k'}},'value',[1,10,0.1]);
res = simulx('model','pk2b_model.txt','parameter',p,'output',Cc,'treatment',trt);
```

Remark: It is not possible using this approach to define the bioavailability (i.e. the fraction of the dose which is absorbed) which is assumed to be equal to 1.

`simulx` is therefore extremely flexible for defining complex dose regimens. Let's go back to Example 2.42.6 where three types of administrations were defined in model `pk6\_model.txt`. We can imagine a treatment that combine these three types of administration:

pk6a_simulx.M

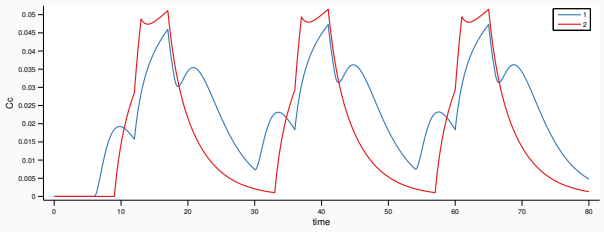
```
adm1 = struct('time',6:12:66,'amount',2,'type',1);
adm2 = struct('time',9:24:57,'amount',1,'type',2);
adm3 = struct('time',12:24:60,'amount',1,'rate',0.2,'type',3);
trt={adm1,adm2,adm3};
Cc = struct('name','Cc','time',0:0.1:80);
p.name =  {'F1','F2','ka','Tk0','kl','k23','k32','V','k','Vm','Km'};
p.value = [0.5, 0.8, 0.5, 4, 0.5, 0.3, 0.5, 10, 0.2, 0.5, 1];
res=simulx('model','pk6_model.txt','treatment',trt,'parameter',p,'output',Cc);

plotmlx(res)
```



We can also imagine to define two treatments by combining one type of oral administration with the iv administration:

pk6b_simulx.m

```
>>g1 = struct('treatment',{{adm1, adm3}});
>>g2 = struct('treatment',{{adm2, adm3}});
>>res=simulx('model','pk6_model.txt','group',{g1,g2},'parameter',p,'output',Cc);
>>plotmlx(res)
```



Remark: It is also possible to use `mlxplore` for exploring the PK model. The same input arguments can be used with the function `mlxplore`.

```
>>mlxplore('model','pk6_model.txt','treatment',trt,'parameter',p,'output',Cc)
```

Or,

```
>>mlxplore('model','pk6_model.txt','group',{g1,g2},'parameter',p,'output',Cc)
```

2.5 Simulation of several individuals and groups

The demos used in this section are in `/demos/5_groups`

2.5.1 Defining groups

Group size

When there is only one group, `group` is a list with 2 arguments:

`size` defines the size of the group. It is a scalar or a vector if the model has different sections

`level` defines the levels of randomization. It is required when the model has several sections, and hence several possible levels of randomization.

EXAMPLE 5.1 Simulation of several populations replicates of the longitudinal data.

group1_model.txt

```
[LONGITUDINAL]
input = {V, k, a}
EQUATION:
Cc = pkmodel(V,k)
DEFINITION:
y = {distribution=normal, prediction=Cc, sd=a}
```

One single dose of 100 units is given to 5 subjects. `simulx` script for simulating $(y_{ij}, 1 \leq i \leq 5, 1 \leq j \leq 11)$ at times $t_{ij} = 0, 1, 2, \ldots, 10$, with $V = 10$, $k = 0.2$ and $a = 0.5$:

group1_simulx.m

```
adm = struct('time',0,'amount',100);
y = struct('name','y','time',0:1:10);
p = struct('name',{{'V','k','a'}},'value',[10,0.2,0.5]);
g = struct('size',5);
res = simulx('model','group1_model.txt','parameter',p,'output',y,...
             'treatment',adm,'group',g);
```

EXAMPLE 5.2 Simulation of several individual parameters and replicates of the longitudinal data.

group2_model.txt

```
[LONGITUDINAL]
input = {V, k, a}
EQUATION:
Cc = pkmodel(V,k)
DEFINITION:
y = {distribution=normal, prediction=Cc, sd=a}

[INDIVIDUAL]
input = {V_pop, omega_V, w}
EQUATION:
Vpred=V_pop*(w/70)
DEFINITION:
V = {distribution=lognormal, prediction=Vpred, sd=omega_V}
```

In this example, we want to simulate 2 individuals and 3 replicates of the longitudinal data per individual.

group2_simulx.m

```
adm = struct('time',0,'amount',100);
p = struct('name',{{'V_pop','omega_V','w','k','a'}},'value',[10,0.3,75,0.2,0.5]);
y = struct('name','y','time',0:1:10);
op= struct('name',{'V'});
g = struct('size',[2,3],'level',{{'individual','longitudinal'}});
res = simulx('model','group2_model.txt','parameter',p,'output',{y,op},...
             'treatment',adm,'group',g);
displaymlx(res)
```

```
    id         V
     1      8.70
     2      8.70
     3      8.70
     4      7.13
     5      7.13
     6      7.13
```

Any combination is then possible with the same model. We can for example simulate 10 individuals and only one replicate of data per individual

```
g = struct('size',10,'level','individual');
```

or only one individual and 10 replicates for this individual

```
g = struct('size',10,'level','longitudinal');
```

EXAMPLE 5.3 Simulation of several populations, individual covariates, individual parameters and replicates of the longitudinal data.

Suppose now that we have defined in the model a submodel for the covariates and a submodel for the population parameters:

group3_model.txt

```
[LONGITUDINAL]
input = {V, k, a}
EQUATION:
Cc = pkmodel(V,k)
DEFINITION:
y = {distribution=normal, prediction=Cc, sd=a}

[INDIVIDUAL]
input = {V_pop, omega_V, w}
EQUATION:
Vpred=V_pop*(w/70)
DEFINITION:
V = {distribution=lognormal, prediction=Vpred, sd=omega_V}

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

[POPULATION]
input = {ks, Vs, gk, gV}
DEFINITION:
k = {distribution=logitnormal, reference=ks, sd=gk}
V_pop = {distribution=normal, mean=Vs, sd=gV}
```

We can for example simulate 2 populations and 3 individual covariates $w_{\ell i}$ per population. In this example, only one vector of individual parameters and one replicate of longitudinal data are simulated per individual.

group3_simulx.m

```
adm = struct('time',0,'amount',100);
p = struct('name',{{'Vs','gV','ks','gk','omega_V','w_pop','omega_w','a'}},...
           'value',[10,1,0.2,0.3,0.3,70,10,0.5]);
op = struct('name',{{'V_pop','k','w','V'}});
y = struct('name','y','time',0:1:10);
g = struct('size',[2,3],'level',{{'population','covariate'}});

res = simulx('model','group3_model.txt','parameter',p,'output',{op,y},...
             'treatment',adm,'group',g);

displaymlx(res)
```

```
    id     V_pop         k         w         V
     1     10.48      0.25     66.75      7.09
     2     10.48      0.25     76.59     15.58
     3     10.48      0.25     52.94      5.09
     4      9.68      0.20     63.91      9.90
     5      9.68      0.20     65.19     11.62
     6      9.68      0.20     72.04      9.79
```

We can then imagine any combination of the four levels of randomization and simulate for instance a total of 200 ($2 \times 10 \times 2 \times 5$) series of longitudinal data:

```
>> g = struct('size', [2,10,2,5],...
              'level',{{'population','covariate','individual','longitudinal'}});
```

Different dose regimens per group

group4_model.txt

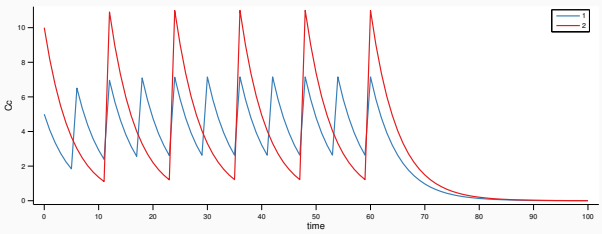
```
[LONGITUDINAL]
input = {V, k}
EQUATION:
Cc = pkmodel(V,k)
```

Groups can share the same PK model, but with different dose regimens:

EXAMPLE 5.4

group4_simulx.m

```
adm1 = struct('time',0:6:60,'amount',50);
adm2 = struct('time',0:12:60,'amount',100);
Cc = struct('name','Cc','time',0:1:100);
p = struct('name',{{'V','k'}},'value',[10,0.2]);
g1 = struct('treatment',adm1);
g2 = struct('treatment',adm2);
g= {g1, g2};
res = simulx('model','group4_model.txt','parameter',p,'output',Cc,'group',g);

plotmlx(res)
```



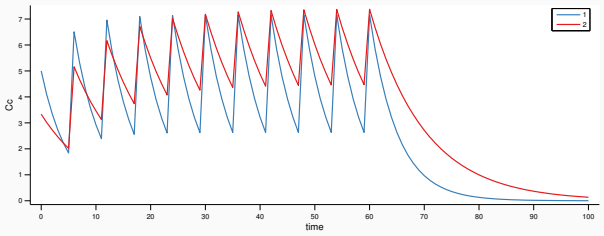
Different parameter values per group

We can also use different set of parameters per group:

EXAMPLE 5.5

group5_simulx.m

```
adm = struct('time',0:6:60,'amount',50);
Cc = struct('name','Cc','time',0:1:100);
p1 = struct('name',{{'V','k'}},'value',[10,0.2]);
p2 = struct('name',{{'V','k'}},'value',[15,0.1]);
g1 = struct('parameter',p1);
g2 = struct('parameter',p2);
g= {g1, g2};
res = simulx('model','group4_model.txt','treatment',adm,'output',Cc,'group',g);

plotmlx(res)
```



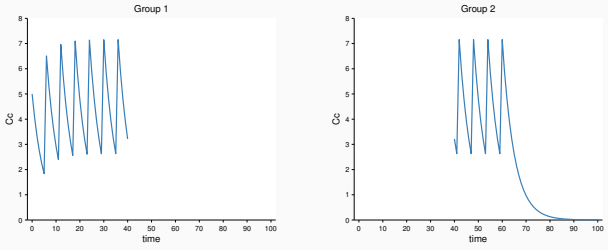
Different outputs per group

Each group can have its own set of outputs:

EXAMPLE 5.6

group6_simulx.m

```
adm = struct('time',0:6:60,'amount',50);
Cc1 = struct('name','Cc','time',0:1:40);
Cc2 = struct('name','Cc','time',40:1:100);
p = struct('name',{{'V','k'}},'value',[10,0.2]);
g1 = struct('output',Cc1);
g2 = struct('output',Cc2);
g= {g1, g2};
res = simulx('model','group4_model.txt','parameter',p,'treatment',adm,'group',g);

plotmlx(res)
```



Miscellaneous

Any combination is then possible. In the following example, the two groups have different sizes, different treatments, different sets of parameters and different outputs. Note that these

fields `treatment`, `parameter` and `output` do not need anymore to be used as arguments of the `simulx` function since they are already used for defining the groups.

EXAMPLE 5.7

group7_simulx.m

```
adm1 = struct('time',0:6:60,'amount',50);
adm2 = struct('time',0:12:60,'amount',100);
y1 = struct('name','y','time',30:2:100);
y2 = struct('name','y','time',0:2:70);
p1 = struct('name',{{'V_pop','omega_V','w','k','a'}},...
            'value',[10,0.3,50,0.2,0.5]);
p2 = struct('name',{{'V_pop','omega_V','w','k','a'}},...
            'value',[15,0.3,75,0.1,0.5]);
g = cell(1,2);
g{1} = struct('size',3,'level','individual','parameter',p1,...
              'administration',adm1,'output',y1);
g{2} = struct('size',2,'level','individual','parameter',p2,...
              'administration',adm2,'output',y2);

res = simulx('model','group2_model.txt','group',g,'record','essai.txt');
```

2.5.2 Using individual data

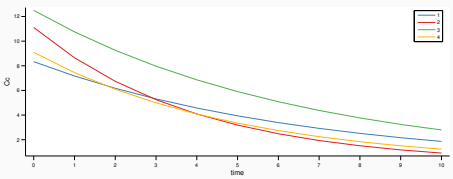
Different parameters values per subject

Individual data can be used as input. Individual parameters are used in this first example.

EXAMPLE 5.8

data1_simulx.m

```
adm = struct('time',0,'amount',100);
Cc = struct('name','Cc','time',0:1:10);
p = struct('name',{{'V','k'}},'header',{{'id','V','k'}});
p.value = [
     1    12   0.15
     2     9   0.25
     3     8   0.15
     4    11   0.20];

res = simulx('model','group4_model.txt','parameter',p,'output',Cc,'treatment',adm);

plotmlx(res)
```



It is also possible to combine individual parameters (V in this example) and parameters which are shared by all the individuals (k here):

EXAMPLE 5.9

data2_simulx.m

```
adm = struct('time',0,'amount',100);
Cc = struct('name','Cc','time',0:1:10);
p=cell(1,2);
p{1} = struct('name','V','header',{{'id','V'}});
p{1}.value = [
     1    12
     2     9
     3     8
     4    11];
p{2} = struct('name','k','value',0.2);

res = simulx('model','group4_model.txt','parameter',p,'output',Cc,'treatment',adm);
```

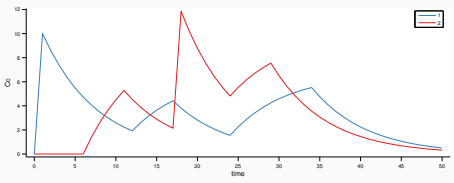
Different dose regimens per subject

If the dose regimen is defined in a data file, it should contain columns `time` and `amt` and possibly `rate` and `type`. Each row contains the complete set of information for a certain dose.

Example 5.10

data3_simulx.m

```
adm = struct('header',{{'id','time','amount', 'rate'}});
adm.value = [
     1     1   100  Inf
     1    12    50  10
     1    24   100  10
     2     6    75  15
     2    18   100  Inf
     2    24    75  15 ];
Cc = struct('name','Cc','time',0:1:50);
p = struct('name',{{'V','k'}},'value',[10,0.15]);

res = simulx('model','group4_model.txt','parameter',p,'output',Cc,'treatment',adm);

plotmlx(res)
```



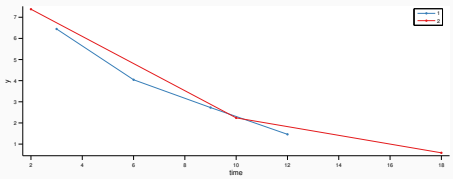
Different output designs per subject

Individual designs for the outputs can also be defined. Here, different observation times per individual are used.

Example 5.11

data4_simulx.m

```
adm = struct('time',0,'amount',100);
y = struct('name','y','header',{{'id','time'}});
y.value = [
      1     3
      1     6
      1     9
      1    12
      2     2
      2    10
      2    18  ];
p = struct('name',{{'V','k','a'}},'value',[10,0.15,0.3]);

res = simulx('model','group1_model.txt','parameter',p,'output',y,'treatment',adm);

opt=struct('symbol','.-','output','y');
plotmlx(res,'option',opt)
```



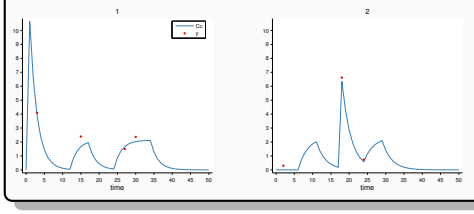
Miscellaneous

It is possible to combine all these features. Note that any of these individual components (treatment, parameters, covariates, outputs) can easily be loaded from one or various datafiles. A modular representation of the data (i.e. where different informations such as dose history, covariates, observations are stored in different sections of the datafile) is then much more efficient than a unique table that contains all these informations.

Example 5.12

data5_simulx.m

```
adm = struct('header',{{'id','time','amount', 'rate'}});
adm.value = [
     1     1   100  Inf
     1    12    50  10
     1    24   100  10
     2     6    75  15
     2    18   100  Inf
     2    24    75  15 ];
out{1} = struct('name','Cc','time',0:1:50);
out{2} = struct('name','y','header',{{'id','time'}});
out{2}.value = [
     1     3
     1    15
     1    27
     1    30
     2     2
     2    18
     2    24  ];
p{1} = struct('name',{{'V_pop','omega_V','a'}},'value',[10,0.3,0.5]);
p{2} = struct('name',{{'w','k'}},'header',{{'id','w','k'}});
p{2}.value = [
     1    75  0.5
     2    60  0.4];

res = simulx('model','group2_model.txt','parameter',p,'output',out,'treatment',adm);

opt.output = {{'Cc','y'}};
opt.symbol = {{'-','.'}};
opt.split = 'individual';
plotmlx(res,'option',opt)
```



2.6 Examples

The demos used in this section are in `/demos/6_examples`

2.6.1 PKPD model and Clinical trial

We consider in this first example a 1 one compartment PK model (zero order absorption and linear elimination) that we combine with three different PD models (immediate response, effect compartment and indirect response models).

You can run `pkpd1_simulx.m` or `pkpd1_mlxplore.m` to visualize the structural model implemented in `pkpd1\_model.txt`

Let us see in more detail the implementation and the use of the statistical model. All the parameters are log-normally distributed except $Imax$ who takes its value between 0 and 1 and who is logit-normally distributed.

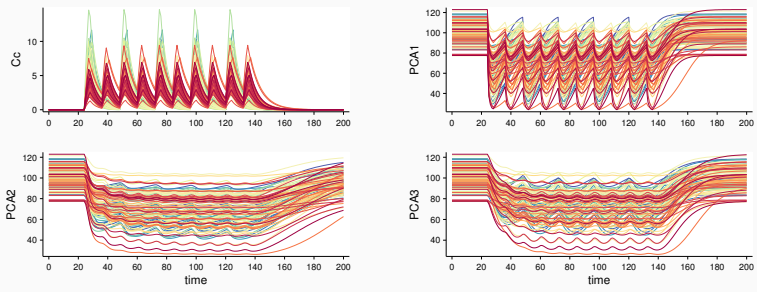
pkpd2_model.txt

```
[LONGITUDINAL]
input={Tk0,V,k,ke0,Imax,S0,IC50,kout,a1,a2,a3,a4}

EQUATION:
{Cc, Ce}  = pkmodel(Tk0, V, k, ke0)
Ec = Imax*Cc/(Cc+IC50)
PCA1 = S0*(1 - Ec)
Ee = Imax*Ce/(Ce+IC50)
PCA2 = S0*(1 - Ee)
PCA3_0 = S0
ddt_PCA3 = kout*((1-Ec)*S0- PCA3)

DEFINITION:
y1 ={distribution=lognormal, prediction=Cc,   sd=a1}
y2 ={distribution=normal,    prediction=PCA1, sd=a2}
y3 ={distribution=normal,    prediction=PCA2, sd=a3}
y4 ={distribution=normal,    prediction=PCA3, sd=a4}

;-----------------------------------------------------
[INDIVIDUAL]
input={Tk0_pop,omega_Tk0,V_pop,omega_V,beta_V,k_pop,omega_k,beta_k,
       ke0_pop,omega_ke0,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:
Tk0 ={distribution=lognormal, prediction=Tk0_pop,  sd=omega_Tk0}
V   ={distribution=lognormal, prediction=V_pred,   sd=omega_V}
k   ={distribution=lognormal, prediction=k_pred,   sd=omega_k}
ke0 ={distribution=lognormal, prediction=ke0_pop,  sd=omega_ke0}
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 use `simulx` in this example to simulate a clinical trial with two arms and 20 patients in each arm.

pkpd2_simulx.m

```
adm1 = struct('amount',100,'time',24:24:143);
adm2 = struct('amount',50,'time',24:12:143);

p.name = {'w_pop','omega_w','Tk0_pop','omega_Tk0','V_pop','omega_V','beta_V',...
          'k_pop','omega_k','beta_k','ke0_pop','omega_ke0','Imax_pop',...
          'omega_Imax','S0_pop','omega_S0','IC50_pop','omega_IC50',...
          'kout_pop','omega_kout','a1','a2','a3','a4'};
p.value = [70, 10, 4, 0.2, 10, 0.3, 1, 0.2, 0.2, -0.25, 0.05, 0.1, 0.5, 1, ...
           100, 0.1, 1, 0.2, 0.1, 0.1, 0.1, 2, 2, 2];

ind = struct('name',{{'w','Tk0','V','k','ke0','Imax','S0','IC50','kout'}});
f = struct('name',{{'Cc','PCA1','PCA2','PCA3'}},'time',0:1:200);
y = struct('name',{{'y1','y2','y3','y4'}},'time',1:3:200);
out={ind,f,y};

g1 = struct('size',50,'level','covariate', 'treatment',adm1);
g2 = struct('size',50,'level','covariate', 'treatment',adm2);
g = {g1, g2};

res = simulx('model','pkpd2_model.txt','parameter',p,'output',out,'group',g);

opt.output = {'Cc','PCA1','PCA2','PCA3'};
opt.legend = 'off';
stg = struct('linewidth',0.5);
plotmlx(res,'option',opt,'setting',stg)
```



2.6.2 TGI model

We use here the tumor growth inhibition model proposed in Ribba et al. (2012). This model describes tumor size evolution in patients treated with chemotherapy:

A tumor is considered to be composed of proliferative cells (PT) and nonproliferative quiescent cells (Q). Treatment directly eliminates proliferative cells. Quiescent cells can also be affected by treatment and become damaged quiescent cells (QP). The size of the tumor is therefore proportional to the total number of cells PSTAR=PT+Q+QP

The chemotherapy pharmacokinetics are modeled using a kinetic-pharmacodynamic approach. C represents the concentration of a virtual drug encompassing the various chemotherapeutic components of the treatment.

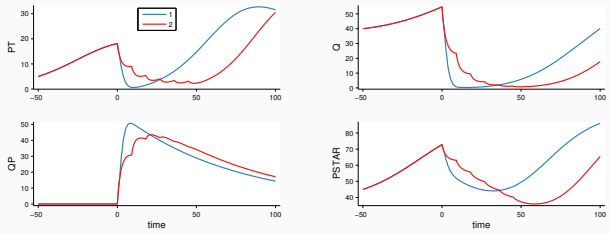
tgi1_model.txt

```
[LONGITUDINAL]
input={K, KDE, KPQ, KQPP, LAMBDAP, GAMMA, DELTAQP, a}

EQUATION:
PT_0=5
Q_0 = 40
PSTAR = PT+Q+QP
ddt_C = -KDE*C
ddt_PT = LAMBDAP*PT*(1-PSTAR/K) + KQPP*QP - KPQ*PT - GAMMA*KDE*PT*C
ddt_Q = KPQ*PT - GAMMA*KDE*Q*C
ddt_QP = GAMMA*KDE*Q*C - KQPP*QP - DELTAQP*QP
```

simulx can be used for computing the size of the tumor $P^\star = PT+Q+QP$ and its different components of a patient before, under and after treatment. We compare in this examples two treatments (1 unit every 12 months and 0.25 unit every 3 months):

tgi1_simulx.m

```
adm1 = struct('amount',1,'time',0:1.5:7.5,'target','C');
adm2 = struct('amount',1,'time',0:9:45,'target','C');
f = struct('name',{{'PT','Q','QP','PSTAR'}},'time',-50:0.5:100);
psi.name={'K','KDE','KPQ','KQPP','LAMBDAP','GAMMA','DELTAQP'};
psi.value=[100, 0.3, 0.025, 0.004, 0.12, 1, 0.01];
g1 = struct('treatment',adm1);
g2 = struct('treatment',adm2);
g = {g1, g2};
res=simulx('model','tgi1_model.txt','parameter',psi,'group',g,'output',f);

plotmlx(res)
```



A statistical model can now be introduced:

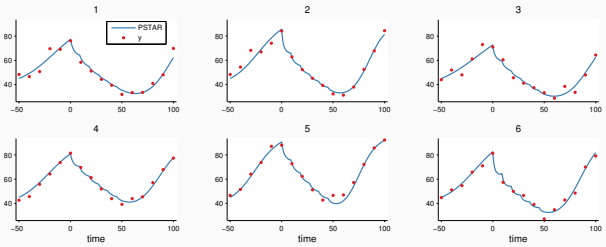
tgi2_model.txt

```
[LONGITUDINAL]
input={K, KDE, KPQ, KQPP, LAMBDAP, GAMMA, DELTAQP, a}

EQUATION:
PT_0=5
Q_0 = 40
PSTAR = PT+Q+QP
ddt_C = -KDE*C
ddt_PT = LAMBDAP*PT*(1-PSTAR/K) + KQPP*QP - KPQ*PT - GAMMA*KDE*PT*C
ddt_Q = KPQ*PT - GAMMA*KDE*Q*C
ddt_QP = GAMMA*KDE*Q*C - KQPP*QP - DELTAQP*QP

DEFINITION:
y ={distribution=normal, prediction=PSTAR, sd=a}

;----------------------------------------------------
[INDIVIDUAL]
input={KDE_pop,omega_KDE,KPQ_pop,omega_KPQ,KQPP_pop,omega_KQPP,LAMBDAP_pop,
       omega_LAMBDAP,GAMMA_pop,omega_GAMMA,DELTAQP_pop,omega_DELTAQP}

DEFINITION:
KDE     = {distribution=lognormal, prediction=KDE_pop,     sd=omega_KDE}
KPQ     = {distribution=lognormal, prediction=KPQ_pop,     sd=omega_KPQ}
KQPP    = {distribution=lognormal, prediction=KQPP_pop,    sd=omega_KQPP}
LAMBDAP = {distribution=lognormal, prediction=LAMBDAP_pop,sd=omega_LAMBDAP}
GAMMA   = {distribution=lognormal, prediction=GAMMA_pop,   sd=omega_GAMMA}
DELTAQP = {distribution=lognormal, prediction=DELTAQP_pop,sd=omega_DELTAQP}
```

simulx is used in this example for simulating the size of the tumor 6 patients:

tgi2_simulx.m

```
adm = struct('amount',1,'time',0:9:45,'target','C');
f = struct('name','PSTAR','time',-50:0.5:100);
y = struct('name','y','time',-50:10:100);
out = {f, y};
psi.name={'K','KDE_pop','omega_KDE','KPQ_pop','omega_KPQ','KQPP_pop',...
          'omega_KQPP','LAMBDAP_pop','omega_LAMBDAP','GAMMA_pop',...
          'omega_GAMMA','DELTAQP_pop','omega_DELTAQP','a'};
psi.value=[100,0.3,0.2,0.025,0.2,0.004,0.2,0.12,0.2,1,0.2,0.01,0.2,3];
g = struct('treatment',adm,'size',6, 'level','individual');
res=simulx('model','tgi2_model.txt','parameter',psi,'group',g,'output',out);

opt.output = {{'PSTAR','y'}}; % PSTAR and y on the same plots
opt.symbol = {{'-','.'}};
opt.split = 'individual'; % 1 individual per subplot
plotmlx(res,'option',opt)
```



2.6.3 Epidemic model

We consider here the SEIR epidemic model of Genik and Van Den Driessche (1999, example 4.4.1), where the total population N is decomposed into four states: S, susceptible but not infective; E, exposed but not infective; I, infective; R, recovered.

There are two time delays in the model: a temporary immunity delay τ and a latency delay ω. Then, the model is mathematically represented with a system of delayed differential equations (DDE).

DDEs can be implemented with MLXTRAN using the reserved keyword `delay`:

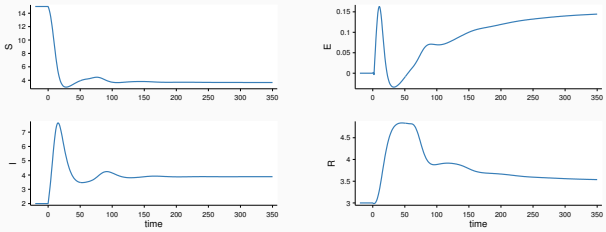
seir_model.txt

```
[LONGITUDINAL]
input = {A,lambda,gamma,epsilon,d,tau,omega}

EQUATION:
t0 = 0
S_0 =15
E_0 =0
I_0 =2
R_0 =3

N = S+E+I+R

ddt_S = A - d*S -lambda*S*I/N
+gamma*delay(I,tau)*exp(-d*tau)

ddt_E =lambda*S*I/N -d*E
 -lambda*delay(S,omega)*delay(I,omega)*exp(-d*omega)
 /(delay(I,omega)+delay(S,omega)+delay(E,omega)+delay(R,omega))

ddt_I = -(gamma+epsilon+d)*I
+lambda*delay(S,omega)*delay(I,omega)*exp(-d*omega)
/(delay(I,omega)+delay(S,omega)+delay(E,omega)+delay(R,omega))

ddt_R = gamma*I -d*R  -gamma*delay(I,tau)*exp(-d*tau)
```

We can display the different components of the model with `simulx`:

seir_simulx.m

```
p.name =  {'A' ,'lambda','gamma','epsilon','d' ,'omega','tau'} ;
p.value = [ 0.5,   0.4  , 0.02  ,  0.1  , 0.01 ,   0.3 , 60  ];
out = struct('name',{{'S', 'E', 'I', 'R'}},'time',-20:350);
res = simulx('model','seir_model.txt','parameter',p,'output',out);
```



2.6.4 Gene expression in single cells

We consider here the high-osmolarity glycerol (HOG) pathway in budding yeast. The related Hog1-induced gene expression model is given by a reaction network described in Gonzalez et al. (2013).

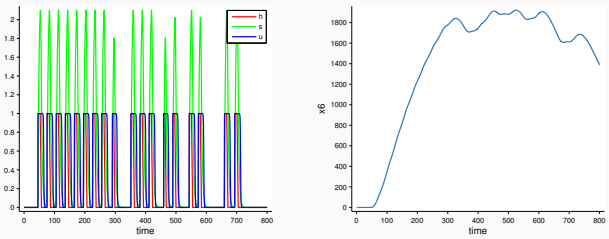
The input $u(t-\tau)$ is the delayed gene activation rate caused by an osmotic shock.

hog_model.txt

```
[LONGITUDINAL]
input={c1,c2,c3,c4,c5,c6,c7,c8,x30,tau}

PK:
depot(target=h,Tlag=tau)

EQUATION:
kappa=0.7
gamma=0.3
nh=6.13
Kd=0.1418
s0=0.0158

ddt_h=0
ddt_s=kappa*h - gamma*s
sh=(s+s0)^nh
u=sh/(Kd^nh+sh)

x1_0 = 1
x3_0 = x30
ddt_x1 = c2*x2 - c1*u*x1
ddt_x2 = c1*u*x1 - c2*x2 + c4*x4 - c3*x2*x3
ddt_x3 = c4*x4 - c3*x2*x3
ddt_x4 = c3*x2*x3 - c4*x4
ddt_x5 = c5*x4 - c8*x5
ddt_x6 = c6*x5 - c7*x6
```

hog_simulx.m

```
ton = struct('amount',1, 'rate',1);
ton.time=[35, 65, 95, 125, 155, 185, 215, 245, 280,...
          341, 371, 401, 449, 479, 533, 563, 649, 683];
toff = struct('amount',-1, 'rate',0.25);
toff.time=[43, 73, 103, 133, 163, 193, 223, 253, 285,...
           349, 379, 409, 454, 486, 541, 570, 657, 691];
tr={ton,toff};

p.name = {'c1','c2','c3','c4','c5','c6','c7','c8','x30','tau'} ;
p.value = [75,2500,0.00002,0.04,1.2,800,0.005,0.1,150,10];
out = struct('name',{{'h', 's', 'u', 'x6'}},'time',0:800);
res = simulx('model','hog_model.txt','treatment',tr,'parameter',p,'output',out);

opt.output={{'h','s','u'}, {'x6'}};
opt.symbol={{'r-','g-','b-'}, {'-'}};
plotmlx(res,'option',opt)
```



To explore the model using `mlxplore`:

mlxplore

```
>> mlxplore('model','hog_model.txt','treatment',tr,'parameter',p,'output',out)
```

2.7 Extensions

The demos used in this section are in `/demos/7_extensions`

2.7.1 Inline MLXTRANmodel

Instead of defining the model in an external text file, it is possible to define it in the `simulx` script using the function `inlineModel`.

The MLXTRAN script should be inserted just after the MATLAB command `myModel = inlineModel;` (`myModel` can be any name defined by the user), in the comment area delimited by `\%{` and `\%}`.

inline_simulx.m

```
myModel = inlineModel;
%{
  [LONGITUDINAL]
  input = {u, v}
  EQUATION:
  t0=0.2
  f_0=10
  ddt_f = -u*f/(v+f)
%}

f = struct('name','f','time',0:0.01:2);
p = struct('name',{'u','v'},'value',[40,10]);
res = simulx('model',myModel,'parameter',p,'output',f);
```

Remark: It is possible also to run `mlxplore` with an inline defined model

```
>>mlxplore('model',myModel,'parameter',p,'output',f)
```

2.7.2 Writing the simulated data in a file

A datafile with the simulated data can be created using the setting `recordFile`.

record1_simulx.m

```
myModel = inlineModel;
%{
  [LONGITUDINAL]
  input = {u, v}
  EQUATION:
  t0=0.2
  f_0=10
  ddt_f = -u*f/(v+f)
%}

f = struct('name','f','time',0:0.01:2);
p = struct('name',{'u','v'},'value',[40,10]);
s = struct('recordFile','simulx1_data.txt');
res = simulx('model',myModel,'parameter',p,'output',f,'settings',s);
```

See also example `record2\simulx.R` for a more complex examples where groups with several levels of randomization are created.

2.7.3 Optimization

When `simulx` is called for the first time with a given MLXTRAN model, a C++ code is automatically generated from this model and automatically compiled. This process takes some time (a few seconds). If `simulx` is called again with the same model, this process is not performed anymore. The run is then much faster than the first one.

For each run of `simulx`, the design of the simulation must be created and loaded in memory. This process is also time consuming. If several calls of `simulx` are done successively with the same model and the same design, it is possible to optimize significantly the computational time by preventing `simulx` to create and load the design each time.

`simulx` converts the inputs of the function defined in several cell arrays and structures into a unique cell array which can be processed by a mex file `Mlxcompute`. It is possible to define this new structure as a second output argument of `simulx`.

```
»[res,dataIn] = simulx(... );
```

Using then dataIn as input argument of `simulx` will prevent to create the design again.

```
»res = simulx('data',dataIn);
```

Furthermore, setting the settings `loadDesign` to `false` prevents to load the design again.

```
»s = struct('loadDesign',false);
»res = simulx('data',dataIn,'settings',s);
```

Suppose that we want to simulate 1000 replicates of the same trial using always the same design but different individual parameters. We can prevent to load the design for each replicate as follows:

optim2_simulx.m

```
myModel=inlineModel;
%{
[LONGITUDINAL]
input = {V, k}
EQUATION:
Cc = pkmodel(V,k)
%}

adm = struct('time',0:12:200,'amount',100);
Cc = struct('name','Cc','time',200:2:224);

N=100;
V_pop=10;
k_pop=0.1;
omega_V=0.3;
omega_k=0.2;
O=diag([omega_V, omega_k]);
mu = repmat([V_pop, k_pop],N,1);
p0=mu.*exp(randn(N,2)*O);
p.name={'V','k'};
p.header={'id','V','k'};
p.value = [(1:N)' p0];

[res,dataIn] = simulx('model',myModel,'parameter',p,'output',Cc,'treatment',adm);

M=1000;
data=cell(1,M);
s = struct('loadDesign',false);
for m=1:M
    pm=mu.*exp(randn(N,2)*O);
    dataIn.individual_parameters.value=pm;
    res = simulx('data',dataIn,'settings',s);
    data(m)={res};
end
```

Look at example `optim3_simulx` to see how this approach can be efficiently used for computing confidence intervals via Monte-Carlo simulation.

2.7.4 Initialization of the random number generator

Use the field `seed` of `settings` to set the initialization of the random number generator.

```
»s = struct('seed',123456);
»res = simulx(..., 'settings',s);
```

The same results will be obtained at each run if the seed is not modified.

See examples `seed1_simulx` and `seed2_simulx`.

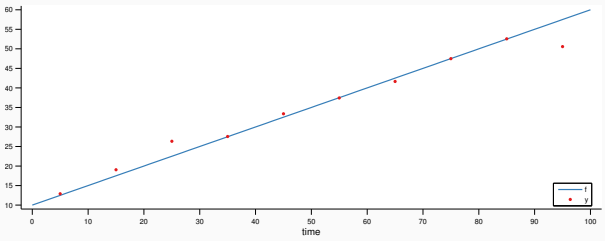
2.7.5 Defining random variables in the EQUATION block

It is possible to define random variables in the `EQUATION` block instead of the `DEFINITION` block of the MLXTRAN script, using the syntax `x ~ normal(mu, sigma)` where `mu` is the mean and `sigma` the standard deviation of a normal random variable.

Then `x` can be used in the `EQUATION` block as any other variable.

random_simulx.m

```
myModel = inlineModel;
%{
[LONGITUDINAL]
input =  {a, b, s}

EQUATION:
f = a + b*t
e ~ normal(0,s)
y = f*exp(e)
%}

o1  = struct('name','f','time',0:100);
o2  = struct('name','y','time',5:10:100);
p   = struct('name',{{'a','b','s'}},'value',[10, 0.5, 0.1]);
res = simulx('model',myModel,'parameter',p,'output',{o1,o2});

opt.output = {{'f','y'}};
opt.symbol = {{'-','.'}};
plotmlx(res,'option',opt)
```



2.7.6 Defining correlation between Gaussian random variables

It is possible with MLXTRAN to introduce linear correlations between Gaussian random variables. Each element of the variance-covariance matrix should be defined.

In the following example, $(\log(a_i), b_i, \text{logit}(c_i)$ is a Gaussian vector. Then, `{r(a,b)` is the correlation between $\log(a_i)$ and b_i. Since the matrix is symmetric, it is not necessary to define `{r(b,a)`. Any correlation which is not defined is assumed to be 0.

correlation_simulx.m

```
myModel = inlineModel;
%{
[INDIVIDUAL]
input = {a_pop, o_a, b_pop, o_b, c_pop, o_c, r_ab, r_ac, r_bc}
DEFINITION:
a = {distribution=lognormal,   reference=a_pop, sd=o_a}
b = {distribution=normal,      reference=b_pop, sd=o_b}
c = {distribution=logitnormal, reference=c_pop, sd=o_c}
correlation = {r(a,b)=r_ab, r(a,c)=r_ac, r(b,c)=r_bc}

[LONGITUDINAL]
input={a, b, c}
EQUATION:
f = a + b*t + c*t^2
%}

op = struct('name',{{'a','b','c'}});
of = struct('name','f','time',0:0.1:4);
p  = struct('name',{{'a_pop','o_a','b_pop','o_b','c_pop','o_c','r_ab','r_ac','r_bc'}},...
            'value',[10, 0.3, -5, 0.5, 0.8, 0.4, -0.6, -0.4, 0.7]);
g  = struct('size',1000,'level','individual');

res = simulx('model',myModel,'parameter',p,'group',g,'output',{op,of});
```

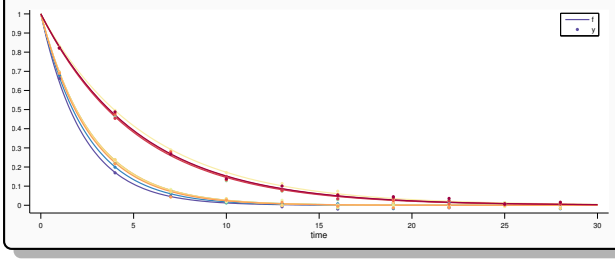
2.7.7 Simulation of categorical covariates

The distribution for categorical covariates can be defined in the section `[COVARIATE]`

Males and females are simulated in this example. The individual parameter k depends then on gender.

correlation_simulx.m

```
myModel = inlineModel;
%{
[COVARIATE]
input={p_F}
DEFINITION:
gender = { type=categorical,
           categories={M,F},
           P(gender=F)=p_F }

[INDIVIDUAL]
input={k_pop, omega_k, gender, beta_F}
gender={type=categorical,categories={M,F}}
DEFINITION:
k = { distribution=lognormal,
      reference=k_pop,
      covariate=gender,
      coefficient={0,beta_F},
      sd=omega_k }

[LONGITUDINAL]
input =  {k, a}
EQUATION:
f = exp(-k*t)
DEFINITION:
y = {distribution=normal, prediction=f, sd=a}
%}

p = struct('name',{{'p_F','k_pop','omega_k','beta_F','a'}},...
           'value',[0.4,0.2,0.1,0.6,0.01]);
f = struct('name','f','time',0:0.1:30);
y = struct('name','y','time',1:3:30);
ind = struct('name',{{'gender'}});
out={ind,f,y};
g = struct('size',12,'level','covariate');
s.seed=12345;
res = simulx('model',myModel,'parameter',p,'output',out,...
             'group',g,'settings',s);

gender=cell2mat(res{1}.value);
freq_F = mean(gender==2);
disp(['proportion of females: ',num2str(freq_F)]);
opt = struct;
opt.output = {{'f','y'}};
opt.symbol = {{'-','.'}};
%opt.split = 'gender';
plotmlx(res,'option',opt)
```



See demos `cat2_simulx` for another example where the model combines a continuous and a categorical covariate with 3 categories.

2.8 Integrating **simulx** in a workflow

The demos used in this section are in `/demos/8_workflow`

A typical workflow consists first in modelling some data (i.e., building a model and estimating the parameters of the model) and then simulating new data from the model.

We can use, for instance, MONOLIX for modelling and simulx for simulation. Then, a MONOLIX project is directly used as input of simulx: the design (subjects, times of measurement, dosing regimens) and the individual covariates defined in the original datafile, the model used for estimation and the estimated parameters are used for the simulation. In other word, all the information required for simulation is directly obtained from the MONOLIX project.

In this example, MONOLIX was used for modelling the warfarin PKPD data. Project warfarinPKPD_project.mlxtran contains all the information about data, model and algorithms used for estimating the population and the individual PKPD parameters. This project can be loaded in MONOLIX and run:

- the data is in the file warfarin_data.txt,
- the structural model is implemented in pkpd1_mlxt.txt,

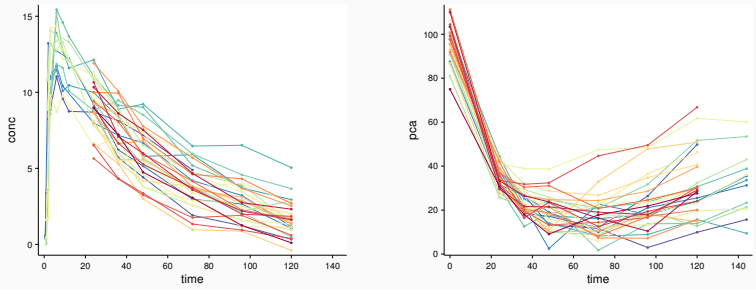
pkpd1_mlxt.txt

```
INPUT:
parameter =  {Tlag, ka, V, Cl, Imax, IC50, kin, kout}

EQUATION:
Cc = pkmodel(Tlag, ka, V, Cl)
E_0 = kin/kout
ddt_E = kin*(1-Imax*Cc/(Cc+IC50)) - kout*E

OUTPUT:
output = {Cc, E}
```

- the results are in the folder warfarinPKPD_project:
  - estimates.txt is a table with the population parameter estimates,
  - indiv_parameters.txt is a table with the individual parameter estimates (conditional modes and conditional means).

workflow1_simulx.m

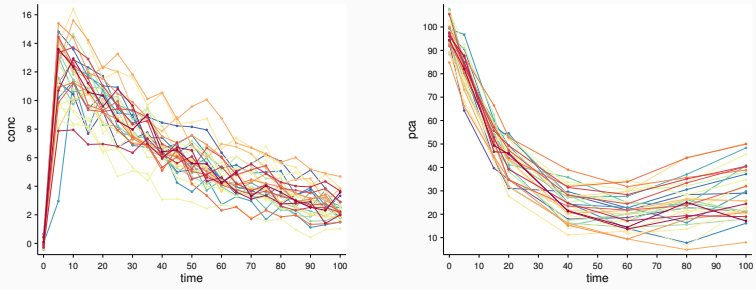
```
res = simulx('project','projects/warfarinPKPD_project.mlxtran');
opt = struct('legend','off','symbol','.-');
stg = struct('linewidth',1,'markersize',8);
plotmlx(res,'option',opt,'setting',stg)
```



The original design can be modified for simulation. We may want, for instance, to simulate PK and PD data at times different from those of the original data.

workflow2_simulx.m

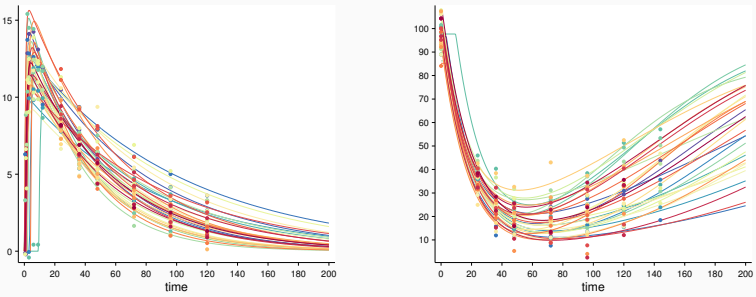
```
out1 = struct('name','conc','time',0:5:100);
out2 = struct('name','pca','time',[0 5 15 20:20:100]);

res = simulx('project','projects/warfarinPKPD_project.mlxtran',...
             'output',{out1, out2});
opt = struct('legend','off','symbol','.-');
stg = struct('linewidth',1,'markersize',8);
plotmlx(res,'option',opt,'setting',stg)
```



We may also want to define additional outputs, such as individual covariates, individual parameters, or any variable defined in the structural model.

workflow3_simulx.m

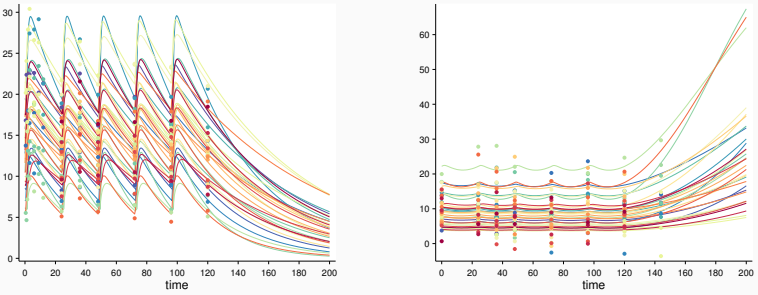
```
out1 = struct('name','wt');
out2 = struct('name',{{'V','IC50'}});
out3 = struct('name',{{'Cc','E'}},'time',0:0.5:200);

res = simulx('project','projects/warfarinPKPD_project.mlxtran',...
             'output',{out1, out2, out3});

opt.output = {{'Cc','conc'} , {'E','pca'}};
opt.symbol = {{'-','.'} , {'-','.'}};
stg = struct('linewidth',1,'markersize',12);
plotmlx(res,'option',opt,'setting',stg)
```



It is possible to replace the original dosing regimen with a new one.

workflow4_simulx.m

```
adm = struct('amount',50,'time',-240:24:96);
out = struct('name',{{'Cc','E'}},'time',0:0.5:200);

res = simulx('project','projects/warfarinPKPD_project.mlxtran',...
             'treatment',adm,'output',out);

opt.output = {{'Cc','conc'} , {'E','pca'}};
opt.symbol = {{'-','.'} , {'-','.'}};
stg = struct('linewidth',1,'markersize',12);
plotmlx(res,'option',opt,'setting',stg)
```



It is possible to replace some of the estimated parameters by some user defined values. The original covariates of the datafile can also be replaced by user defined – or simulated – values.

workflow5_simulx.m

```
p1 = struct('name',{{'V_pop','omega_V','beta_V_lw70'}},'value',[10, 0.2, 0]);
p2 = struct('name','wt','header',{{'id', 'value'}},'value',[(1:32)' 70+randn(32,1)*10]);
out = struct('name',{{'wt','V','IC50'}});

res = simulx('project','projects/warfarinPKPD_project.mlxtran',...
             'parameter',{p1,p2},'output',out);
```

Instead of simulating the individual parameters, individual estimates can be used for simulating the longitudinal data.

The conditional modes and/or conditional means computed by MONOLIX are stored in the table `indiv_parameters.txt`. Then, Use `simulx('parameter','mode',...)` for using the mode and `simulx('parameter','mean',...)` for using the mean.

workflow6_simulx.m

```
out = struct('name',{{'Tlag','ka','V','Cl','IC50'}});

res = simulx('project','projects/warfarinPKPD_project.mlxtran',...
             'parameter','mode',...
             'output',out);

displaymlx(res)

    id      Tlag        ka         V        Cl      IC50
     1      0.55      0.33      8.29      0.27      1.19
     2      1.00      0.99      7.89      0.11      1.54
     3      0.57      0.76      8.02      0.13      1.74
     4      0.82      1.05      8.60      0.11      0.48
     5      0.80      1.27      4.99      0.07      1.30
     :         :         :         :         :         :
```

Lastly, it is possible to replace the original model by a new one which uses the same parameters (or a subset of the original parameters).

In this example, a PK model without lag-time and a direct response model is used instead of the original PKPD model implemented in `pkpd1_model.txt`

pkpd2_model.txt

```
[LONGITUDINAL]
input = {a_1, b_1, a_2, ka, V, Cl, Imax, IC50, kin, kout}

EQUATION:
Cc = pkmodel(ka, V, Cl)
E0 = kin/kout
E = E0*(1-Imax*Cc/(Cc+IC50))

DEFINITION:
conc = {distribution=normal, prediction=Cc, errorModel=combined2(a_1,b_1) }
pca  = {distribution=normal, prediction=E, errorModel=constant(a_2) }

[INDIVIDUAL]
  .
  .
  .
```

workflow7_simulx.m

```
res = simulx('project','projects/warfarinPKPD_project.mlxtran',...
             'model','projects/pkpd2_model.txt');
```

Remark: Not all the MONOLIX projects are supported by this version of `simulx`. Features not supported:

- Mixture models
- IOV
- Steady state
- categorical covariates

2.9 Using PharmML model

The demos used in this section are in `/demos/9_pharmml`

When a PharmML model is used with `simulx`, it is automatically converted into a MLXTRAN model by `simulx`.

A PharmML model can also be converted into a MLXTRAN model outside of `simulx` using the function `pharmml2mlxtran`:

```
>> pharmml2mlxtran('tgi2_model.xml')
```

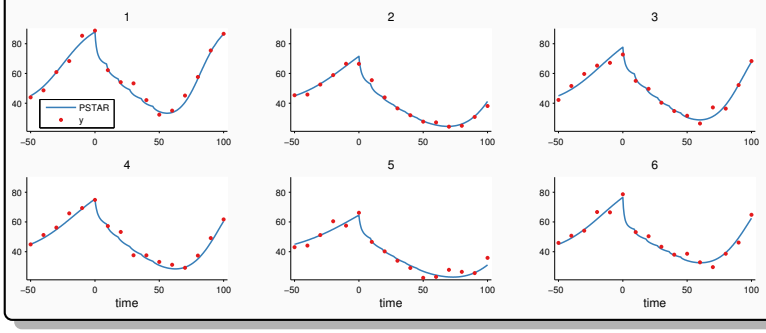
Then, a new text file (`tgi_model_mlxt.txt` in this example) with the same model encoded with MLXTRAN is automatically created.

Four examples of the use of `simulx` with PharmML are provided in the folder `simulxR/demos/pharmML`.

- pharmml1_simulx.R uses the model pk_model.xml
- pharmml2_simulx.R uses the model pkpd_model.xml
- pharmml3_simulx.R and pharmml4_simulx.R use the model tgi2_model.xml. These examples are the same as demos/examples/tgi1_simulix.R and demos/examples/tgi2_simulix.R, but using tgi2_model.xml instead of tgi1_model.txt and tgi2_model.txt

pharmml4_simulx.m

```
adm = struct('amount',1,'time',0:9:45,'target','C');
f = struct('name','PSTAR','time',-50:0.5:100);
y = struct('name','y','time',-50:10:100);
out = {f, y};
psi.name={'K','KDE_pop','omega_KDE','KPQ_pop','omega_KPQ','KQPP_pop',...
          'omega_KQPP','LAMBDAP_pop','omega_LAMBDAP','GAMMA_pop',...
          'omega_GAMMA','DELTAQP_pop','omega_DELTAQP','a'};
psi.value=[100,0.3,0.2,0.025,0.2,0.004,0.2,0.12,0.2,1,0.2,0.01,0.2,3];
g = struct('treatment',adm,'size',6, 'level','individual');

res=simulx('model','tgi2_model.xml','parameter',psi,'group',g,'output',out);

opt.output = {{'PSTAR','y'}}; % PSTAR and y on the same plots
opt.symbol = {{'-','.'}};
opt.split = 'individual'; % 1 individual per subplot
plotmlx(res,'option',opt)
```



There are some limitations in the translation PharmML -> MLXTRAN with this version of Mlxm:

- only definitions are supported for defining individual parameters (equations are not supported),
- only statistical models for the longitudinal data and the individual parameters can be defined (statistical models for the population parameters and for the covariates are not supported),

Remark: these limitations concern the translation PharmML -> MLXTRAN, not MLXTRAN itself which accepts all these features.

Chapter 3

pkmodel

3.1 Overview

The `pkmodel` function used in a MLXTRAN script (see Section 2.4.2) can also be used as a MATLAB function.

Many standard PK models can be implemented using `pkmodel`, including, 1, 2 and 3 compartments model, linear and nonlinear elimination, lag time or transit compartment, etc.

Usage

```
res = pkmodel(time, dose, parameter)
```

Input arguments

`time`	a $n \times 1$ vector of times	
`dose`	a structure with fields	
	`time`	a vector of times of administration
	`amount`	a scalar or a vector of amounts
	`rate` (*opt.*)	a scalar or a vector of infusion rates
`parameter`	a structure with fields	
	`name`	a cell array of names of PK parameters
	`value`	a vector of parameters values (or a matrix with N rows)

It's the names of the PK parameters that define the PK model.

Output arguments

`res` a structure with fields

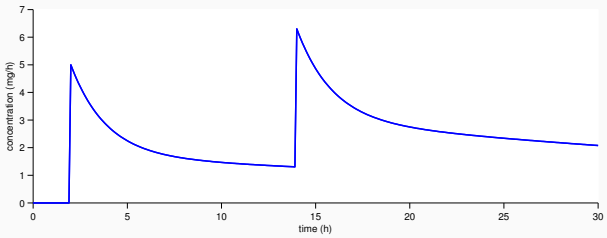
- `time` the $n \times 1$ vector of times
- `cc` predicted concentration in the central compartment ($n \times N$ matrix with one column per individual)
- `ce` predicted concentration in the effect compartment (only if `ke0` is defined as input parameter)

3.2 Examples

A two compartments model for infusion administrations is used in this first example:

demo1_pkmodel.m

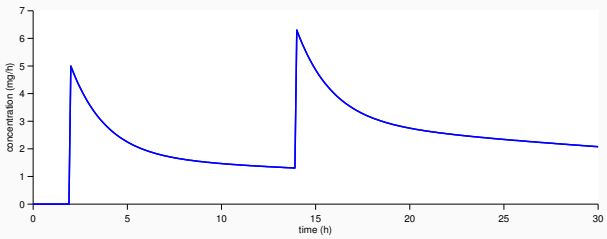
```
t = 0:0.1:30;
adm = struct('time',[2 14], 'amount',40);
p = struct('name',{{'V','Cl','k12','k21'}},'value',[8,0.5,0.3,0.2]);
res = pkmodel(t,adm,p);

plot(res.t,res.cc)
xlabel('time (h)');  ylabel('concentration (mg/h)');
```



A model with transit compartment is used in this second example:

demo2_pkmodel.m

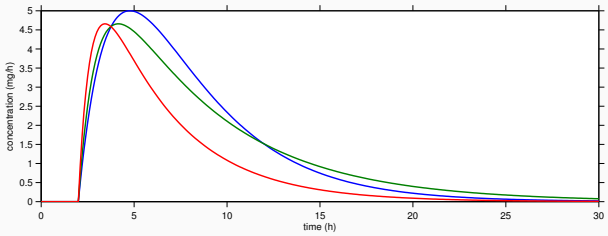
```
t = 0:0.1:80;
adm = struct('time',[1,13,37], 'amount',1);
p = struct('name',{{'Mtt', 'Ktr', 'ka', 'V', 'Vm', 'Km', 'p'}},...
           'value',[5, 1, 0.5, 10, 1, 0.6, 0.5]);
res = pkmodel(t,adm,p);

plot(res.t,res.cc)
xlabel('time (h)');  ylabel('concentration (g/h)');
```



It is possible to compute the concentration for several sets of PK parameters:

demo3_pkmodel.m

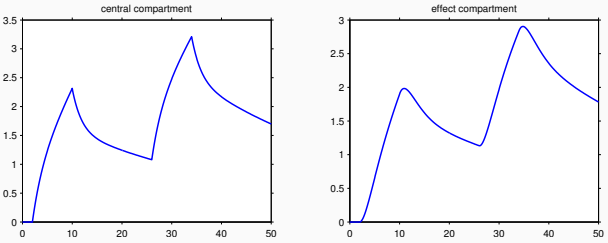
```
t = 0:0.1:30;
adm = struct('time',2, 'amount',40);
p = struct('name',{{'ka','V','Cl'}},'header',{{'id','ka','V','Cl'}});
p.value = [1, 0.5, 4,   1
           2,   1, 6,   1
           3, 1.5, 6, 1.5];
res = pkmodel(t,adm,p);

plot(res.t,res.cc)
xlabel('time (h)');  ylabel('concentration (mg/h)');
```



If `ke0` is used as an input argument of the function, then this parameter is used as the transfer rate constant between the central and effect compartments. The output argument `r` contains an additional column `ce` which is the concentration in the central compartment:

demo4_pkmodel.m

```
t = 0:0.1:50;
adm = struct('time',2:24:100, 'amount',40, 'rate',5);
p = struct('name',{{'V','Cl','k12','k21','ke0'}},...
           'value',[8,0.5,0.3,0.2,0.5]);
res = pkmodel(t,adm,p);

subplot(121); plot(res.t,res.cc); title('central compartment')
subplot(122); plot(res.t,res.ce); title('effect compartment')
```



3.3 Input arguments of the **pkmodel** function

- `ka` : absorption rate constant,
- `Tk0` : duration of a zero order absorption,
- `Tlag` : lag time,
- `Mtt` and `Ktr` : mean transit time and transfer rate constant between transit compartments,
- `p` : fraction of the dose which is absorbed,
- `k` : elimination rate constant,
- `Cl` : clearance,

- (Vm,Km) : parameters of the Michaelis Menten equation
- (k12, k21) : transfer rate constants between the central and the peripheral compartment 2,
- (k13, k31) : transfer rate constants between the central and the peripheral compartment 3,
- ke0 : transfer rate constant between the central and the effect compartment.

Chapter 4

mlxplore

4.1 Overview

MLXPLORE is a graphical and interactive software for the exploration and visualization of complex models.

MLXPLORE is available at

http://www.lixoft.eu/products/mlxplore/get-mlxplore/

Function `mlxplore` allows us to run MLXPLORE from MATLAB.

Usage

mlxplore(`'model','mlxt.txt','parameter',par,'output',out`) displays the longitudinal outputs defined in `out` using model `mlxt.txt` with parameters values defined in `par`.

mlxplore(`'model', ... ,'treatment',tr`) uses the *source terms* (i.e. the doses for pharmacometrics applications) defined in `tr`.

mlxplore(`'model', ... ,'group',gr`) creates different individuals (groups of size 1) using the information in `gr`.

Input arguments

`model`	a model MLXTRAN or `PharmML` used for the simulation	
`parameter`	a structure with fields	
	`name`	a cell array of parameters names
	`value`	a vector of parameters values
`output`	a a structure with fields	
	`name`	a cell array of outputs names
	`time`	a vector of times
`treatment` (*opt.*)	a a structure with fields	
	`time`	a vector of inputs times
	`amount`	a scalar or a vector of amounts
	`rate` (*opt.*)	a scalar or a vector of infusion rates
	`type` (*opt.*)	an integer (in the case of multiple types of administration)

group (*opt.*)	a structure with field	
	treatment	used for comparing different treatments

As simulx, mlxplore can use models implemented both in MLXTRAN and PharmML.

MLXPLORE only displays functions of times. Then, a section [LONGITUDINAL] with a block EQUATION is mandatory. Only variables defined in this bloc EQUATION are displayed. A block DEFINITION will be ignored by MLXPLORE. Sections [COVARIATE] and [INDIVIDUAL] can be used with MLXPLORE for exploring the statistical component of the model.

See the MLXPLORE documentation for more information about MLXTRAN for MLXPLORE (Lixoft, 2014).

4.2 Examples

The demos used in this section are in /demos/6_examples

Several other demos are available in the different demo folders. Each of these demos is based on a demo for simulx and mainly consists in replacing

»res = simulx('model', ...)

with

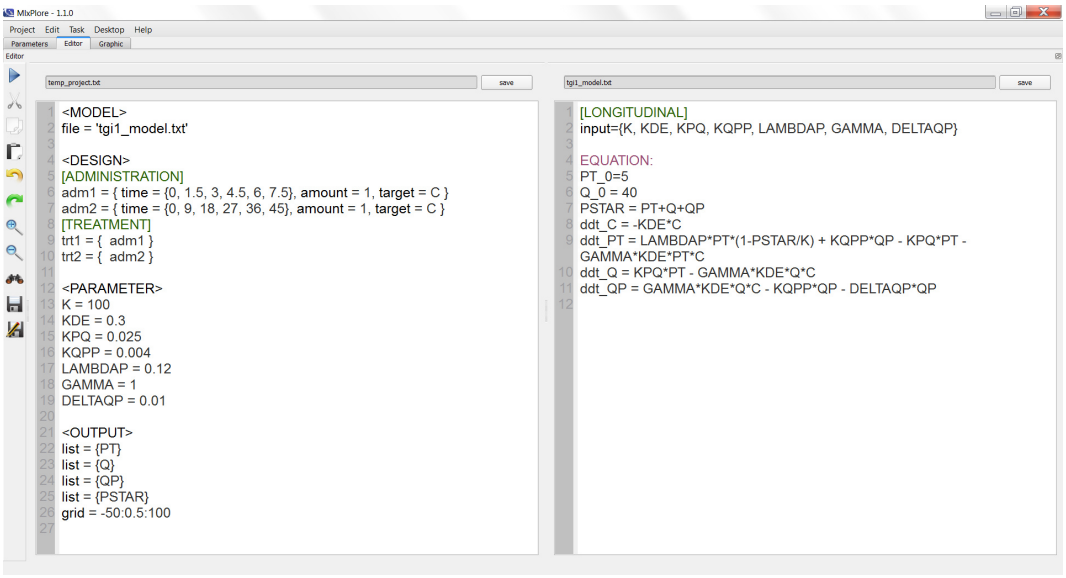
»mlxplore('model', ...)

We have used simulx in Section 2.6.2 to simulate the tumor growth model of Ribba et al. (2012). It is now possible to explore this model using mlxplore. Exactly the same MATLAB script can be used to define the design, the dose regimens and the outputs, except that we use now the MATLAB function mlxplore instead of simulx to launch automatically MLXPLORE.

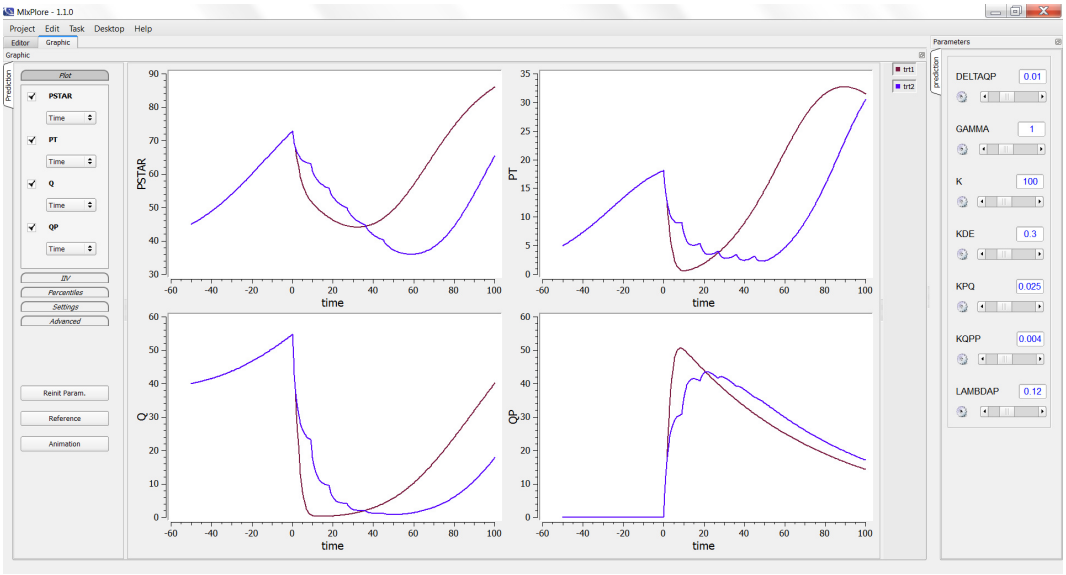
tgi1_mlxplore.m

```
adm1 = struct('amount',1,'time',0:1.5:7.5,'target','C');
adm2 = struct('amount',1,'time',0:9:45,'target','C');

f = struct('name',{{'PT','Q','QP','PSTAR'}},'time',-50:0.5:100);

psi.name={'K','KDE','KPQ','KQPP','LAMBDAP','GAMMA','DELTAQP'};
psi.value=[100, 0.3, 0.025, 0.004, 0.12, 1, 0.01];

g1 = struct('treatment',adm1);
g2 = struct('treatment',adm2);
g = {g1, g2};

mlxplore('model','tgi1_model.txt','parameter',psi,'group',g,'output',f);
```

mlxplore automatically creates a script for MLXPLORE with the same information:



It is then possible to modify the parameters values with the sliders and visualize the impact on the curves which are displayed.



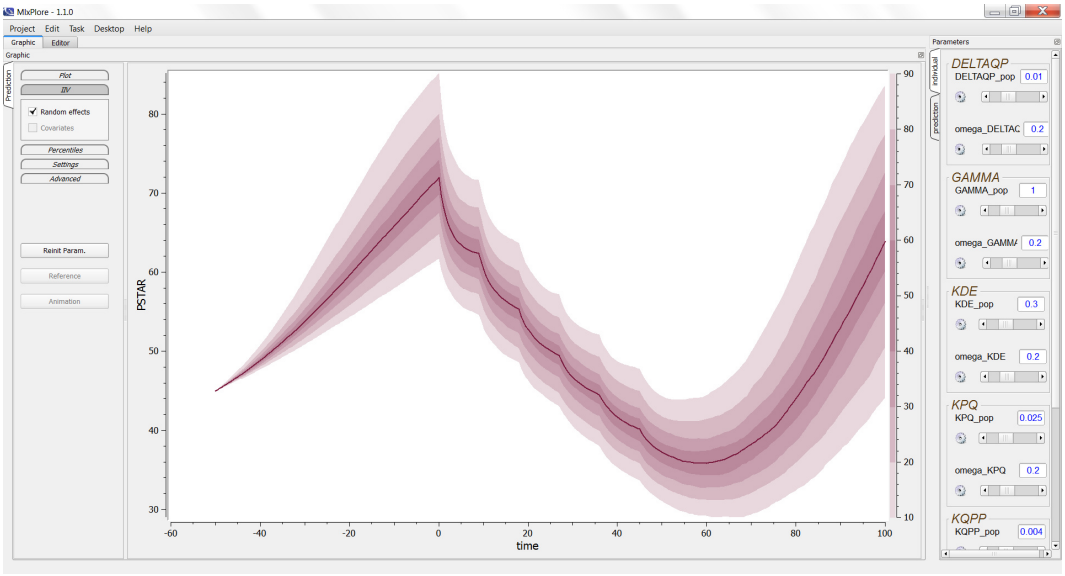
`mlxplore` allows us to visualize not only the structural model but also the statistical model, which is of fundamental importance in the population approach. We can thus visualize the impact of inter-individual variability of model parameters on predictions.

We will now use `mlxplore` for exploring the statistical component of model `tgi2_model.txt`. We use in this example the same design and the same parameters values as those used in tgi2_mlxplore.m

tgi2_mlxplore.m

```
adm = struct('amount',1,'time',0:9:45,'target','C');

psi.name={'K','KDE_pop','omega_KDE','KPQ_pop','omega_KPQ','KQPP_pop',...
          'omega_KQPP','LAMBDAP_pop','omega_LAMBDAP','GAMMA_pop',...
          'omega_GAMMA','DELTAQP_pop','omega_DELTAQP','a'};
psi.value=[100,0.3,0.2,0.025,0.2,0.004,0.2,0.12,0.2,1,0.2,0.01,0.2,3];

f = struct('name','PSTAR','time',-50:0.5:100);

mlxplore('model','tgi2_model.txt','treatment',adm,'parameter',psi,'output',f);
```

Here, MLXPLORE is used to visualize the probability distribution of P^*: 20%,40%, 60% and 80% prediction intervals are displayed with different colors. It is then possible to modify interactively the values of the population parameters (reference values and standard deviations of the random effects) to visualize the impact on the probability distribution of the tumor growth in the population.



Chapter 5

Processing the outputs of **simulx**

5.1 plotmlx

5.1.1 Usage

plotmlx plot of longitudinal data simulated with simulx.

plotmlx(res) plots the components of res which are longitudinal data and where res is the output of simulx.

plotmlx(res,'option',opt) uses the options defined in the structure opt. The following table lists the fields of opt. All these fields are optional.

Field	Value	Description
output	Cell array of cell arrays. Default = all the outputs	Lists of outputs to be displayed. Each cell contains the list of outputs to be displayed in a same panel
symbol	Cell array of cell arrays. Default = '-' (solid lines with default color order).	Colors and symbols to be used for the plots. Same dimension as output
split	{'none'}|'individual'|'group'	Specifies the variable on which to split the data into separate subpanels.
figure	Vector of integers. Default = [1, 2, ...]	Handles of figures
legend	{'on'}|'off'	Display a legend
xlabel	Character string	X-axis label
tile	{'horizontal'} | 'vertical'	Tile plots with horizontal/vertical panels

plotmlx(res,'settings',stg, ...) uses the graphical settings defined in the structure stg. The following table lists the fields of opt. All these fields are optional.

Field	Value	Description
linewidth	{0.5} | scalar	Line width in points
markersize	{6} | scalar	Marker size in points
labelfontsize	Integer	Font size of the axis labels
tickfontsize	Integer	Font size of the axis tick labels
titlefontsize	Integer	Font size of the titles
colormap	Matlab built-in colormap	Color look-up table

5.1.2 Examples

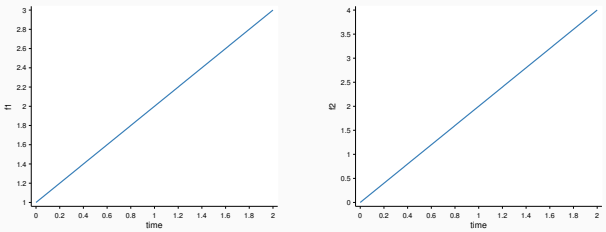
Example 1

We us simulx in this first example for computing 2 functions of time f_1 and f_2 and for simulating 2 sequences of longitudinal data y_1 and y_2 for only one individual. Here, the output of simulx is a cell array res with four cells containing, respectively, f_1, f_2, y_1 and y_2.

Let's start plotting f_1 and f_2 on 2 different panels using the default options and settings.

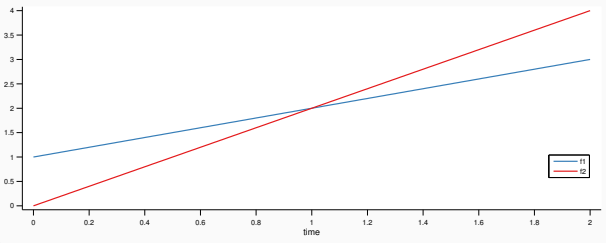
demo1_plotmlx.m

```
myModel = inlineModel;
%{
  [LONGITUDINAL]
  EQUATION:
  f1 = 1+t
  f2 = 2*t
  DEFINITION:
  y1 = {distribution=normal, prediction=f1, sd=0.3}
  y2 = {distribution=normal, prediction=f2, sd=0.1}
%}

f = struct('name',{{'f1', 'f2'}},'time',0:0.01:2);
y = struct('name',{{'y1', 'y2'}},'time',0:0.25:2);
res = simulx('model',myModel,'output',{f,y});
plotmlx(res(1:2))
```



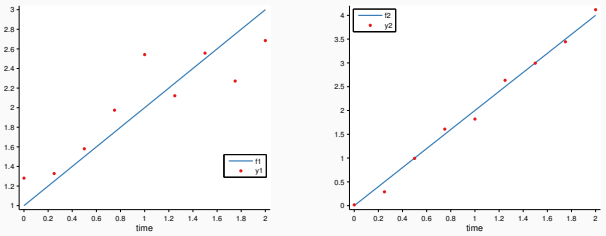
We can now plot f_1 and f_2 on the same panel:

```
>>opt.output = {{'f1', 'f2'}};
>>plotmlx(res,'option',opt)
```



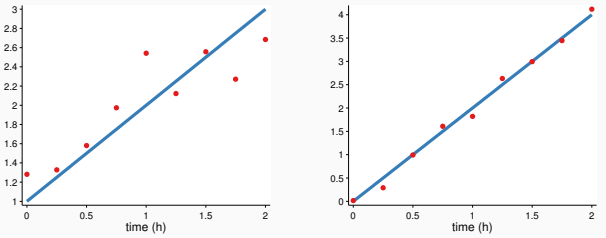
Now, plot f_1 and y_1 on a same subpanel with different symbols and f_2 and y_2 on another subpanel:

```
>>opt.output = {{'f1', 'y1'},{'f2', 'y2'}};
>>opt.symbol = {{'-', '.'},{'-', '.'}};
>>plotmlx(res,'option',opt)
```



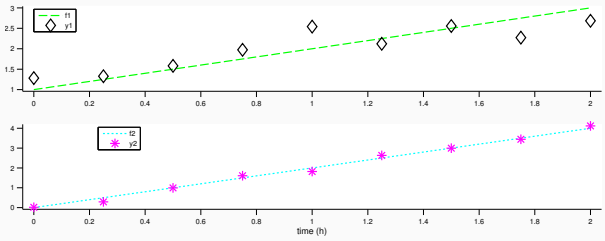
We can also remove the legend, change the x-axis label and modify the graphics settings:

```
>>opt.legend='off';
>>opt.xlabel='time (h)';
>>stg.linewidth=4;
>>stg.markersize=20;
>>stg.labelfontsize=15;
>>stg.tickfontsize=12;
>>plotmlx(res,'option',opt,'setting',stg)
```



Different symbols and colors can be used:

```
>>opt.symbol = {{'g--', 'kd'},{'c:', 'm*'}};
>>opt.tile='vertical';
>>opt.legend='on';
plotmlx(res,'option',opt)
```

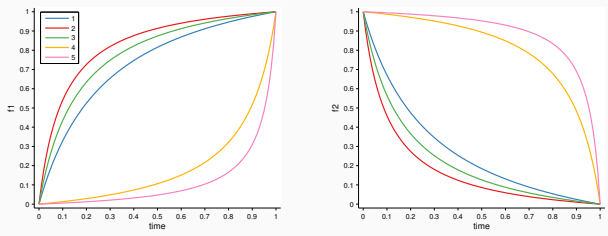


Example 2

We define 2 groups in this example and we simulate 3 individuals from group 1 and 2 individuals from group 2. We use then `plotmlx` for plotting the two functions f_1 and f_2.

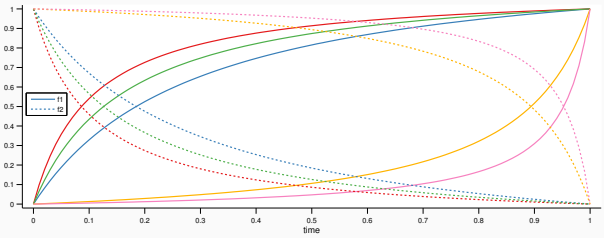
demo2_plotmlx.m

```
myModel = inlineModel;
%{
  [LONGITUDINAL]
  input = k
  EQUATION:
  if k>0
    f = (t+k*t)/(1+k*t)
  else
    f = t/(1-k+k*t)
  end
  f1 = f
  f2 = 1-f1
  DEFINITION:
  y1 = {distribution=normal, prediction=f1, sd=0.02}
  y2 = {distribution=normal, prediction=f2, sd=0.03}
  [INDIVIDUAL]
  input = k_pop
  DEFINITION:
  k = {distribution=normal, reference=k_pop, sd=5}
%}

f = struct('name',{{'f1', 'f2'}},'time',0:0.01:1);
y = struct('name',{{'y1', 'y2'}},'time',0.1:0.1:0.9);
p1 = struct('name', 'k_pop', 'value', 5);
p2 = struct('name', 'k_pop', 'value', -10);
g1 = struct('size', 3, 'level','individual', 'parameter', p1);
g2 = struct('size', 2, 'level','individual', 'parameter', p2);
res = simulx('model',myModel,'output',{f,y},'group',{g1,g2});
plotmlx(res(1:2))
```



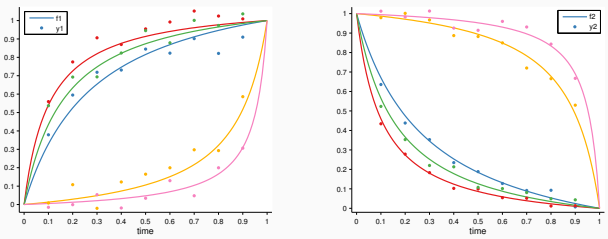
F_1 and f_2 are now displayed on the same panel with different symbols:

```
>>opt.output = {{'f1', 'f2'}};
>>opt.symbol = {{'-', ':'}};
>>plotmlx(res,'option',opt)
```



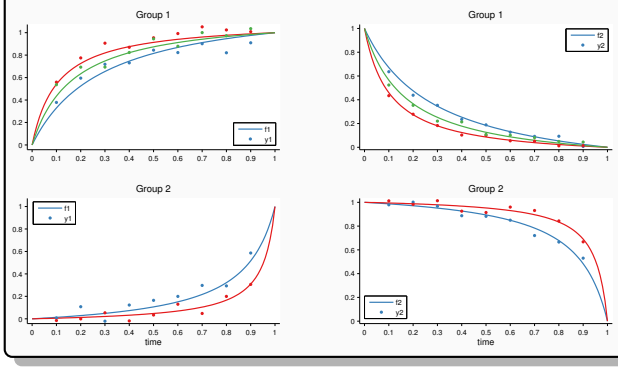
We can plot f_1 and y_1 on a same subpanel with different symbols and f_2 and y_2 on another subpanel:

```
>>opt.output = {{'f1', 'y1'},{'f2', 'y2'}};
>>opt.symbol = {{'-', '.'},{'-', '.'}};
>>plotmlx(res,'option',opt)
```



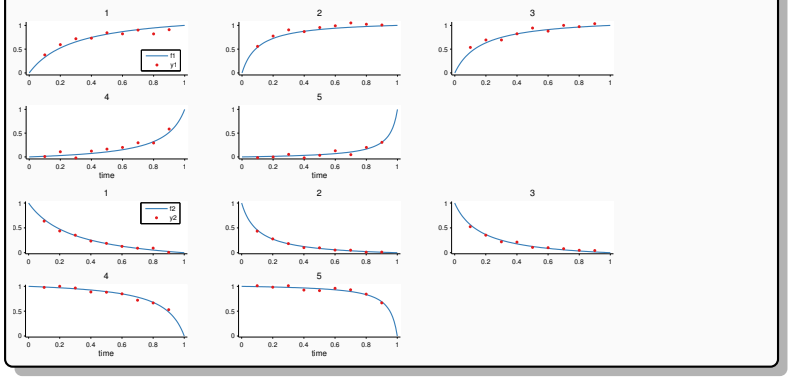
or split the data per group:

```
>>opt.split = 'group';
>>plotmlx(res,'option',opt)
```



or split the data per individual:

```
>>opt.split = 'individual';
>>plotmlx(res,'option',opt)
```



If a categorical covariate has been simulated, it can be used instead of `group` for splitting the data (see demos `cat1_simulx` and `cat2_simulx`).

5.2 kmplotmlx

kmplotmlx Kaplan-Meir plot of survival data simulated with `simulx`.

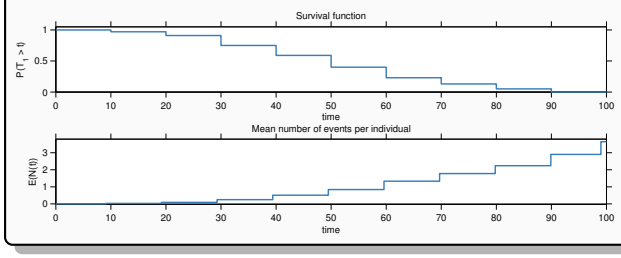
kmplotmlx(`event`) displays an estimate of the survival function obtained with `event`, where `event` is an output of `simulx`.

In the case of repeated events, the survival function for the first event is displayed, as well as the mean number of events per individual.

Repeated and interval censored events for 100 individuals are simulated in this example using model `event2_model.txt`.

event2_simulx.m

```
p = struct('name',{{'beta','lambda'}},'value',[2.5,50]);
h = struct('name','h','time',0:0.1:30);
e = struct('name','e','time',0);
g = struct('size',100);
res = simulx('model','event2_model.txt','parameter',p,'output',{h,e},'group',g);

kmplotmlx(res(2))
```



5.3 displaymlx

displaymlx display of the individual parameters simulated with `simulx`.

displaymlx(res) displays as a table the the outputs of simulx which are are individual parameters (longitudinal data is ignored). res is a cell array.

group2_simulx.m

```
adm = struct('time',0,'amount',100);
p = struct('name',{{'V_pop','omega_V','w','k','a'}},'value',[10,0.3,75,0.2,0.5]);
y = struct('name','y','time',0:1:10);
op= struct('name',{'V'});
g = struct('size',[2,3],'level',{{'individual','longitudinal'}});
res = simulx('model','group2_model.txt','parameter',p,'output',{y,op},...
             'treatment',adm,'group',g);
displaymlx(res)

    id         V
     1      8.70
     2      8.70
     3      8.70
     4      7.13
     5      7.13
     6      7.13
```

References

Genik, L. and Van Den Driessche, P. (1999). An epidemic model with recruitment-death demographics and discrete delays. *Differential Equations with Applications to Biology, Fields Institute Communications*, 21:237–249.

Gonzalez, A., Uhlendorf, J., Schaul, J., Cinquemani, E., Batt, G., and Ferrari-Trecate, G. (2013). Identification of biological models from single-cell data: a comparison between mixed-effects and moment-based inference. In *Proceedings of ECC - 12th European Control Conference*, pages 2652–2657.

Lavielle, M. (2014). *Mixed Effects Models for the Population Approach: Models, Tasks, Methods and Tools*, volume (to appear). Chapman & Hall/CRC Biostatistics Series.

Lixoft (2014a). Mlxtran documentation. Available at `http://www.lixoft.eu/wp-content/resources/docs/MonolixMlxtran.pdf`

Lixoft (2014b). Mlxtran tutorial. Available at `http://www.lixoft.eu/wp-content/resources/docs/modelMLXTRANtutorial.pdf`

Lixoft (2014c). Mlxplore documentation. Available at `http://www.lixoft.eu/wp-content/uploads/2013/09/MlxtranModel_forMlxPlore.pdf`

DDMoRe - WP4. PharmML 0.2.1. `http://pharmml.org`

Ribba, B., Kaloshi, G., Peyre, M., Ricard, D., Calvez, V., Tod, M., Cajavec-Bernard, B., Idbaih, A., Psimaras, D., Dainese, L., Pallud, J., Cartalat-Carel, S., Delattre, J.-Y., Honnorat, J., Grenier, E., and Ducray, F. (2012). A tumor growth inhibition model for low-grade glioma treated with chemotherapy or radiotherapy. *Clinical Cancer Research*, 18(18):5071–5080.