Package ‘REMixed’

June 27, 2025

Type Package

Title Regularized Estimation in Mixed Effect Model

Version 0.1.0

Maintainer Auriane Gabaut <auriane.gabaut@inria.fr>

Description
Implementation of an algorithm in two steps to estimate parameters of a model whose latent dynamics are inferred through latent processes, jointly regularized. This package uses 'Monolix' software (<https://monolixsuite.slp-software.com/>), which provide robust statistical method for non-linear mixed effects modeling. 'Monolix' must have been installed prior to use.

SystemRequirements 'Monolix'
(<https://monolixsuite.slp-software.com/>)

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports deSolve, Rsmlx, doSNOW, dplyr, fastGHQuad, ggplot2, snow,
stringr, Rmpfr

RoxygenNote 7.3.2

Depends R (>= 3.5.0), foreach

NeedsCompilation no

Author Auriane Gabaut [aut, cre],
Ariane Bercu [aut],
Mélanie Prague [aut],
Cécile Proust-Lima [aut]

Repository CRAN

Date/Publication 2025-06-27 12:50:06 UTC

Contents

`AIC.remix` *AIC for remix object*

Description

Computes akaike information criterion from the output of `remix` as

$$AIC = -2\mathcal{LL}_y(\hat{\theta},\hat{\alpha}) + k \times P$$

where P is the total number of parameters estimated and $\mathcal{LL}_y(\hat{\theta},\hat{\alpha})$ the log-likelihood of the model.

Usage

```
## S3 method for class 'remix'
AIC(object, ..., k)
```

Arguments

`object`	output of `remix`.
`...`	additional arguments.
`k`	numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC.

Value

AIC.

References

Akaike, H. 1998. Information theory and an extension of the maximum likelihood principle, Selected papers of hirotugu akaike, 199-213. New York: Springer.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)
lambda = 1440

res = remix(project = project,
            dynFUN = dynFUN_demo,
            y = y,
            ObsModel.transfo = ObsModel.transfo,
            alpha = alpha,
            selfInit = TRUE,
            eps1=10**(-2),
            eps2=1,
            lambda=lambda)

AIC(res)

## End(Not run)
```

`BIC.remix` *BIC for remix object*

Description

Computes bayesian information criterion from the output of `remix` as

$$BIC = -2\mathcal{LL}_y(\hat{\theta}, \hat{\alpha}) + \log(N)P$$

where P is the total number of parameters estimated, N the number of subject and $\mathcal{LL}_y(\hat{\theta}, \hat{\alpha})$ the log-likelihood of the model.

Usage

```
## S3 method for class 'remix'
BIC(object, ...)
```

Arguments

`object`	output of `remix`.
`...`	additional arguments.

Value

BIC.

References

Schwarz, G. 1978. Estimating the dimension of a model. The annals of statistics 6 (2): 461-464

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)
lambda = 1440

res = remix(project = project,
            dynFUN = dynFUN_demo,
            y = y,
            ObsModel.transfo = ObsModel.transfo,
            alpha = alpha,
            selfInit = TRUE,
            eps1=10**(-2),
            eps2=1,
            lambda=lambda)

BIC(res)

## End(Not run)
```

`BICc` *BICc*

Description

Computes corrected bayesian information criterion as

$$BICc = -2\mathcal{LL}_y(\hat{\theta},\hat{\alpha}) + P_R \log(N) + P_F \log(n_{tot})$$

where P_F is the total number of parameters linked to fixed effects, P_R to random effects, N the number of subject, n_tot the total number of observations and $\mathcal{LL}_y(\hat{\theta},\hat{\alpha})$ the log-likelihood of the model.

Usage

```
BICc(object, ...)
```

Arguments

`object`	output of `remix` or `cv.remix`
`...`	opptional additional arguments.

Value

BICc.

References

Delattre M, Lavielle M, Poursat M-A. A note on BIC in mixed-effects models. Elect J Stat. 2014; 8(1): 456-475.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)
lambda = 1440

res = remix(project = project,
            dynFUN = dynFUN_demo,
            y = y,
            ObsModel.transfo = ObsModel.transfo,
            alpha = alpha,
            selfInit = TRUE,
            eps1=10**(-2),
            eps2=1,
            lambda=lambda)

BICc(res)

## End(Not run)
```

---

computeFinalTest *Compute final estimation*

Description

Computes a final saem and wald test if ‘test‘ on the final model found by remix algorithm.

Usage

```
computeFinalTest(
  remix.output,
  dynFUN,
  y,
  ObsModel.transfo,
  final.project = NULL,
  pop.set = NULL,
  prune = NULL,
  n = NULL,
  parallel = TRUE,
  ncores = NULL,
  print = TRUE,
  digits = 3,
  trueValue = NULL,
  test = TRUE,
  p.max = 0.05
)
```

Arguments

remix.output	a remix outputs. It’s important that the project path of this outputs is still existing.
dynFUN	function computing the dynamics of interest for a set of parameters. This function need to contain every sub-function that it may needs (as it is called in a foreach loop). The output of this function need to return a data.frame with time as first columns and named dynamics in other columns. It must take in input :

	• `y` a named vector with the initial condition. The names are the dynamics names. • `parms` a named vector of parameter. • `time` vector a timepoint. See `dynFUN_demo`, `model.clairon`, `model.pasin` or `model.pk` for examples.
`y`	initial condition of the mechanism model, conform to what is asked in dynFUN.
`ObsModel.transfo`	list containing two lists of transformations and two vectors linking each transformations to their observation model name in the Monolix project. The list should include identity transformations and be named `S` and `R`. The two vectors should be named `linkS` and `linkR`. Both `S` (for the direct observation models) and `linkS`, as well as `R` (for latent process models) and `linkR`, must have the same length. • `S`: a list of transformations for the direct observation models. Each transformation corresponds to a variable $Y_p = h_p(S_p)$, where the name indicates which dynamic is observed (from `dynFUN`); • `linkS` : a vector specifying the observation model names (that is used in the monolix project, `alpha1`, etc.) for each transformation, in the same order as in `S`; • `R`: similarly, a list of transformations for the latent process models. Although currently there is only one latent dynamic, each $s_k, k \leq K$ transformation corresponds to the same dynamic but may vary for each Y_k observed. The names should match the output from `dynFUN`; • `linkR` : a vector specifying the observation model names for each transformation, in the same order as in `R`.
`final.project`	directory of the final Monolix project (default add "_upd" to the Monolix project).
`pop.set`	population parameters setting for final estimation (see details).
`prune`	percentage for prunning ($\in [0;1]$) in the Adaptative Gauss-Hermite algorithm used to compute the log-likelihood and its derivates (see `gh.LL`).
`n`	number of points for gaussian quadrature (see `gh.LL`).
`parallel`	logical, if the computation should be done in parallel when possible (default TRUE).
`ncores`	number of cores for parallelization (default NULL and `detectCores` is used).
`print`	logical, if the results and algotihm steps should be displayed in the console (default to TRUE).
`digits`	number of digits to print (default to 3).
`trueValue`	-for simulation purposes- named vector of true value for parameters.
`test`	if Wald test should be computed at the end of the iteration.
`p.max`	maximum value to each for wald test p.value (default 0.05).

Details

For population parameter estimation settings, see (<https://monolixsuite.slp-software.com/r-functions/2024R1/setpopulation

Value

a remix object on which final SAEM and test, if `test` is `TRUE`, have been computed.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)

res = cv.remix(project = project,
               dynFUN = dynFUN_demo,
               y = y,
               ObsModel.transfo = ObsModel.transfo,
               alpha = alpha,
               selfInit = TRUE,
               eps1=10**(-2),
               ncores=8,
               nlambda=8,
               eps2=1)

res_with_test = computeFinalTest(retrieveBest(res0,criterion=BICc),
                                 dynFUN_demo,
                                 y,
                                 ObsModel.transfo)

## End(Not run)
```

`cv.remix` *REMixed algorithm over a grid of λ*

Description

Regularization and Estimation in MIXed effects model, over a regularization path.

Usage

```
cv.remix(
  project = NULL,
  final.project = NULL,
  dynFUN,
  y,
  ObsModel.transfo,
  alpha,
  lambda.grid = NULL,
  alambda = 0.001,
  nlambda = 50,
  lambda_max = NULL,
  eps1 = 10^(-2),
  eps2 = 10^(-1),
  selfInit = FALSE,
  pop.set1 = NULL,
  pop.set2 = NULL,
  prune = NULL,
  n = NULL,
  parallel = TRUE,
  ncores = NULL,
  print = TRUE,
  digits = 3,
  trueValue = NULL,
  unlinkBuildProject = TRUE,
  max.iter = +Inf
)
```

Arguments

`project`	directory of the Monolix project (in .mlxtran). If NULL, the current loaded project is used (default is NULL).
`final.project`	directory of the final Monolix project (default add "_upd" to the Monolix project).
`dynFUN`	function computing the dynamics of interest for a set of parameters. This function need to contain every sub-function that it may needs (as it is called in a `foreach` loop). The output of this function need to return a data.frame with `time` as first columns and named dynamics in other columns. It must take in input : `y` a named vector with the initial condition. The names are the dynamics names. `parms` a named vector of parameter. `time` vector a timepoint. See `dynFUN_demo`, `model.clairon`, `model.pasin` or `model.pk` for examples.
`y`	initial condition of the mechanism model, conform to what is asked in dynFUN.
`ObsModel.transfo`	list containing two lists of transformations and two vectors linking each transformations to their observation model name in the Monolix project. The list should include identity transformations and be named `S` and `R`. The two vectors should be named `linkS` and `linkR`. Both `S` (for the direct observation models) and `linkS`, as well as `R` (for latent process models) and `linkR`, must have the same length.

- `S` a list of transformations for the direct observation models. Each transformation corresponds to a variable $Y_p = h_p(S_p)$, where the name indicates which dynamic is observed (from `dynFUN`);
- `linkS` a vector specifying the observation model names (that is used in the monolix project, `alpha1`, etc.) for each transformation, in the same order as in `S`;
- `R` similarly, a list of transformations for the latent process models. Although currently there is only one latent dynamic, each $s_k, k \leq K$ transformation corresponds to the same dynamic but may vary for each Y_k observed. The names should match the output from `dynFUN`;
- `linkR` a vector specifying the observation model names for each transformation, in the same order as in `R`.

`alpha`	named list of named vector "`alpha0`", "`alpha1`" (all `alpha1` are mandatory). The name of `alpha$alpha0` and `alpha$alpha1` are the observation model names from the monolix project to which they are linked (if the observations models are defined whithout intercept, alpha$alpha0 need to be set to the vector NULL).
`lambda.grid`	grid of user-suuplied penalisation parameters for the lasso regularization (if NULL, the sequence is computed based on the data).
`alambda`	if `lambda.grid` is null, coefficients used to compute the grid (default to 0.05, see details).
`nlambda`	if `lambda.grid` is null, number of lambda parameter to test (default to 50).
`lambda_max`	if `lambda.grid` is null, maximum of the lambda grid to test (default is automatically computed, see details)
`eps1`	integer (>0) used to define the convergence criteria for the regression parameters.
`eps2`	integer (>0) used to define the convergence criteria for the likelihood.
`selfInit`	logical, if the SAEM is already done in the monolix project should be use as the initial point of the algorithm (if FALSE, SAEM is automatically compute according to `pop.set1` settings ; if TRUE, a SAEM through monolix need to have been launched).
`pop.set1`	population parameters setting for initialisation (see details).
`pop.set2`	population parameters setting for iterations.
`prune`	percentage for prunning ($\in [0;1]$) in the Adaptative Gauss-Hermite algorithm used to compute the log-likelihood and its derivates (see `gh.LL`).
`n`	number of points for gaussian quadrature (see `gh.LL`).
`parallel`	logical, if the computation should be done in parallel when possible (default TRUE).
`ncores`	number of cores for parallelization (default NULL and `detectCores` is used).
`print`	logical, if the results and algotihm steps should be displayed in the console (default to TRUE).
`digits`	number of digits to print (default to 3).
`trueValue`	-for simulation purposes- named vector of true value for parameters.
`unlinkBuildProject`	logical, if the build project of each lambda should be deleted.
`max.iter`	maximum number of iteration (default 20).

Details

See REMixed-package for details on the model. For each $\lambda \in \Lambda$, the remix is launched. For population parameter estimation settings, see (<https://monolixsuite.slp-software.com/r-functions/2024R1/setpopulationparametere

Value

A list of outputs of the final project and of the iterative process over each value of `lambda.grid`:

`info` Information about the parameters.

`project` The project path if not unlinked.

`lambda` The grid of λ.

`BIC` Vector of BIC values for the model built over the grid of λ.

`BICc` Vector of BICc values for the model built over the grid of λ.

`LL` Vector of log-likelihoods for the model built over the grid of λ.

`LL.pen` Vector of penalized log-likelihoods for the model built over the grid of λ.

`res` List of all REMixed results for each λ (see remix).

`outputs` List of all REMixed outputs for each λ (see remix).

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)

res = cv.remix(project = project,
               dynFUN = dynFUN_demo,
               y = y,
               ObsModel.transfo = ObsModel.transfo,
               alpha = alpha,
               selfInit = TRUE,
               eps1=10**(-2),
               ncores=8,
               nlambda=8,
               eps2=1)

## End(Not run)
```

`dynFUN_demo` *Dynamic functions demo*

Description

Example of solver for `remix` and `cv.remix` algorithm. It is perfectly adapted for the Monolix demo project (see `getMLXdir`).

Usage

```
dynFUN_demo
```

Format

`dynFUN_demo` function of `t`, `y`, `parms` :

`t` vector of timepoint.

`y` initial condition, named vector of form `c(AB=<...>,S=<...>)`.

`parms` named vector of model parameter ; should contain `phi_S`,`delta_AB`,`delta_S`.

Details

Suppose you have antibodies secreting cells -S- that produces antibodies -AB- at rate φ_S. These two biological entities decay respectively at rate δ_S and δ_{AB}. The biological mechanism behind is :

$$\begin{cases} \frac{d}{dt}S(t) & = & -\delta_S S(t) \\ \frac{d}{dt}AB(t) & = & \varphi_S S(t) - \delta_{AB}AB(t) \\ (S(0), AB(0)) & = & (S_0, AB_0) \end{cases}$$

References

Pasin C, Balelli I, Van Effelterre T, Bockstal V, Solforosi L, Prague M, Douoguih M, Thiébaut R, for the EBOVAC1 Consortium. 2019. Dynamics of the humoral immune response to a prime-boost Ebola vaccine: quantification and sources of variation. J Virol 93 : e00579-19. https://doi.org/10.1128/JVI.00579-19

See Also

`model.pasin`, `getMLXdir`.

Examples

```
t = seq(0,300,1)
y =c(AB=1000,S=5)
parms = c(phi_S = 611, delta_AB = 0.03, delta_S=0.01)

res <- dynFUN_demo(t,y,parms)
```

```
plot(res[,"time"],
     log10(res[,"AB"]),
     ylab="log10(AB(t))",
     xlab="time (days)",
     main="Antibody titer over the time",
     type="l")

plot(res[,"time"],
     res[,"S"],
     ylab="S(t)",
     xlab="time (days)",
     main="Antibody secreting cells quantity over time",
     type="l")
```

eBIC *eBIC*

Description

Computes extended bayesian information criterion as

$$eBIC = -2\mathcal{LL}_y(\hat{\theta},\hat{\alpha}) + P\log(N) + 2\gamma\log(\binom{(}{k}, K))$$

where P is the total number of parameters estimated, N the number of subject, $\mathcal{LL}_y(\hat{\theta},\hat{\alpha})$ the log-likelihood of the model, K the number of submodel to explore (here the numbre of biomarkers tested) and k the numbre of biomarkers selected in the model.

Usage

```
eBIC(object, ...)
```

Arguments

`object`	output of `remix` or `cv.remix`.
`...`	opptional additional arguments.

Value

eBIC.

References

Chen, J. and Z. Chen. 2008. Extended Bayesian information criteria for model selection with large model spaces. Biometrika 95 (3): 759-771.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)
lambda = 1440

res = remix(project = project,
            dynFUN = dynFUN_demo,
            y = y,
            ObsModel.transfo = ObsModel.transfo,
            alpha = alpha,
            selfInit = TRUE,
            eps1=10**(-2),
            eps2=1,
            lambda=lambda)

eBIC(res)

## End(Not run)
```

`extract` *extract remix results from cvRemix object*

Description

Extracts a build from a cvRemix object.

Usage

```
extract(fit, n)
```

Arguments

`fit`	output of `cv.remix`;
`n`	rank (in the ‘fit$lambda‘) to extract.

Value

outputs from `remix` algorithm of rank ‘n‘ computed by `cv.remix`.

See Also

cv.remix, remix.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)

cv.outputs = cv.Remix(project = project,
            dynFUN = dynFUN_demo,
            y = y,
            ObsModel.transfo = ObsModel.transfo,
            alpha = alpha,
            selfInit = TRUE,
            eps1=10**(-2),
            ncores=8,
            eps2=1)

res <- extract(cv.outputs,6)

plotConvergence(res)

trueValue = read.csv(paste0(dirname(project),"/demoSMLX/Simulation/populationParameters.txt"))


plotSAEM(res,paramToPlot = c("delta_S_pop","phi_S_pop","delta_AB_pop"),trueValue=trueValue)

## End(Not run)
```

---

getMLXdir *Get monolix demo project path*

Description

Get monolix demo project path

Usage

```
getMLXdir()
```

Value

path to the monolix demo from REMix package.

See Also

dynFUN_demo.

Examples

```
print(getMLXdir())
```

---

gh.LL *Adaptive Gauss-Hermite approximation of log-likelihood derivatives*

Description

Computes Adaptive Gauss-Hermite approximation of the log-likelihood and its derivatives in NLMEM with latent observation processes, see REMixed-package for details on the model.

Usage

```
gh.LL(
  dynFUN,
  y,
  mu = NULL,
  Omega = NULL,
  theta = NULL,
  alpha1 = NULL,
  covariates = NULL,
  ParModel.transfo = NULL,
  ParModel.transfo.inv = NULL,
  Sobs = NULL,
  Robs = NULL,
  Serr = NULL,
  Rerr = NULL,
  ObsModel.transfo = NULL,
  data = NULL,
  n = NULL,
  prune = NULL,
  parallel = TRUE,
  ncores = NULL,
  onlyLL = FALSE,
  verbose = TRUE
)
```

Arguments

`dynFUN`	function computing the dynamics of interest for a set of parameters. This function need to contain every sub-function that it may needs (as it is called in a foreach loop). The output of this function need to return a data.frame with `time` : as first columns and named dynamics in other columns. It must take in input : • `y` : a named vector with the initial condition. The names are the dynamics names. • `parms` : a named vector of parameter. • `time` : vector a timepoint. See `dynFUN_demo`, `model.clairon`, `model.pasin` or `model.pk` for examples.
`y`	initial condition of the mechanism model, conform to what is asked in `dynFUN`.
`mu`	list of individuals random effects estimation (vector of r.e. need to be named by the parameter names), use to locate the density mass; (optional, see description).
`Omega`	list of individuals estimated standard deviation diagonal matrix (matrix need to have rows and columns named by the parameter names), use to locate the density mass; (optional, see description).
`theta`	list of model parameters containing (see details) • `phi_pop` : named vector with the population parameters with no r.e. $(\phi_{l\ pop})_{l\leq L}$ (NULL if none) ; • `psi_pop` : named vector with the population parameters with r.e. $(\psi_{l\ pop})_{l\leq m}$; • `gamma` : named list (for each parameters) of named vector (for each covariates) of covariate effects from parameters with no r.e. ; • `beta` : named list (for each parameters) of named vector (for each covariates) of covariate effects from parameters with r.e.. • `alpha0` : named vector of $(\alpha_{0k})_{k\leq K}$ parameters (names are identifier of the observation model, such as in a Monolix project); • `omega` : named vector of estimated r.e. standard deviation; (optional, see description).
`alpha1`	named vector of regulatization parameters $(\alpha_{1k})_{k\leq K}$, with identifier of observation model as names, (optional, see description).
`covariates`	matrix of individual covariates (size N x n). Individuals must be sorted in the same order than in `mu` and `Omega`, (optional, see description).
`ParModel.transfo`	named list of transformation functions $(h_l)_{l\leq m}$ and $(s_k)_{k\leq K}$ for the individual parameter model (names must be consistent with `phi_pop` and `psi_pop`, missing entries are set by default to the identity function ; optional, see description).
`ParModel.transfo.inv`	Named list of inverse transformation functions for the individual parameter model (names must be consistent with `phi_pop` and `psi_pop` ; optional, see description).
`Sobs`	list of individuals trajectories for the direct observation models $(Y_{pi})_{p\leq P,i\leq N}$. Each element $i \leq N$ of the list, is a list of $p \leq P$ data.frame with time $(t_{pij})_{j\leq n_{ip}}$ and observations $(Y_{pij})_{j\leq n_{ip}}$. Each data.frame is named with the observation model identifiers.

`Robs`	list of individuals trajectories for the latent observation models $(Z_{ki})_{k\leq K, i\leq N}$. Each element $i \leq N$ of the list, is a list of $k \leq K$ data.frame with time $(t_{kij})_{j\leq n_{ik}}$ and observations $(Z_{kij})_{j\leq n_{ik}}$. Each data.frame is named with the observation model identifiers.
`Serr`	named vector of the estimated error mocel constants $(\varsigma_p)_{p\leq P}$ with observation model identifiers as names.
`Rerr`	named vector of the estimated error mocel constants $(\sigma_k)_{k\leq K}$ with observation model identifiers as names.
`ObsModel.transfo`	list containing two lists of transformations and two vectors linking each transformations to their observation model name in the Monolix project. The list should include identity transformations and be named `S` and `R`. The two vectors should be named `linkS` and `linkR`. Both `S` (for the direct observation models) and `linkS`, as well as `R` (for latent process models) and `linkR`, must have the same length. • `S`: a list of transformations for the direct observation models. Each transformation corresponds to a variable $Y_p = h_p(S_p)$, where the name indicates which dynamic is observed (from `dynFUN`); • `linkS` : a vector specifying the observation model names (that is used in the monolix project, `alpha1`, etc.) for each transformation, in the same order as in `S`; • `R`: similarly, a list of transformations for the latent process models. Although currently there is only one latent dynamic, each $s_k, k \leq K$ transformation corresponds to the same dynamic but may vary for each Y_k observed. The names should match the output from `dynFUN`; • `linkR` : a vector specifying the observation model names for each transformation, in the same order as in `R`.
`data`	output from `readMLX` containing parameters "`mu`", "`Omega`", "`theta`", "`alpha1`", "`covariates`", "`ParModel.transfo`", "`ParModel.transfo.inv`", "`Sobs`", "`Robs`", "`Serr`", "`Rerr`", "`ObsModel.transfo`" extract from a monolix project.
`n`	number of points per dimension to use for the Gauss-Hermite quadrature rule.
`prune`	integer between 0 and 1, percentage of pruning for the Gauss-Hermite quadrature rule (default NULL).
`parallel`	logical, if computation should be done in parallel.
`ncores`	number of cores to use for parallelization, default will detect the number of cores available.
`onlyLL`	logical, if only the log-likelihood should be computed (and not $\partial_{\alpha_1} LL$ or $\partial^2_{\alpha_1} LL$).
`verbose`	logical, if progress bar should be printed through the computation.

Details

Based on notation introduced `REMixed-package`. The log-likelihood of the model $LL(\theta, \alpha_1)$ for a set of population parameters θ and regulatization parameters α_1 is estimated using Adaptative Gausse-Hermite quadrature, using conditional distribution estimation to locate the mass of the integrand. If the project has been initialized as a Monolix project, the user can use `readMLX` function to retrieve all the project information needed here.

Value

A list with the approximation by Gauss-Hermite quadrature of the likelihood `L`, the log-likelihood `LL`, the gradient of the log-likelihood `dLL`, and the Hessian of the log-likelihood `ddLL` at the point θ, α provided.

Examples

```
## Not run:
project <- getMLXdir()


ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

data <- readMLX(project,ObsModel.transfo,alpha)

LL <- gh.LL(dynFUN = dynFUN_demo,
            y = c(S=5,AB=1000),
            ObsModel.transfo=ObsModel.transfo,
            data = data)

print(LL)

## End(Not run)
```

`indParm` *Generate individual parameters*

Description

Generate the individual parameters of indivual whose covariates are `covariates` and random effects `eta_i`.

Usage

```
indParm(theta, covariates, eta_i, transfo, transfo.inv)
```

Arguments

`theta`	list with at least `phi_pop`, `psi_pop`, `gamma`, `beta` (named ; corresponding to the model parameter ϕ_{pop}, ψ_{pop}, γ, β) : • `phi_pop` named vector of population parameters without r.e ; • `psi_pop` named vector of population parameters with r.e ;

- gamma named list of vector of covariates effects for phi_pop parameters, if NULL no covariates effect on parameters. ;
- beta named list of vector of covariates effects for each psi_pop, if NULL no covariates effect on parameters.

covariates	line data.frame of individual covariates ;
eta_i	named vector of random effect for each psi parameter ;
transfo	named list of transformation functions $(h_l)_{l\leq m}$ and $(s_k)_{k\leq K}$ for the individual parameter model (names must be consistent with phi_pop and psi_pop, missing entries are set by default to the identity function).
transfo.inv	amed list of inverse transformation functions for the individual parameter model (names must be consistent withphi_pop andpsi_pop).

Details

The models used for the parameters are :

$$h_l(\psi_{li}) = h_l(\psi_{lpop}) + X_i\beta_l + \eta_{li}$$

with h_l the transformation, β_l the vector of covariates effect and with η_i the random effects associated ψ_l parameter ;

$$g_k(\phi_{ki}) = g_k(\phi_{kpop}) + X_i\gamma_l$$

with g_k the transformation and γ_k the vector of covariates effect associated ϕ_k parameter.

Value

a list with phi_i and psi_i parameters.

See Also

model.clairon, model.pasin.

Examples

```
phi_pop = c(delta_S = 0.231, delta_L = 0.000316)
psi_pop = c(delta_Ab = 0.025,phi_S = 3057, phi_L = 16.6)
gamma = NULL
covariates = data.frame(cAGE = runif(1,-15,15), G1 = rnorm(1), G2 = rnorm(1))
beta = list(delta_Ab=c(0,1.2,0),phi_S = c(0.93,0,0),phi_L=c(0,0,0.8))

theta=list(phi_pop = phi_pop,psi_pop = psi_pop,gamma = gamma, beta = beta)
eta_i = c(delta_Ab = rnorm(1,0,0.3),phi_S=rnorm(1,0,0.92),phi_L=rnorm(1,0,0.85))
transfo = list(delta_Ab=log,phi_S=log,phi_L=log)
transfo.inv = list(delta_Ab = exp,phi_S=exp,phi_L=exp)

indParm(theta,covariates,eta_i,transfo,transfo.inv)
```

initStrat *Initialization strategy*

Description

Selecting an initialization point by grouping biomarkers of project and running the SAEM. Initial condition is then selected using maximum log-likelihood.

Usage

```
initStrat(
  project,
  alpha,
  ObsModel.transfo,
  Nb_genes = 2,
  trueValue = NULL,
  pop.set = NULL,
  useSettingsInAPI = FALSE,
  conditionalDistributionSampling = FALSE,
  print = TRUE,
  digits = 2,
  unlinkTemporaryProject = TRUE,
  seed = NULL
)
```

Arguments

`project` directory of the Monolix project (in .mlxtran). If NULL, the current loaded project is used (default is NULL).

`alpha` named list of named vector "`alpha0`", "`alpha1`" (all `alpha1` are mandatory). The name of `alpha$alpha0` and `alpha$alpha1` are the observation model names from the monolix project to which they are linked (if the observations models are defined whithout intercept, alpha$alpha0 need to be set to the vector NULL).

`ObsModel.transfo`
list containing two lists of transformations and two vectors linking each transformations to their observation model name in the Monolix project. The list should include identity transformations and be named `S` and `R`. The two vectors should be named `linkS` and `linkR`.

Both `S` (for the direct observation models) and `linkS`, as well as `R` (for latent process models) and `linkR`, must have the same length.

- `S`: a list of transformations for the direct observation models. Each transformation corresponds to a variable $Y_p = h_p(S_p)$, where the name indicates which dynamic is observed (from `dynFUN`);
- `linkS` : a vector specifying the observation model names (that is used in the monolix project, `alpha1`, etc.) for each transformation, in the same order as in `S`;

- R: similarly, a list of transformations for the latent process models. Although currently there is only one latent dynamic, each $s_k, k \leq K$ transformation corresponds to the same dynamic but may vary for each Y_k observed. The names should match the output from dynFUN;
- linkR : a vector specifying the observation model names for each transformation, in the same order as in R.

Nb_genes	Size of group of genes.
trueValue	-for simulation purposes- named vector of true value for parameters.
pop.set	population parameters setting for initialization (see details).
useSettingsInAPI	logical, if the settings for SAEM should not be changed from what is currently set in the project.
conditionalDistributionSampling	logical, if conditional distribution estimation should be done on the final project.
print	logical, if the results and algotihm steps should be displayed in the console (default to TRUE).
digits	number of digits to print (default to 2).
unlinkTemporaryProject	If temporary project (of genes group) is deleted (defaut: TRUE)
seed	value of the seed used to initialize the group (see set.seed).

Details

For population parameter estimation settings, see (<https://monolixsuite.slp-software.com/r-functions/2024R1/setpopulationp

Value

a list of outputs for every group of genes tested with composition of the group, final parameter estimates, final scores estimates (OFV, AIC, BIC, BICc), temporary project directory. The final selected set is initialize in the project.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

initStrat(project,alpha,ObsModel.transfo,seed=1710)

## End(Not run)
```

`model.clairon` *Model from Clairon and al.,2023*

Description

Generates the dynamics of antibodies secreting cells -S- that produces antibodies -AB- over time, with two injection of vaccine at time $t_0 = 0$ and t_{inj}, using Clairon and al., 2023, model.

Usage

```
model.clairon(t, y, parms, tinj = 21)
```

Arguments

`t`	vector of timepoint.
`y`	initial condition, named vector of form c(S=S0,Ab=A0).
`parms`	named vector of model parameter (should contain "`fM2`","`theta`","`delta_S`","`delta_Ab`","`delta_V`").
`tinj`	time of injection (default to 21).

Details

Model is defined as

$$\begin{cases} \frac{d}{dt}S(t) & = & f_{\overline{M}_k}e^{-\delta_V(t-t_k)} - \delta_S S(t) \\ \frac{d}{dt}Ab(t) & = & \theta S(t) - \delta_{Ab}Ab(t) \end{cases}$$

on each interval $I_1 = [0; t_{inj}[$ and $I_2 = [t_{inj}; +\infty[$. For each interval I_k, we have t_k corresponding to the last injection date of vaccine, such that $t_1 = 0$ and $t_2 = t_{inj}$. By definition, $f_{\overline{M}_1} = 1$ (Clairon and al., 2023).

Value

Matrix of time and observation of antibody secreting cells S and antibody titer Ab.

References

Quentin Clairon, Melanie Prague, Delphine Planas, Timothee Bruel, Laurent Hocqueloux, et al. Modeling the evolution of the neutralizing antibody response against SARS-CoV-2 variants after several administrations of Bnt162b2. 2023. hal-03946556

See Also

`indParm`

Examples

```
y = c(S=1,Ab=0)

parms = c(fM2 = 4.5,
          theta = 18.7,
          delta_S = 0.01,
          delta_Ab = 0.23,
          delta_V = 2.7)

t = seq(0,35,1)

res <- model.clairon(t,y,parms)

plot(res)
```

`model.pasin` *Model from Pasin and al.,2019*

Description

Generate trajectory of the Humoral Immune Response to a Prime-Boost Ebola Vaccine.

Usage

```
model.pasin(t, y, parms)
```

Arguments

`t`	vector of time ;
`y`	initial condition, named vector of form `c(Ab=<...>,S=<...>,L=<...>)` ;
`parms`	named vector of model parameter ; should contain `"theta_S","theta_L","delta_Ab","delta_S","delta_`

Details

The model correspond to the dynamics of the humoral response, from 7 days after the boost immunization with antibodies secreting cells -S and L, characterized by their half lives- that produces antibodies -AB- at rate θ_S and θ_L. All these biological entities decay at rate repectively δ_S, δ_L and δ_{Ab}. Model is then defined as

$$\begin{cases} \frac{d}{dt}Ab(t) &=& \theta_S S(t) + \theta_L L(t) - \delta_{Ab} Ab(t) \\ \frac{d}{dt}S(t) &=& -\delta_S S(t) \\ \frac{d}{dt}L(t) &=& -\delta_L L(t) \end{cases}$$

Value

Matrix of `time` and observation of antibody titer `Ab`, and ASCs `S` and `L`.

References

Pasin C, Balelli I, Van Effelterre T, Bockstal V, Solforosi L, Prague M, Douoguih M, Thiébaut R, for the EBOVAC1 Consortium. 2019. Dynamics of the humoral immune response to a prime-boost Ebola vaccine: quantification and sources of variation. J Virol 93: e00579-19. https://doi.org/10.1128/JVI.00579-19

See Also

`indParm`, `model.clairon`.

Examples

```
y = c(Ab=0,S=5,L=5)
parms = c(theta_S = 611,
          theta_L = 3.5,
          delta_Ab = 0.025,
          delta_S = 0.231,
          delta_L = 0.000152)

t = seq(0,100,5)
res <- model.pasin(t,y,parms)
plot(res)
```

`model.pk` *Generate trajectory of PK model*

Description

The administration is via a bolus. The PK model has one compartment (volume V) and a linear elimination (clearance Cl). The parameter ka is defined as $ka = \frac{Cl}{V}$.

Usage

```
model.pk(t, y, parms)
```

Arguments

`t`	vector of time ;
`y`	initial condition, named vector of form c(C=C0) ;
`parms`	named vector of model parameter ; should contain either "`Cl`" and "`V`" or "`ka`".

Value

Matrix of time and observation of Concentration C.

See Also

indParm.

Examples

```
res <- model.pk(seq(0,30,1),c(C=100),parms=c(ka=1))

plot(res)
```

---

plot.cvRemix *Plot of cv.remix object*

Description

Calibration plot for cvRemix object.

Usage

```
## S3 method for class 'cvRemix'
plot(x, criterion = BICc, trueValue = NULL, ...)
```

Arguments

x	output of cv.remix.
criterion	which criterion function to take into account. Default is the function 'BICc", but one can use 'BIC', 'AIC', 'eBIC' or any function depending on a ‘cvRemix‘ object.
trueValue	-for simulation purposes- named vector of true value for parameters.
...	opptional additional arguments.

Value

A plot.

See Also

cv.remix

`plotCalibration` *Calibration plot*

Description

Calibration plot

Usage

```
plotCalibration(
  fit,
  legend.position = "none",
  trueValue = NULL,
  criterion = BICc,
  dismin = TRUE
)
```

Arguments

`fit`	fit object of class cvRemix, from `cv.remix`.
`legend.position`	(default NULL) the default position of legends ("none", "left", "right", "bottom", "top", "inside").
`trueValue`	(for simulation purpose) named vector containing the true value of regularization parameter.
`criterion`	function ; which criterion among 'BIC', 'eBIC', 'AIC', 'BICc', or function of cvRemix object to take into account (default : BICc).
`dismin`	logical ; if minimizers of information criterion should be display.

Value

Calibration plot, over the lambda.grid.

See Also

`remix`, `cv.remix`.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))
```

```
alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)

res = cv.remix(project = project,
               dynFUN = dynFUN_demo,
               y = y,
               ObsModel.transfo = ObsModel.transfo,
               alpha = alpha,
               selfInit = TRUE,
               eps1=10**(-2),
               ncores=8,
               nlambda=8,
               eps2=1)

plotCalibration(res)

plotIC(res)

## End(Not run)
```

`plotConvergence` *Log-likelihood convergence*

Description

Log-likelihood convergence

Usage

```
plotConvergence(fit, ...)
```

Arguments

`fit`	fit object of class remix, from `remix` or a certain build from `cv.remix` output.
`...`	opptional additional arguments.

Value

Log-Likelihood values throughout the algorithm iteration.

See Also

`remix`, `cv.remix`.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)
lambda = 1440

res = remix(project = project,
            dynFUN = dynFUN,
            y = y,
            ObsModel.transfo = ObsModel.transfo,
            alpha = alpha,
            selfInit = TRUE,
            eps1=10**(-2),
            eps2=1,
            lambda=lambda)

plotConvergence(res)

trueValue = read.csv(paste0(dirname(project),"/demoSMLX/Simulation/populationParameters.txt"))#'

plotSAEM(res,paramToPlot = c("delta_S_pop","phi_S_pop","delta_AB_pop"),trueValue=trueValue)

## End(Not run)
```

---

plotIC *IC plot*

Description

IC plot

Usage

```
plotIC(fit, criterion = BICc, dismin = TRUE)
```

Arguments

fit	fit object of class cvRemix, from cv.remix;
criterion	which criterion among 'BICc', 'BIC', 'AIC' or 'eBIC' to take into account (default: BICc);
dismin	logical ; if minimizers of information criterion should be display.

Value

IC trhoughout the lambda.grid.

See Also

remix, cv.remix.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)

res = cv.remix(project = project,
               dynFUN = dynFUN_demo,
               y = y,
               ObsModel.transfo = ObsModel.transfo,
               alpha = alpha,
               selfInit = TRUE,
               eps1=10**(-2),
               ncores=8,
               nlambda=8,
               eps2=1)

plotCalibration(res)

plotIC(res)

## End(Not run)
```

---

plotInit *Plot initialization*

Description

Plot initialization

Usage

```
plotInit(init, alpha = NULL, trueValue = NULL)
```

Arguments

`init`	outputs from `initStrat` function.
`alpha`	named list of named vector "`alpha0`", "`alpha1`" (all `alpha1` are mandatory). The name of `alpha$alpha0` and `alpha$alpha1` are the observation model names from the monolix project to which they are linked (if the observations models are defined whithout intercept, alpha$alpha0 need to be set to the vector NULL).
`trueValue`	(for simulation purpose) named vector containing the true value of regularization parameter.

Value

log-likelihood value for all groups of genes tested.

See Also

`initStrat`.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

init <- initStrat(project,alpha,ObsModel.transfo,seed=1710)

plotInit(init)

## End(Not run)
```

plotSAEM *Display the value of parameters at each iteration*

Description

Display the value of parameters at each iteration

Usage

```
plotSAEM(fit, paramToPlot = "all", trueValue = NULL)
```

Arguments

`fit`	object of class remix, from `remix` or a certain build from `cv.remix` output.
`paramToPlot`	Population parameters to plot (which have been estimated by SAEM) ;
`trueValue`	(for simulation purpose) vector named of true values ;

Value

For each parameters, the values at the end of each iteration of remix algorithm is drawn. Moreover, the SAEM steps of each iteration are displayed.

See Also

`remix`, `cv.remix`.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)
lambda = 1440

res = remix(project = project,
            dynFUN = dynFUN_demo,
            y = y,
            ObsModel.transfo = ObsModel.transfo,
            alpha = alpha,
            selfInit = TRUE,
            eps1=10**(-2),
            eps2=1,
            lambda=lambda)

plotConvergence(res)

trueValue = read.csv(paste0(dirname(project),"/demoSMLX/Simulation/populationParameters.txt"))

plotSAEM(res,paramToPlot = c("delta_S_pop","phi_S_pop","delta_AB_pop"),trueValue=trueValue)

## End(Not run)
```

readMLX *Extract Data for REMixed Algorithm from a Monolix Project*

Description

This function retrieves all necessary information from a Monolix project file to format the input for the REMixed package. It gathers all relevant data required for the REMix algorithm.

Usage

```
readMLX(project = NULL, ObsModel.transfo, alpha)
```

Arguments

`project` directory of the Monolix project (in .mlxtran). If NULL, the current loaded project is used (default is NULL).

`ObsModel.transfo`

list containing two lists of transformations and two vectors linking each transformations to their observation model name in the Monolix project. The list should include identity transformations and be named `S` and `R`. The two vectors should be named `linkS` and `linkR`.

Both `S` (for the direct observation models) and `linkS`, as well as `R` (for latent process models) and `linkR`, must have the same length.

- `S`: a list of transformations for the direct observation models. Each transformation corresponds to a variable $Y_p = h_p(S_p)$, where the name indicates which dynamic is observed (from `dynFUN`);
- `linkS` : a vector specifying the observation model names (that is used in the monolix project, `alpha1`, etc.) for each transformation, in the same order as in `S`;
- `R`: similarly, a list of transformations for the latent process models. Although currently there is only one latent dynamic, each $s_k, k \leq K$ transformation corresponds to the same dynamic but may vary for each Y_k observed. The names should match the output from `dynFUN`;
- `linkR` : a vector specifying the observation model names for each transformation, in the same order as in `R`.

`alpha` named list of named vector "`alpha0`", "`alpha1`" (all `alpha1` are mandatory). The names of `alpha$alpha0` and `alpha$alpha1` are the observation model names from the monolix project to which they are linked (if the observations models are defined whithout intercept, alpha$alpha0 need to be set to the vector NULL).

Details

To simplify its use, functions `remix`, `cv.remix`, `gh.LL` can be used with arguments `data` rather than all necessary informations "`theta`", "`alpha1`", "`covariates`", "`ParModel.transfo`", "`ParModel.transfo.inv`", "Sobs", "Robs", "Serr", "Rerr", "ObsModel.transfo" that could be extract from a monolix project. If the SAEM task of the project hasn't been launched, it's the initial condition and not the estimated parameters that are returned. If the conditional distribution estimation task has been launched, parameters "mu" and "Omega" are returned too.

Value

A list containing parameters, transformations, and observations from the Monolix project in the format needed for the REMixed algorithm :

- mu list of individuals random effects estimation (vector of r.e. need to be named by the parameter names), use to locate the density mass (if conditional distribution estimation through Monolix has been launched);
- Omega list of individuals estimated standard deviation diagonal matrix (matrix need to have rows and columns named by the parameter names), use to locate the density mass (if conditional distribution estimation through Monolix has been launched);
- theta list of model parameters containing i
  - phi_pop : named vector with the population parameters with no r.e. $(\phi_{l\ pop})_{l\leq L}$ (NULL if none) ;
  - psi_pop : named vector with the population parameters with r.e. $(\psi_{l\ pop})_{l\leq m}$;
  - gamma : named list (for each parameters) of named vector (for each covariates) of covariate effects from parameters with no r.e. ;
  - beta : named list (for each parameters) of named vector (for each covariates) of covariate effects from parameters with r.e..
  - alpha0 : named vector of $(\alpha_{0k})_{k\leq K}$ parameters (names are identifier of the observation model, such as in a Monolix project);
  - omega : named vector of estimated r.e. standard deviation;
- alpha1 named vector of regulatization parameters $(\alpha_{1k})_{k\leq K}$, with identifier of observation model as names;
- covariates matrix of individual covariates (size N x n). Individuals must be sorted in the same order than in mu and Omega;
- ParModel.transfo named list of transformation functions $(h_l)_{l\leq m}$ and $(s_k)_{k\leq K}$ for the individual parameter model (names must be consistent with phi_pop and psi_pop, missing entries are set by default to the identity function ;
- ParModel.transfo.inv named list of inverse transformation functions for the individual parameter model (names must be consistent with phi_pop and psi_pop ;
- Sobs ist of individuals trajectories for the direct observation models $(Y_{pi})_{p\leq P,i\leq N}$. Each element $i\leq N$ of the list, is a list of $p\leq P$ data.frame with time $(t_{pij})_{j\leq n_{ip}}$ and observations $(Y_{pij})_{j\leq n_{ip}}$. Each data.frame is named with the observation model identifiers ;
- Robs list of individuals trajectories for the latent observation models $(Z_{ki})_{k\leq K,i\leq N}$. Each element $i\leq N$ of the list, is a list of $k\leq K$ data.frame with time $(t_{kij})_{j\leq n_{ik}}$ and observations $(Z_{kij})_{j\leq n_{ik}}$. Each data.frame is named with the observation model identifiers ;
- Serr named vector of the estimated error mocel constants $(\varsigma_p)_{p\leq P}$ with observation model identifiers as names ;
- Rerr named vector of the estimated error mocel constants $(\sigma_k)_{k\leq K}$ with observation model identifiers as names ;
- ObsModel.transfo same as inputObsModel.transfo list.

See Also

remix, cv.remix.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

res <- readMLX(project,ObsModel.transfo,alpha)

## End(Not run)
```

---

remix *REMixed algorithm*

Description

Regularization and Estimation in Mixed effects model.

Usage

```
remix(
  project = NULL,
  final.project = NULL,
  dynFUN,
  y,
  ObsModel.transfo,
  alpha,
  lambda,
  eps1 = 10^(-2),
  eps2 = 10^(-1),
  selfInit = FALSE,
  pop.set1 = NULL,
  pop.set2 = NULL,
  pop.set3 = NULL,
  prune = NULL,
  n = NULL,
  parallel = TRUE,
  ncores = NULL,
  print = TRUE,
  verbose = FALSE,
  digits = 3,
  trueValue = NULL,
  finalSAEM = FALSE,
  test = TRUE,
  max.iter = +Inf,
  p.max = 0.05
)
```

Arguments

`project` directory of the Monolix project (in .mlxtran). If NULL, the current loaded project is used (default is NULL).

`final.project` directory of the final Monolix project (default add "_upd" to the Monolix project).

`dynFUN` function computing the dynamics of interest for a set of parameters. This function need to contain every sub-function that it may needs (as it is called in a `foreach` loop). The output of this function need to return a data.frame with `time` as first columns and named dynamics in other columns. It must take in input :

- `y` a named vector with the initial condition. The names are the dynamics names.
- `parms` a named vector of parameter.
- `time` vector a timepoint.

See `dynFUN_demo`, `model.clairon`, `model.pasin` or `model.pk` for examples.

`y` initial condition of the mechanism model, conform to what is asked in dynFUN.

`ObsModel.transfo`

list containing two lists of transformations and two vectors linking each transformations to their observation model name in the Monolix project. The list should include identity transformations and be named `S` and `R`. The two vectors should be named `linkS` and `linkR`.

Both `S` (for the direct observation models) and `linkS`, as well as `R` (for latent process models) and `linkR`, must have the same length.

- `S` a list of transformations for the direct observation models. Each transformation corresponds to a variable $Y_p = h_p(S_p)$, where the name indicates which dynamic is observed (from `dynFUN`);
- `linkS` a vector specifying the observation model names (that is used in the monolix project, `alpha1`, etc.) for each transformation, in the same order as in `S`;
- `R` similarly, a list of transformations for the latent process models. Although currently there is only one latent dynamic, each $s_k, k \leq K$ transformation corresponds to the same dynamic but may vary for each Y_k observed. The names should match the output from `dynFUN`;
- `linkR` a vector specifying the observation model names for each transformation, in the same order as in `R`.

`alpha` named list of named vector "`alpha0`", "`alpha1`" (all `alpha1` are mandatory). The name of `alpha$alpha0` and `alpha$alpha1` are the observation model names

	from the monolix project to which they are linked (if the observations models are defined whithout intercept, alpha$alpha0 need to be set to the vector NULL).
`lambda`	penalization parameter λ.
`eps1`	integer (>0) used to define the convergence criteria for the regression parameters.
`eps2`	integer (>0) used to define the convergence criteria for the likelihood.
`selfInit`	logical, if the SAEM is already done in the monolix project should be use as the initial point of the algorithm (if FALSE, SAEM is automatically compute according to `pop.set1` settings ; if TRUE, a SAEM through monolix need to have been launched).
`pop.set1`	population parameters setting for initialisation (see details).
`pop.set2`	population parameters setting for iterations.
`pop.set3`	population parameters setting for final estimation.
`prune`	percentage for prunning ($\in [0;1]$) in the Adaptative Gauss-Hermite algorithm used to compute the log-likelihood and its derivates (see `gh.LL`).
`n`	number of points for gaussian quadrature (see `gh.LL`).
`parallel`	logical, if the computation should be done in parallel when possible (default TRUE).
`ncores`	number of cores for parallelization (default NULL and `detectCores` is used).
`print`	logical, if the results and algotihm steps should be displayed in the console (default to TRUE).
`verbose`	logical, if progress bar should be printed when possible.
`digits`	number of digits to print (default to 3).
`trueValue`	-for simulation purposes- named vector of true value for parameters.
`finalSAEM`	logical, if a final SAEM should be launch with respect to the final selected set.
`test`	if Wald test should be computed at the end of the iteration.
`max.iter`	maximum number of iterations (default 20).
`p.max`	maximum value to each for wald test p.value (default 0.05).

Details

See `REMixed-package` for details on the model. For population parameter estimation settings, see (<https://monolixsuite.slp-software.com/r-functions/2024R1/setpopulationparameterestimationsettings>).

Value

a list of outputs of final project and through the iteration :

`info` informations about the parameters (project path, regulatization and population parameter names, alpha names, value of lambda used, if final SAEM and test has been computed, parameters p.max and N) ;

`finalRes` containing loglikelihood `LL` and penalized loglikelihood `LL.pen` values, final population parameters `param` and final regularization parameters `alpha` values, number of iterations `iter` and `time` needed , if computed, the estimated standard errors `standardError` and if test computed, the final results before test `saemBeforeTest` ;

`iterOutputs` the list of all remix outputs, i.e. parameters, lieklihood, SAEM estimates and convergence criterion value over the iteration.

See Also

cv.remix.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)
lambda = 382.22

res = remix(project = project,
            dynFUN = dynFUN_demo,
            y = y,
            ObsModel.transfo = ObsModel.transfo,
            alpha = alpha,
            selfInit = TRUE,
            eps1=10**(-2),
            eps2=1,
            lambda=lambda)

summary(res)

trueValue = read.csv(paste0(dirname(project),"/demoSMLX/Simulation/populationParameters.txt"))

plotSAEM(res,paramToPlot = c("delta_S_pop","phi_S_pop","delta_AB_pop"),trueValue=trueValue)

## End(Not run)
```

---

retrieveBest *REMixed results*

Description

Extracts the build minimizing an information criterion over a grid of lambda.

Usage

```
retrieveBest(fit, criterion = BICc)
```

Arguments

`fit`	output of `cv.remix`;
`criterion`	which criterion function to take into account. Default is the function ’BICc", but one can use ’BIC’, ’AIC’, ’eBIC’ or any function depending on a ‘cvRemix‘ object.

Value

outputs from `remix` algorithm achieving the best IC among those computed by `cv.remix`.

See Also

`cv.remix`, `remix`, `BIC.remix`, `eBIC`, `AIC.remix`, `BICc`.

Examples

```
## Not run:
project <- getMLXdir()

ObsModel.transfo = list(S=list(AB=log10),
                        linkS="yAB",
                        R=rep(list(S=function(x){x}),5),
                        linkR = paste0("yG",1:5))

alpha=list(alpha0=NULL,
           alpha1=setNames(paste0("alpha_1",1:5),paste0("yG",1:5)))

y = c(S=5,AB=1000)

cv.outputs = cv.Remix(project = project,
            dynFUN = dynFUN_demo,
            y = y,
            ObsModel.transfo = ObsModel.transfo,
            alpha = alpha,
            selfInit = TRUE,
            eps1=10**(-2),
            ncores=8,
            eps2=1)

res <- retrieveBest(cv.outputs)

plotConvergence(res)

trueValue = read.csv(paste0(dirname(project),"/demoSMLX/Simulation/populationParameters.txt"))#'

plotSAEM(res,paramToPlot = c("delta_S_pop","phi_S_pop","delta_AB_pop"),trueValue=trueValue)

## End(Not run)
```

Index