Package 'saemix'

May 9, 2026

Title Stochastic Approximation Expectation Maximization (SAEM) Algorithm

Version 3.5

Description The 'saemix' package implements the Stochastic Approximation EM algorithm for parameter estimation in (non)linear mixed effects models. It (i) computes the maximum likelihood estimator of the population parameters, without any approximation of the model (linearisation, quadrature approximation,...), using the Stochastic Approximation Expectation Maximization (SAEM) algorithm, (ii) provides standard errors for the maximum likelihood estimator (iii) estimates the conditional modes, the conditional means and the conditional standard deviations of the individual parameters, using the Hastings-Metropolis algorithm (see Comets et al. (2017) <doi:10.18637/jss.v080.i03>). Many applications of SAEM in agronomy, animal breeding and PKPD analysis have been published by members of the Monolix group. The full PDF documentation for the package including references about the algorithm and examples can be downloaded on the github of the IAME research institute for 'saemix': <https://github.com/iame-researchCenter/saemix/blob/7638e1b09ccb01cdff173068e01c266e906f76eb/docsaem.pdf>.

License GPL (>= 2)

LazyLoad yes

LazyData yes

Imports graphics, stats, methods, gridExtra, ggplot2, grid, rlang, mclust, scales, MASS

Suggests testthat, survival

Depends npde (>= 3.2)

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Collate 'aaa_generics.R' 'SaemixData.R' 'SaemixData-methods.R' 'SaemixData-methods_covariates.R' 'SaemixModel.R' 'SaemixRes.R' 'SaemixObject.R' 'backward.R' 'compute_LL.R' 'forward.R' 'func_FIM.R' 'func_aux.R' 'func_bootstrap.R' 'func_compare.R' 'func_discreteVPC.R' 'func_distcond.R' 'func_estimParam.R' 'func_exploreData.R' 'func_npde.R' 'func_plots.R'

'func_simulations.R' 'func_stepwise.R' 'main.R' 'main_estep.R'
'main_initialiseMainAlgo.R' 'main_mstep.R' 'stepwise.R' 'zzz.R'

Author Emmanuelle Comets [aut, cre],
Audrey Lavenu [aut],
Marc Lavielle [aut],
Belhal Karimi [aut],
Maud Delattre [ctb],
Alexandra Lavalley-Morelle [ctb],
Marilou Chanel [ctb],
Johannes Ranke [ctb] (ORCID: <https://orcid.org/0000-0003-4371-6538>),
Sofia Kaisaridi [ctb],
Lucie Fayette [ctb]

Maintainer Emmanuelle Comets <emmanuelle.comets@inserm.fr>

Repository CRAN

Date/Publication 2026-02-25 14:20:08 UTC

Contents

`backward.procedure` *Backward procedure for joint selection of covariates and random effects*

Description

Joint selection of covariates and random effects in a nonlinear mixed effects model by a backward-type algorithm based on two different versions of BIC for covariate selection and random effects selection respectively. Selection is made among the covariates as such specified in the SaemixData object. Only uncorrelated random effects structures are considered.

Usage

```
backward.procedure(saemixObject, trace = TRUE)
```

Arguments

`saemixObject`	An object returned by the `saemix` function
`trace`	If TRUE, a table summarizing the steps of the algorithm is printed. Default "TRUE"

Value

An object of the SaemixObject class storing the covariate model and the covariance structure of random effects of the final model.

Author(s)

Maud Delattre

References

M Delattre, M Lavielle, MA Poursat (2014) A note on BIC in mixed effects models. Electronic Journal of Statistics 8(1) p. 456-475 M Delattre, MA Poursat (2017) BIC strategies for model choice in a population approach. (arXiv:1612.02405)

checkInitialFixedEffects *Check initial fixed effects for an SaemixModel object applied to an SaemixData object*

Description

Check initial fixed effects for an SaemixModel object applied to an SaemixData object

Usage

```
checkInitialFixedEffects(model, data, psi = c(), id = c(), ...)
```

Arguments

model	an SaemixModel object
data	an SaemixData object (the predictors will then be extracted from the object using the name.predictors slot of the object)
psi	a vector or a dataframe giving the parameters for which predictions are to be computed (defaults to empty). The number of columns in psi (or the number of elements of psi, if psi is given as a vector) should match the number of parameters in the model, otherwise an error message will be shown and the function will return empty. If psi is NA, the predictions are computed for the population parameters in the model (first line of the psi0 slot). Covariates are not taken into account in the prediction. If psi is a dataframe, each line will be used for a separate 'subject' in the predictors dataframe, as indicated by the id argument; if id is not given, only the first line of psi will be used.
id	the vector of subjects for which individual plots will be obtained. If empty, the first 12 subjects in the dataset will be used (subject id's are taken from the name.group slot in the data object). If id is given, individual plots will be shown for the matching subjects in the dataset (eg if id=c(1:6), the first 6 subjects in the dataframe will be used for the plots, retrieving their ID from the data object)
...	unused argument, for consistency with the generic

Details

The function uses the model slot of the SaemixModel object to obtain predictions, using the predictors object. The user is responsible for giving all the predictors needed by the model function. if psi is not given, the predictions will be computed for the population parameters (first line of the psi0 slot) of the object.

The predictions correspond to the structure of the model. For models defined as a structural model, individual plots for the subjects in id overlaying the predictions for the parameters psi and the individual data are shown, and the predictions correspond to f(t_ij, psi). For models defined in terms of their likelihood, the predictions returned correspond to the log-likelihood. No individual graphs are currently available for discrete data models.

Warning: this function is currently under development and the output may change in future versions of the package

Value

the predictions corresponding to the values for each observation in the predictors of either the model f or log-likelihood. For Gaussian data models, the function also plots the data overlayed with the model predictions for each subject in id (where id is the index in the N subjects).

Examples

```
data(theo.saemix)
saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,modeltype="structural",
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

checkInitialFixedEffects(saemix.model, saemix.data, id=c(1:6))
checkInitialFixedEffects(saemix.model, saemix.data, id=c(1:6), psi=c(0.5, 30, 2)) # better fit
```

`coef.saemix` *Extract coefficients from an saemix fit*

Description

Extract coefficients from an saemix fit

Usage

```
## S3 method for class 'SaemixObject'
coef(object, ...)
```

Arguments

`object`	an SaemixObject object
`...`	further arguments to be passed to or from other methods

Value

a list with 3 components:

fixed fixed effects

population a list of population parameters with two elements, a matrix containing the untransformed parameters psi and a matrix containing the transformed parameters phi

individual a list of individual parameters with two elements, a matrix containing the untransformed parameters psi and a matrix containing the transformed parameters phi

---

`compare.saemix` *Model comparison with information criteria (AIC, BIC).*

Description

A specific penalty is used for BIC (BIC.cov) when the compared models have in common the structural model and the covariance structure for the random effects (see Delattre et al., 2014).

Usage

```
compare.saemix(..., method = c("is", "lin", "gq"))
```

Arguments

`...`	Two or more objects returned by the `saemix` function
`method`	The method used for computing the likelihood : "is" (Importance Sampling), "lin" (Linearisation) or "gq" (Gaussian quadrature). The default is to use importance sampling "is". If the requested likelihood is not available in all model objects, the method stops with a warning.

Details

Note that the comparison between two or more models will only be valid if they are fitted to the same dataset.

Value

A matrix of information criteria is returned, with at least two columns containing respectively AIC and BIC values for each of the compared models. When the models have in common the structural model and the covariance structure for the random effects, the matrix includes an additional column with BIC.cov values that are more appropriate when the comparison only concerns the covariates.

Author(s)

Maud Delattre

References

M Delattre, M Lavielle, MA Poursat (2014) A note on BIC in mixed effects models. Electronic Journal of Statistics 8(1) p. 456-475

Examples

```
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

# Definition of models to be compared
model1cpt<-function(psi,id,xidep) {
   dose<-xidep[,1]
   tim<-xidep[,2]
   ka<-psi[id,1]
   V<-psi[id,2]
   CL<-psi[id,3]
   k<-CL/V
   ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
   return(ypred)
}
# Model with one covariate
saemix.model1<-saemixModel(model=model1cpt,modeltype="structural",
  description="One-compartment model, clearance dependent on weight",
 psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3,byrow=TRUE, dimnames=list(NULL, c("ka","V","CL"))),
  transform.par=c(1,1,1),covariate.model=matrix(c(0,0,1,0,0,0),ncol=3,byrow=TRUE))
# Model with two covariates
saemix.model2<-saemixModel(model=model1cpt,modeltype="structural",
  description="One-compartment model, clearance dependent on weight, volume dependent on sex",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3,byrow=TRUE, dimnames=list(NULL, c("ka","V","CL"))),
   transform.par=c(1,1,1),covariate.model=matrix(c(0,0,1,0,1,0),ncol=3,byrow=TRUE))
# Model with three covariates
saemix.model3<-saemixModel(model=model1cpt,modeltype="structural",
 description="One-cpt model, clearance, absorption dependent on weight, volume dependent on sex",
 psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3,byrow=TRUE, dimnames=list(NULL, c("ka","V","CL"))),
  transform.par=c(1,1,1),covariate.model=matrix(c(1,0,1,0,1,0),ncol=3,byrow=TRUE))


# Running the main algorithm to estimate the population parameters
saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE)
saemix.fit1<-saemix(saemix.model1,saemix.data,saemix.options)
saemix.fit2<-saemix(saemix.model2,saemix.data,saemix.options)
saemix.fit3<-saemix(saemix.model3,saemix.data,saemix.options)

compare.saemix(saemix.fit1, saemix.fit2, saemix.fit3)
```

```
compare.saemix(saemix.fit1, saemix.fit2, saemix.fit3, method = "lin")

# We need to explicitly run Gaussian Quadrature if we want to use it in
# the comparisons
saemix.fit1 <- llgq.saemix(saemix.fit1)
saemix.fit2 <- llgq.saemix(saemix.fit2)
saemix.fit3 <- llgq.saemix(saemix.fit3)
compare.saemix(saemix.fit1, saemix.fit2, saemix.fit3, method = "gq")
```

`conddist.saemix` *Estimate conditional mean and variance of individual parameters using the MCMC algorithm*

Description

When the parameters of the model have been estimated, we can estimate the individual parameters (psi_i).

Usage

```
conddist.saemix(saemixObject, nsamp = 1, max.iter = NULL, plot = FALSE, ...)
```

Arguments

`saemixObject`	an object returned by the `saemix` function
`nsamp`	Number of samples to be drawn in the conditional distribution for each subject. Defaults to 1
`max.iter`	Maximum number of iterations for the computation of the conditional estimates. Defaults to twice the total number of iterations (see above)
`plot`	a boolean indicating whether to display convergence plots (defaults to FALSE)
`...`	optional arguments passed to the plots. Plots will appear if the argument plot is set to TRUE

Details

Let hattheta be the estimated value of theta computed with the SAEM algorithm and let p(phi_i |y_i; hattheta) be the conditional distribution of phi_i for 1<=i<=N . We use the MCMC procedure used in the SAEM algorithm to estimate these conditional distributions. We empirically estimate the conditional mean E(phi_i |y_i; hattheta) and the conditional standard deviation sd(phi_i |y_i; hattheta).

See PDF documentation for details of the computation. Briefly, the MCMC algorithm is used to obtain samples from the individual conditional distributions of the parameters. The algorithm is initialised for each subject to the conditional estimate of the individual parameters obtained at the end of the SAEMIX fit. A convergence criterion is used to ensure convergence of the mean and variance of the conditional distributions. When nsamp>1, several chains of the MCMC algorithm

are run in parallel to obtain samples from the conditional distributions, and the convergence criterion must be achieved for all chains. When nsamp>1, the estimate of the conditional mean is obtained by averaging over the different samples, and the samples from the conditional distribution are output as an array of dimension N x nb of parameters x nsamp in arrays phi.samp for the sampled phi_i and psi.samp for the corresponding psi_i. The variance of the conditional phi_i for each sample is given in a corresponding array phi.samp.var (the variance of the conditional psi_i is not given but may be computed via the delta-method or by transforming the confidence interval for phi_i).

The shrinkage for any given parameter for the conditional estimate is obtained as

Sh=1-var(eta_i)/omega(eta)

where var(eta_i) is the empirical variance of the estimates of the individual random effects, and omega(eta) is the estimated variance.

When the plot argument is set to TRUE, convergence graphs are produced They can be used assess whether the mean of the individual estimates (on the scale of the parameters) and the mean of the SD of the random effects have stabilised over the ipar.lmcmc (defaults to 50) iterations. The evolution of these variables for each parameter is shown as a continuous line while the shaded area represents the acceptable variation.

The function adds or modifies the following elements in the results:

cond.mean.phi Conditional mean of the individual distribution of the parameters (obtained as the mean of the samples)

cond.var.phi Conditional variance of the individual distribution of the parameters (obtained as the mean of the estimated variance of the samples)

cond.shrinkage Estimate of the shrinkage for the conditional estimates

cond.mean.eta Conditional mean of the individual distribution of the parameters (obtained as the mean of the samples)

phi.samp An array with 3 dimensions, giving nsamp samples from the conditional distributions of the individual parameters

phi.samp.var The estimated individual variances for the sampled parameters phi.samp

A warning is output if the maximum number of iterations is reached without convergence (the maximum number of iterations is the sum of the elements in saemix.options$nbiter.saemix).

Author(s)

Emmanuelle Comets <emmanuelle.comets@inserm.fr>

Audrey Lavenu

Marc Lavielle

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixData,SaemixModel, SaemixObject,saemixControl,saemix

Examples

```
# Not run (strict time constraints for CRAN)
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,dimnames=list(NULL,
  c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE), error.model="constant")

saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE)

saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)
saemix.fit<-conddist.saemix(saemix.fit,nsamp=3, plot=TRUE)

# First sample from the conditional distribution
# (a N (nb of subject) by nb.etas (nb of parameters) matrix)
print(head(saemix.fit["results"]["phi.samp"][,,1]))

# Second sample
print(head(saemix.fit["results"]["phi.samp"][,,2]))
```

---

cow.saemix *Evolution of the weight of 560 cows, in SAEM format*

Description

The cow.saemix data contains records of the weight of 560 cows on 9 or 10 occasions.

Usage

```
cow.saemix
```

Format

This data frame contains the following columns:

cow the unique identifier for each cow

time time (days)

weight a numeric vector giving the weight of the cow (kg)

birthyear year of birth (between 1988 and 1998)

twin existence of a twin (no=1, yes=2)

birthrank the rank of birth (beetween 3 and 7)

Details

An exponential model was assumed to describe the weight gain with time: y_ij = A_i (1- B_i exp(- K_i t_ij)) +epsilon_ij

References

JC Pinheiro, DM Bates (2000) *Mixed-effects Models in S and S-PLUS*, Springer, New York (Appendix A.19)

Examples

```
data(cow.saemix)
saemix.data<-saemixData(name.data=cow.saemix,header=TRUE,name.group=c("cow"),
      name.predictors=c("time"),name.response=c("weight"),
      name.covariates=c("birthyear","twin","birthrank"),
      units=list(x="days",y="kg",covariates=c("yr","-","-")))

growthcow<-function(psi,id,xidep) {
  x<-xidep[,1]
  a<-psi[id,1]
  b<-psi[id,2]
  k<-psi[id,3]
  f<-a*(1-b*exp(-k*x))
  return(f)
}
saemix.model<-saemixModel(model=growthcow,
      description="Exponential growth model",
      psi0=matrix(c(700,0.9,0.02,0,0,0),ncol=3,byrow=TRUE,
        dimnames=list(NULL,c("A","B","k"))),transform.par=c(1,1,1),fixed.estim=c(1,1,1),
      covariate.model=matrix(c(0,0,0),ncol=3,byrow=TRUE),
```

```
      covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
      omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

saemix.options<-list(algorithms=c(1,1,1),nb.chains=1,nbiter.saemix=c(200,100),
             seed=4526,save=FALSE,save.graphs=FALSE,displayProgress=FALSE)

# Plotting the data
plot(saemix.data,xlab="Time (day)",ylab="Weight of the cow (kg)")

saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)
```

`createSaemixObject` *Create saemix objects with only data filled in*

Description

Create saemix objects either with empty results or with parameters set by the user. This is an internal function used as a preliminary step to obtain predictions for new data and is not intended to be used directly.

Usage

```
createSaemixObject.empty(model, data, control = list())
```

Arguments

`model`	an saemixModel object
`data`	an saemixData object
`control`	a list of options (if empty, will be set to the default values by `saemixControl`)

Details

with createSaemixObject.empty, the data component is set to the data object, the model component is set to the model object, and the result component is empty

with createSaemixObject.initial, the data and model are set as with createSaemixObject.empty, but the population parameter estimates are used to initialise the result component as in the initialisation step of the algorithm (initialiseMainAlgo)

Value

an object of class "`SaemixObject`".

Examples

```
# TODO
```

dataGen.case *Bootstrap datasets*

Description

These functions create bootstrapped datasets using the bootstrap methods described in saemix.bootstrap

Usage

```
dataGen.case(saemixObject)

dataGen.NP(saemixObject, nsamp = 0, eta.sampc = NULL, conditional = FALSE)

dataGen.Par(saemixObject)
```

Arguments

saemixObject	an object returned by the saemix function
nsamp	number of samples from the conditional distribution (for method="conditional")
eta.sampc	if available, samples from the conditional distribution (otherwise, they are obtained within the function)
conditional	if TRUE, sample from the conditional distribution, if FALSE, sample within the empirical Bayes estimates (EBE) as in the traditional non-parametric residual bootstrap

default.saemix.plots *Wrapper functions to produce certain sets of default plots*

Description

These functions produce default sets of plots, corresponding to diagnostic or individual fits.

Usage

```
default.saemix.plots(saemixObject, ...)
```

Arguments

saemixObject	an object returned by the saemix function
...	optional arguments passed to the plots

Details

These functions are wrapper functions designed to produce default sets of plots to help the user assess their model fits.

Value

Depending on the type argument, the following plots are produced:

- default.saemix.plots by default, the following plots are produced: a plot of the data, convergence plots, plot of the likelihood by importance sampling (if computed), plots of observations versus predictions, scatterplots and distribution of residuals, boxplot of the random effects, correlations between random effects, distribution of the parameters, VPC
- basic.gof basic goodness-of-fit plots: convergence plots, plot of the likelihood by importance sampling (if computed), plots of observations versus predictions
- advanced.gof advanced goodness-of-fit plots: scatterplots and distribution of residuals, VPC,...
- covariate.fits plots of all estimated parameters versus all covariates in the dataset
- individual.fits plots of individual predictions (line) overlayed on individual observations (dots) for all subjects in the dataset

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

saemix, saemix.plot.data, saemix.plot.setoptions, plot.saemix

Examples

```
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
```

```
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

# Reducing the number of iterations due to time constraints for CRAN
saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE,nbiter.saemix=c(100,100))

saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)

default.saemix.plots(saemix.fit)

# Not run (time constraints for CRAN)
# basic.gof(saemix.fit)

# Not run (time constraints for CRAN)
# advanced.gof(saemix.fit)

individual.fits(saemix.fit)
```

`discreteVPC` *VPC for non Gaussian data models*

Description

This function provides VPC plots for non Gaussian data models (work in progress)

Usage

```
discreteVPC(object, outcome = "categorical", verbose = FALSE, ...)
```

Arguments

`object`	an saemixObject object returned by the `saemix` function. The object must include simulated data under the empirical design, using the model and estimated parameters from a fit, produced via the `simulateDiscreteSaemix` function.
`outcome`	type of outcome (valid types are "TTE", "binary", "categorical", "count")
`verbose`	whether to print messages (defaults to FALSE)
`...`	additional arguments, used to pass graphical options (to be implemented, currently not available)

Details

This function is a very rough first attempt at automatically creating VPC plots for models defined through their log-likelihood (categorical, count, or TTE data). It makes use of the new element simulate.function in the model component of the object

- for TTE data, a KM-VPC plot will be produced
- for count, categorical and binary data, a plot showing the proportion of each score/category across time will be shown along with the corresponding prediction intervals from the model These plots can be stratified over a covariate in the data set (currently only categorical covariates) by passing an argument which.cov='name' to the call

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr

References

Brendel, K, Comets, E, Laffont, C, Laveille, C, Mentre, F. Metrics for external model evaluation with an application to the population pharmacokinetics of gliclazide, Pharmaceutical Research 23 (2006), 2036-2049.

Holford, N. The Visual Predictive Check: superiority to standard diagnostic (Rorschach) plots (Abstract 738), in: 14th Meeting of the Population Approach Group in Europe, Pamplona, Spain, 2005.

Ron Keizer, tutorials on VPC TODO

See Also

SaemixObject, saemix, saemix.plot.vpc, simulateDiscreteSaemix

Examples

```
data(lung.saemix)

saemix.data<-saemixData(name.data=lung.saemix,header=TRUE,name.group=c("id"),
name.predictors=c("time","status","cens"),name.response=c("status"),
name.covariates=c("age", "sex", "ph.ecog", "ph.karno", "pat.karno", "wt.loss","meal.cal"),
units=list(x="days",y="",covariates=c("yr","","-","%","%","cal","pounds")))

weibulltte.model<-function(psi,id,xidep) {
  T<-xidep[,1]
  y<-xidep[,2] # events (1=event, 0=no event)
  cens<-which(xidep[,3]==1) # censoring times (subject specific)
  init <- which(T==0)
  lambda <- psi[id,1] # Parameters of the Weibull model
  beta <- psi[id,2]
  Nj <- length(T)
  ind <- setdiff(1:Nj, append(init,cens)) # indices of events
  hazard <- (beta/lambda)*(T/lambda)^(beta-1) # H'
  H <- (T/lambda)^beta # H
  logpdf <- rep(0,Nj) # ln(l(T=0))=0
  logpdf[cens] <- -H[cens] + H[cens-1] # ln(l(T=censoring time))
```

```
  logpdf[ind] <- -H[ind] + H[ind-1] + log(hazard[ind]) # ln(l(T=event time))
  return(logpdf)
}

simulateWeibullTTE <- function(psi,id,xidep) {
  T<-xidep[,1]
  y<-xidep[,2] # events (1=event, 0=no event)
  cens<-which(xidep[,3]==1) # censoring times (subject specific)
  init <- which(T==0)
  lambda <- psi[,1] # Parameters of the Weibull model
  beta <- psi[,2]
  tevent<-T
  Vj<-runif(dim(psi)[1])
  tsim<-lambda*(-log(Vj))^(1/beta) # nsuj events
  tevent[T>0]<-tsim
  tevent[tevent[cens]>T[cens]] <- T[tevent[cens]>T[cens]]
  return(tevent)
  }
saemix.model<-saemixModel(model=weibulltte.model,description="time model",modeltype="likelihood",
       simulate.function = simulateWeibullTTE,
           psi0=matrix(c(1,2),ncol=2,byrow=TRUE,dimnames=list(NULL, c("lambda","beta"))),
             transform.par=c(1,1),covariance.model=matrix(c(1,0,0,0),ncol=2, byrow=TRUE))
saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE, displayProgress=FALSE)


tte.fit<-saemix(saemix.model,saemix.data,saemix.options)
simtte.fit <- simulateDiscreteSaemix(tte.fit, nsim=100)
gpl <- discreteVPC(simtte.fit, outcome="TTE")
plot(gpl)
```

`discreteVPCTTE` *VPC for time-to-event models*

Description

This function provides VPC plots for time-to-event data models (work in progress)

Usage

```
discreteVPCTTE(
  object,
  ngrid = 200,
  interpolation.method = "step",
  verbose = FALSE,
  ...
)
```

Arguments

`object`	an saemixObject object returned by the `saemix` function. The object must include simulated data under the empirical design, using the model and estimated parameters from a fit, produced via the `simulateDiscreteSaemix` function
`ngrid`	number of grid points on the X-axis to extrapolate the KM-VPC
`interpolation.method`	method to use for the interpolation of the KM for the simulated datasets. Available methods are "step": the value of the survival function for a given grid point is set to the value of the last time "lin": a linear approximation is used between two consecutive times (defaults to "step")
`verbose`	whether to print messages (defaults to FALSE)
`...`	additional arguments, used to pass graphical options (to be implemented, currently not available)

Details

add details on TTE VPC, RTTE VPC, etc...

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr

See Also

`SaemixObject`, `saemix`, `saemix.plot.vpc`, `simulateDiscreteSaemix`

Tutorials on TTE-VPC TODO

`epilepsy.saemix` *Epilepsy count data*

Description

The epilepsy data from Thall and Vail (1990), available from the MASS package, records two-week seizure counts for 59 epileptics. The number of seizures was recorded for a baseline period of 8 weeks, and then patients were randomly assigned to a treatment group or a control group. Counts were then recorded for four successive two-week periods. The subject's age is the only covariate. See the documentation for epil in the MASS package for details on the dataset.

Source

MASS package in R

References

P Thall, S Vail (1990). Some covariance models for longitudinal count data with overdispersion. Biometrics 46(3):657-71.

Examples

```
# You need to have MASS installed to successfully run this example
if (requireNamespace("MASS")) {

  epilepsy<-MASS::epil
  saemix.data<-saemixData(name.data=epilepsy, name.group=c("subject"),
     name.predictors=c("period","y"),name.response=c("y"),
   name.covariates=c("trt","base", "age"), units=list(x="2-week",y="",covariates=c("","","yr")))
  ## Poisson model with one parameter
  countPoi<-function(psi,id,xidep) {
    y<-xidep[,2]
    lambda<-psi[id,1]
    logp <- -lambda + y*log(lambda) - log(factorial(y))
    return(logp)
    }
 saemix.model<-saemixModel(model=countPoi,description="Count model Poisson",modeltype="likelihood",
  psi0=matrix(c(0.5),ncol=1,byrow=TRUE,dimnames=list(NULL, c("lambda"))), transform.par=c(1))

  saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE, displayProgress=FALSE)
  poisson.fit<-saemix(saemix.model,saemix.data,saemix.options)

}
```

`fim.saemix` *Computes the Fisher Information Matrix by linearisation*

Description

Estimate by linearisation the Fisher Information Matrix and the standard error of the estimated parameters.

Usage

```
fim.saemix(saemixObject)
```

Arguments

`saemixObject`	an object returned by the `saemix` function

Details

The inverse of the Fisher Information Matrix provides an estimate of the variance of the estimated parameters theta. This matrix cannot be computed in closed-form for nonlinear mixed-effect models; instead, an approximation is obtained as the Fisher Information Matrix of the Gaussian model deduced from the nonlinear mixed effects model after linearisation of the function f around the conditional expectation of the individual Gaussian parameters. This matrix is a block matrix (no correlations between the estimated fixed effects and the estimated variances).

Value

The function returns an updated version of the object saemix.fit in which the following elements have been added:

se.fixed: standard error of fixed effects, obtained as part of the diagonal of the inverse of the Fisher Information Matrix (only when fim.saemix has been run, or when saemix.options$algorithms[2] is 1)

se.omega: standard error of the variance of random effects, obtained as part of the diagonal of the inverse of the Fisher Information Matrix (only when fim.saemix has been run, or when the saemix.options$algorithms[2] is 1)

se.res: standard error of the parameters of the residual error model, obtained as part of the diagonal of the inverse of the Fisher Information Matrix (only when fim.saemix has been run, or when the saemix.options$algorithms[2] is 1)

fim: Fisher Information Matrix

ll.lin: likelihood calculated by linearisation

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixObject,saemix

Examples

```
# Running the main algorithm to estimate the population parameters
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
```

```
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE), error.model="constant")

saemix.options<-list(algorithm=c(1,0,0),seed=632545,save=FALSE,save.graphs=FALSE)

# Not run (strict time constraints for CRAN)
# saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)

# Estimating the Fisher Information Matrix using the result of saemix
# & returning the result in the same object
# fim.saemix(saemix.fit)
```

`fitted.saemix` *Extract Model Predictions*

Description

fitted is a generic function which extracts model predictions from objects returned by modelling functions

Usage

```
## S3 method for class 'SaemixRes'
fitted(object, type = c("ipred", "ypred", "ppred", "icpred"), ...)

## S3 method for class 'SaemixObject'
fitted(object, type = c("ipred", "ypred", "ppred", "icpred"), ...)
```

Arguments

`object`	an object of type SaemixRes or SaemixObject
`type`	string determining which predictions are extracted. Possible values are: "ipred" (individual predictions obtained using the mode of the individual distribution for each subject, default), "ppred" (population predictions obtained using the population parameters f(E(theta))), "ypred" (mean of the population predictions

	(E(f(theta)))) and "icpred" (individual predictions obtained using the conditional mean of the individual distribution). See user guide for details.
...	further arguments to be passed to or from other methods

Value

Model predictions

`forward.procedure` *Backward procedure for joint selection of covariates and random effects*

Description

Joint selection of covariates and random effects in a nonlinear mixed effects model by a forward-type algorithm based on two different versions of BIC for covariate selection and random effects selection respectively. Selection is made among the covariates as such specified in the SaemixData object. Only uncorrelated random effects structures are considered.

Usage

```
forward.procedure(saemixObject, trace = TRUE)
```

Arguments

`saemixObject`	An object returned by the `saemix` function
`trace`	If TRUE, a table summarizing the steps of the algorithm is printed. Default "TRUE"

Value

An object of the SaemixObject class storing the covariate model and the covariance structure of random effects of the final model.

Author(s)

Maud Delattre

References

M Delattre, M Lavielle, MA Poursat (2014) A note on BIC in mixed effects models. Electronic Journal of Statistics 8(1) p. 456-475 M Delattre, MA Poursat (2017) BIC strategies for model choice in a population approach. (arXiv:1612.02405)

initialize-methods *Methods for Function initialize*

Description

Constructor functions for Classes in the saemix package

Usage

```
## S4 method for signature 'SaemixData'
initialize(
  .Object,
  name.data,
  header,
  sep,
  na,
  name.group,
  name.predictors,
  name.response,
  name.covariates,
  name.X,
  units,
  name.mdv,
  name.cens,
  name.occ,
  name.ytype,
  verbose = TRUE,
  automatic = TRUE
)

## S4 method for signature 'SaemixRepData'
initialize(.Object, data = NULL, nb.chains = 1)

## S4 method for signature 'SaemixSimData'
initialize(.Object, data = NULL, datasim = NULL)

## S4 method for signature 'SaemixModel'
initialize(
  .Object,
  model,
  description,
  modeltype,
  psi0,
  name.response,
  name.sigma,
  transform.par,
  fixed.estim,
```

```
  error.model,
  covariate.model,
  covariance.model,
  omega.init,
  error.init,
  name.modpar,
  verbose = TRUE
)

## S4 method for signature 'SaemixRes'
initialize(
  .Object,
  status = "empty",
  modeltype,
  name.fixed,
  name.random,
  name.sigma,
  fixed.effects,
  fixed.psi,
  betaC,
  betas,
  omega,
  respar,
  cond.mean.phi,
  cond.var.phi,
  mean.phi,
  phi,
  phi.samp,
  parpop,
  allpar,
  MCOV
)

## S4 method for signature 'SaemixObject'
initialize(.Object, data, model, options = list())
```

Arguments

.Object	an SaemixObject, SaemixRes, SaemixData or SaemixModel object to initialise
name.data	name of the dataset (can be a character string giving the name of a file on disk or of a dataset in the R session, or the name of a dataset
header	whether the dataset/file contains a header. Defaults to TRUE
sep	the field separator character. Defaults to any number of blank spaces ("")
na	a character vector of the strings which are to be interpreted as NA values. Defaults to c(NA)
name.group	name (or number) of the column containing the subject id

`name.predictors`
: name (or number) of the column(s) containing the predictors (the algorithm requires at least one predictor x)

`name.response`
: name (or number) of the column containing the response variable y modelled by predictor(s) x

`name.covariates`
: name (or number) of the column(s) containing the covariates, if present (otherwise missing)

`name.X`
: name of the column containing the regression variable to be used on the X axis in the plots (defaults to the first predictor)

`units`
: list with up to three elements, x, y and optionally covariates, containing the units for the X and Y variables respectively, as well as the units for the different covariates (defaults to empty)

`name.mdv`
: name of the column containing the indicator for missing variable

`name.cens`
: name of the column containing the indicator for censoring

`name.occ`
: name of the column containing the occasion

`name.ytype`
: name of the column containing the index of the response

`verbose`
: a boolean indicating whether messages should be printed out during the creation of the object

`automatic`
: a boolean indicating whether to attempt automatic name recognition when some colum names are missing or wrong (defaults to TRUE)

`data`
: an SaemixData object

`nb.chains`
: number of chains used in the algorithm

`datasim`
: dataframe containing the simulated data

`model`
: name of the function used to compute the structural model. The function should return a vector of predicted values given a matrix of individual parameters, a vector of indices specifying which records belong to a given individual, and a matrix of dependent variables (see example below).

`description`
: a character string, giving a brief description of the model or the analysis

`modeltype`
: a character string giving the model used for analysis

`psi0`
: a matrix with a number of columns equal to the number of parameters in the model, and one (when no covariates are available) or two (when covariates enter the model) giving the initial estimates for the fixed effects. The column names of the matrix should be the names of the parameters in the model, and will be used in the plots and the summaries. When only the estimates of the mean parameters are given, psi0 may be a named vector.

`name.sigma`
: a vector of character string giving the names of the residual error parameters (defaults to "a" and "b")

`transform.par`
: the distribution for each parameter (0=normal, 1=log-normal, 2=probit, 3=logit). Defaults to a vector of 1s (all parameters have a log-normal distribution)

`fixed.estim`
: whether parameters should be estimated (1) or fixed to their initial estimate (0). Defaults to a vector of 1s

error.model	type of residual error model (valid types are constant, proportional, combined and exponential). Defaults to constant
covariate.model	a matrix giving the covariate model. Defaults to no covariate in the model
covariance.model	a square matrix of size equal to the number of parameters in the model, giving the variance-covariance matrix of the model: 1s correspond to estimated variances (in the diagonal) or covariances (off-diagonal elements). Defaults to the identity matrix
omega.init	a square matrix of size equal to the number of parameters in the model, giving the initial estimate for the variance-covariance matrix of the model. The current default is a diagonal matrix with ones for all transformed parameters as well as for all untransformed parameters with an absolute value smaller than one. For untransformed parameters greater or equal to one, their squared value is used as the corresponding diagonal element.
error.init	a vector of size 2 giving the initial value of a and b in the error model. Defaults to 1 for each estimated parameter in the error model
name.modpar	names of the model parameters, if they are not given as the column names (or names) of psi0
status	string indicating whether a model has been run successfully; set to "empty" at initialisation, used to pass on error messages or fit status
name.fixed	a character string giving the name of the fixed parameters
name.random	a character string giving the name of the random parameters
fixed.effects	vector with the estimates of h(mu) and betas in estimation order
fixed.psi	vector with the estimates of h(mu)
betaC	vector with the estimates of betas (estimated fixed effects for covariates)
betas	vector with the estimates of mu
omega	estimated variance-covariance matrix
respar	vector with the estimates of the parameters of the residual error
cond.mean.phi	matrix of size (number of subjects) x (nb of parameters) containing the conditional mean estimates of the, defined as the mean of the conditional distribution
cond.var.phi	matrix of the variances on cond.mean.phi, defined as the variance of the conditional distribution
mean.phi	matrix of size (number of subjects) x (nb of parameters) giving for each subject the estimates of the population parameters including covariate effects
phi	matrix of size (number of subjects) x (nb of parameters) giving for each subject
phi.samp	samples from the individual conditional distributions of the phi
parpop	population parameters at each iteration
allpar	all parameters (including covariate effects) at each iteration
MCOV	design matrix C
options	a list of options passed to the algorithm

Methods

list("signature(.Object = \"SaemixData\")") create a SaemixData object. Please use the saemixData function.

list("signature(.Object = \"SaemixModel\")") create a SaemixModel object Please use the saemixModel function.

list("signature(.Object = \"SaemixObject\")") create a SaemixObject object. This object is obtained after a successful call to saemix

list("signature(.Object = \"SaemixRepData\")") create a SaemixRepData object

list("signature(.Object = \"SaemixRes\")") create a SaemixRes object

list("signature(.Object = \"SaemixSimData\")") create a SaemixSimData object

---

knee.saemix *Knee pain data*

Description

The knee.saemix data represents pain scores recorded in a clinical study in 127 patients with sport related injuries treated with two different therapies. After 3,7 and 10 days of treatment the pain occuring during knee movement was observed.

Usage

```
knee.saemix
```

Format

This data frame contains the following columns:

id subject index in file

time time of measurement (in days)

y knee pain (0=none to 4=severe)

Age patient age (scaled and centered)

Sex patient gender (0=male, 1=female)

RD moderate knee pain (defined as pain score 2 or more)

treatment treatment indicator (0=placebo, 1=treatment)

Age2 patient age, squared (Age^2)

#'

Details

The data in the knee.saemix was reformatted from the knee dataset provided by the catdata package (see data(knee, package="catdata")). A time column was added representing the day of the measurement (with 0 being the baseline value) and each observation corresponds to a different line in the dataset. Treatment was recoded as 0/1 (placebo/treatment), gender as 0/1 (male/female) and Age2 represents the squared of centered Age.

Please refer to the PDF documentation (chapter 4, section 4.6) for more details on the analysis, including examples of diagnostic plots.

Source

catdata package in R

References

G Tutz (2012), Regression for Categorical Data, Cambridge University Press.

#' @examples data(knee.saemix)

#' @keywords datasets

---

llgq.saemix *Log-likelihood using Gaussian Quadrature*

Description

Estimate the log-likelihood using Gaussian Quadrature (multidimensional grid)

Usage

```
llgq.saemix(saemixObject)
```

Arguments

saemixObject	an object returned by the saemix function

Details

The likelihood of the observations is estimated using Gaussian Quadrature (see documentation).

Value

the log-likelihood estimated by Gaussian Quadrature

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixObject,saemix,llis.saemix

Examples

```
# Running the main algorithm to estimate the population parameters
data(theo.saemix)
saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}
saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3,byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE), error.model="constant")

saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE)

# Not run (strict time constraints for CRAN)
# saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)

# Estimating the likelihood by Gaussian Quadrature using the result of saemix
# & returning the result in the same object
# saemix.fit<-llgq.saemix(saemix.fit)
```

`llis.saemix` *Log-likelihood using Importance Sampling*

Description

Estimate the log-likelihood using Importance Sampling

Usage

```
llis.saemix(saemixObject)
```

Arguments

`saemixObject`	an object returned by the `saemix` function

Details

The likelihood of the observations is estimated without any approximation using a Monte-Carlo approach (see documentation).

Value

the log-likelihood estimated by Importance Sampling

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

`SaemixObject`,`saemix`,`llgq.saemix`

Examples

```
# Running the main algorithm to estimate the population parameters
data(theo.saemix)
saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}
saemix.model<-saemixModel(model=model1cpt,
 description="One-compartment model with first-order absorption", modeltype="structural",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

saemix.options<-list(algorithm=c(1,0,0),seed=632545,save=FALSE,save.graphs=FALSE)

# Not run (strict time constraints for CRAN)
# saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)

# Estimating the likelihood by importance sampling using the result of saemix
# & returning the result in the same object
# saemix.fit<-llis.saemix(saemix.fit)
```

---

logLik *Extract likelihood from an SaemixObject resulting from a call to saemix*

Description

The likelihood in saemix can be computed by one of three methods: linearisation (linearisation of the model), importance sampling (stochastic integration) and gaussian quadrature (numerical integration). The linearised likelihood is obtained as a byproduct of the computation of the Fisher Information Matrix (argument FIM=TRUE in the options given to the saemix function). If no method argument is given, this function will attempt to extract the likelihood computed by importance sampling (method="is"), unless the object contains the likelihood computed by linearisation, in which case the function will extract this component instead. If the requested likelihood is not present in the object, it will be computed and aded to the object before returning.

Usage

```
## S3 method for class 'SaemixObject'
logLik(object, method = c("is", "lin", "gq"), ...)

## S3 method for class 'SaemixObject'
AIC(object, method = c("is", "lin", "gq"), ..., k = 2)

## S3 method for class 'SaemixObject'
BIC(object, method = c("is", "lin", "gq"), ...)

## S3 method for class 'covariate'
BIC(object, method = c("is", "lin", "gq"), ...)
```

Arguments

object	name of an SaemixObject object
method	character string, one of c("is","lin","gq"), to select one of the available approximations to the log-likelihood (is: Importance Sampling; lin: linearisation and gq: Gaussian Quadrature). See documentation for details
...	additional arguments
k	numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC

Details

BIC.covariate implements the computation of the BIC from Delattre et al. 2014.

Value

Returns the selected statistical criterion (log-likelihood, AIC, BIC) extracted from the SaemixObject, computed with the 'method' argument if given (defaults to IS).

Author(s)

Emmanuelle Comets <emmanuelle.comets@inserm.fr>

Audrey Lavenu

Marc Lavielle

Maud Delattre

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

AIC,BIC, saemixControl, saemix

---

lung.saemix *NCCTG Lung Cancer Data, in SAEM format*

Description

The lung.saemix contains survival data in patients with advanced lung cancer from the North Central Cancer Treatment Group. Performance scores rate how well the patient can perform usual daily activities. This data is available in the survival library for R and has been reformatted here for use in saemix (see details).

Usage

```
lung.saemix
```

Format

This data frame contains the following columns:

id subject index in file

inst institution code

time observation time since the beginning of follow-up

status 0=alive, 1=dead

cens 0=observed, 1=censored

sex patient gender (0=male, 1=female)

age age in years

ph.ecog ECOG performance score as rated by the physician. 0=asymptomatic, 1= symptomatic but completely ambulatory, 2= in bed <50% of the day, 3= in bed > 50% of the day but not bedbound, 4 = bedbound

ph.karno Karnofsky performance score (bad=0-good=100) rated by physician (%)

pat.karno Karnofsky performance score (bad=0-good=100) rated by patient (%)

meal.cal calories consumed at meals (cal)

wt.loss weight loss in last six months (pounds)

Details

The data in the lung.saemix was reformatted from the lung cancer dataset (see data(cancer, package="survival")). Patients with missing age, sex, institution or physician assessments were removed from the dataset. Status was recoded as 1 for death and 0 for a censored event, and a censoring column was added to denote whether the patient was dead or alive at the time of the last observation. For saemix, a line at time=0 was added for all subjects. Finally, subjects were numbered consecutively from 0 to 1.

Source

Terry Therneau from the survival package in R

References

CL Loprinzi, JA Laurie, HS Wieand, JE Krook, PJ Novotny, JW Kugler, et al. (1994). Prospective evaluation of prognostic variables from patient-completed questionnaires. North Central Cancer Treatment Group. Journal of Clinical Oncology. 12(3):601-7.

Examples

```
data(lung.saemix)

saemix.data<-saemixData(name.data=lung.saemix,header=TRUE,name.group=c("id"),
name.predictors=c("time","status","cens"),name.response=c("status"),
name.covariates=c("age", "sex", "ph.ecog", "ph.karno", "pat.karno", "wt.loss","meal.cal"),
units=list(x="days",y="",covariates=c("yr","","-","%","%","cal","pounds")))
weibulltte.model<-function(psi,id,xidep) {
  T<-xidep[,1]
  y<-xidep[,2] # events (1=event, 0=no event)
  cens<-which(xidep[,3]==1) # censoring times (subject specific)
  init <- which(T==0)
  lambda <- psi[id,1] # Parameters of the Weibull model
  beta <- psi[id,2]
  Nj <- length(T)
  ind <- setdiff(1:Nj, append(init,cens)) # indices of events
  hazard <- (beta/lambda)*(T/lambda)^(beta-1) # H'
  H <- (T/lambda)^beta # H
  logpdf <- rep(0,Nj) # ln(l(T=0))=0
  logpdf[cens] <- -H[cens] + H[cens-1] # ln(l(T=censoring time))
  logpdf[ind] <- -H[ind] + H[ind-1] + log(hazard[ind]) # ln(l(T=event time))
  return(logpdf)
}
saemix.model<-saemixModel(model=weibulltte.model,description="time model",modeltype="likelihood",
          psi0=matrix(c(1,2),ncol=2,byrow=TRUE,dimnames=list(NULL, c("lambda","beta"))),
            transform.par=c(1,1),covariance.model=matrix(c(1,0,0,0),ncol=2, byrow=TRUE))
saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE, displayProgress=FALSE)

tte.fit<-saemix(saemix.model,saemix.data,saemix.options)

# The fit from saemix using the above Weibull model may be compared
# to the non-parametric KM estimate
## Not run:
library(survival)
  lung.surv<-lung.saemix[lung.saemix$time>0,]
  lung.surv$status<-lung.surv$status+1
  Surv(lung.surv$time, lung.surv$status) # 1=censored, 2=dead
  f1 <- survfit(Surv(time, status) ~ 1, data = lung.surv)
  xtim<-seq(0,max(lung.saemix$time), length.out=200)
  estpar<-tte.fit@results@fixed.effects
  ypred<-exp(-(xtim/estpar[1])^(estpar[2]))
  plot(f1, xlab = "Days", ylab = "Overall survival probability")
```

```
  lines(xtim,ypred, col="red",lwd=2)

## End(Not run)
```

`map.saemix` *Estimates of the individual parameters (conditional mode)*

Description

Compute the estimates of the individual parameters PSI_i (conditional mode - Maximum A Posteriori)

Usage

```
map.saemix(saemixObject)
```

Arguments

`saemixObject`	an object returned by the `saemix` function

Details

The MCMC procedure is used to estimate the conditional mode (or Maximum A Posteriori) m(phi_i |yi ; hattheta) = Argmax_phi_i p(phi_i |yi ; hattheta)

Value

`saemixObject:`	returns the object with the estimates of the MAP parameters (see example for usage)

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41. E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

`SaemixObject`,`saemix`

Examples

```
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

saemix.options<-list(algorithm=c(1,0,0),seed=632545,
  save=FALSE,save.graphs=FALSE)

# Not run (strict time constraints for CRAN)
# saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)

# Estimating the individual parameters using the result of saemix
# & returning the result in the same object
# saemix.fit<-map.saemix(saemix.fit)
```

`mydiag` *Matrix diagonal*

Description

Extract or replace the diagonal of a matrix, or construct a diagonal matrix (replace diag function from R-base)

Usage

```
mydiag(x = 1, nrow, ncol)
```

Arguments

x	a matrix, vector or 1D array, or missing.
nrow	Optional number of rows for the result when x is not a matrix.
ncol	Optional number of columns for the result when x is not a matrix.

Value

If x is a matrix then diag(x) returns the diagonal of x. The resulting vector will have names if the matrix x has matching column and rownames.

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

See Also

diag

Examples

```
mydiag(1)
mydiag(c(1,2))
```

---

npdeSaemix *Create an npdeObject from an saemixObject*

Description

This function uses the npde library to compute normalised prediction distribution errors (npde) and normalised prediction discrepancies (npd). The simulations can also be used to plot VPCs. As of version 3.0, the plot functions for diagnostics involving VPC, npde or npd are deprecated and the user will be redirected to the current proposed method (creating an NpdeObject and using the plot functions from the library npde).

Usage

```
npdeSaemix(saemixObject, nsim = 1000)
```

Arguments

saemixObject	a fitted object resulting from a call to saemix()
nsim	the number of simulations used to compute npde (1000 by default, we suggest increasing it for large datasets)

Details

Since version 3.0, the saemix package depends on the npde package, which computes the npd/npde and produces graphs. See the documentation for npde for details on the computation methods See the PDF documentation and the bookdown https://iame-researchcenter.github.io/npde_bookdown/ for details on the different plots available.

Value

An object of class NpdeObject

Author(s)

Emmanuelle Comets emmanuelle.comets@bichat.inserm.fr

References

E Comets, K Brendel, F Mentre (2008). Computing normalised prediction distribution errors to evaluate nonlinear mixed-effect models: the npde add-on package for R. Computer Methods and Programs in Biomedicine, 90:154-66.

K Brendel, E Comets, C Laffont, C Laveille, F Mentre (2006). Metrics for external model evaluation with an application to the population pharmacokinetics of gliclazide. Pharmaceutical Research, 23:2036–49

PDF documentation for npde 3.0: https://github.com/ecomets/npde30/blob/main/userguide_npde_3.1.pdf

See Also

npde.graphs, gof.test

NpdeObject npde.plot.select autonpde

npde.plot.scatterplot npde.plot.dist

Examples

```
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
```

```
}

saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

saemix.options<-list(algorithm=c(1,0,0),seed=632545,save=FALSE,save.graphs=FALSE,
displayProgress=FALSE)
# Not run
# Works interactively but not in the contained environment of CRAN (it looks for a datafile
# instesad of finding the dataset in the environment)
# saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)
# npde.obj<-npdeSaemix(saemix.fit)
# plot(npde.obj)
# plot(npde.obj, plot.type="vpc")
# plot(npde.obj, plot.type="covariates")
# plot(npde.obj, plot.type="cov.x.scatter")
# plot(npde.obj, plot.type="cov.ecdf")
```

`oxboys.saemix` *Heights of Boys in Oxford*

Description

The `oxboys.saemix` data collects the height and age of students in Oxford measured over time. The data frame has 234 rows and 4 columns.

Usage

```
oxboys.saemix
```

Format

This data frame contains the following columns:

Subject an ordered factor giving a unique identifier for each boy in the experiment

age a numeric vector giving the standardized age (dimensionless)

height a numeric vector giving the height of the boy (cm)

Occasion an ordered factor - the result of converting 'age' from a continuous variable to a count so these slightly unbalanced data can be analyzed as balanced

Details

These data are described in Goldstein (1987) as data on the height of a selection of boys from Oxford, England versus a standardized age. The dataset can be found in the package `nlme`. We use an linear model for this data: y_ij = Base_i + slope_i x_ij +epsilon_ij

References

JC Pinheiro, DM Bates (2000) *Mixed-effects Models in S and S-PLUS*, Springer, New York (Appendix A.19)

Examples

```
data(oxboys.saemix)
saemix.data<-saemixData(name.data=oxboys.saemix,header=TRUE,
      name.group=c("Subject"),name.predictors=c("age"),name.response=c("height"),
      units=list(x="yr",y="cm"))

# plot the data
plot(saemix.data)

growth.linear<-function(psi,id,xidep) {
  x<-xidep[,1]
  base<-psi[id,1]
  slope<-psi[id,2]
  f<-base+slope*x
  return(f)
}
saemix.model<-saemixModel(model=growth.linear,description="Linear model",
      psi0=matrix(c(140,1),ncol=2,byrow=TRUE,dimnames=list(NULL,c("base","slope"))),
      transform.par=c(1,0),covariance.model=matrix(c(1,1,1,1),ncol=2,byrow=TRUE),
      error.model="constant")

saemix.options<-list(algorithms=c(1,1,1),nb.chains=1,seed=201004,
      save=FALSE,save.graphs=FALSE,displayProgress=FALSE)

saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)
```

PD1.saemix — *Data simulated according to an Emax response model, in SAEM format*

Description

The `PD1.saemix` and `PD2.saemix` data frames were simulated according to an Emax dose-response model.

Usage

```
PD1.saemix

PD2.saemix
```

Format

This data frame contains the following columns:

subject an variable identifying the subject on whom the observation was made. The ordering is by the dose at which the observation was made.

dose simulated dose.

response simulated response

gender gender (0 for male, 1 for female)

Details

These examples were used by P. Girard and F. Mentre for the symposium dedicated to Comparison of Algorithms Using Simulated Data Sets and Blind Analysis, that took place in Lyon, France, September 2004. The datasets contain 100 individuals, each receiving 3 different doses:(0, 10, 90), (5, 25, 65) or (0, 20, 30). It was assumed that doses were given in a cross-over study with sufficient wash out period to avoid carry over. Responses (y_ij) were simulated with the following pharmacodynamic model: y_ij = E0_i + D_ij Emax_i/(D_ij + ED50_i) +epsilon_ij The individual parameters were simulated according to log (E0_i) = log (E0) + eta_i1 log (Emax_i) = log (Emax) + eta_i2 log (E50_i) = log (E50) + beta w_i + eta_i3

PD1.saemix contains the data simulated with a gender effect, beta=0.3. PD2.saemix contains the data simulated without a gender effect, beta=0.

References

P Girard, F Mentre (2004). Comparison of Algorithms Using Simulated Data Sets and Blind Analysis workshop, Lyon, France.

Examples

```
data(PD1.saemix)
saemix.data<-saemixData(name.data=PD1.saemix,header=TRUE,name.group=c("subject"),
      name.predictors=c("dose"),name.response=c("response"),
      name.covariates=c("gender"), units=list(x="mg",y="-",covariates=c("-")))

modelemax<-function(psi,id,xidep) {
# input:
#   psi : matrix of parameters (3 columns, E0, Emax, EC50)
#   id : vector of indices
#   xidep : dependent variables (same nb of rows as length of id)
# returns:
#   a vector of predictions of length equal to length of id
  dose<-xidep[,1]
  e0<-psi[id,1]
```

```
  emax<-psi[id,2]
  e50<-psi[id,3]
  f<-e0+emax*dose/(e50+dose)
  return(f)
}

# Plotting the data
plot(saemix.data,main="Simulated data PD1")

# Compare models with and without covariates with LL by Importance Sampling
model1<-saemixModel(model=modelemax,description="Emax growth model",
       psi0=matrix(c(20,300,20,0,0,0),ncol=3,byrow=TRUE,dimnames=list(NULL,
       c("E0","Emax","EC50"))), transform.par=c(1,1,1),
       covariate.model=matrix(c(0,0,0), ncol=3,byrow=TRUE),fixed.estim=c(1,1,1))

model2<-saemixModel(model=modelemax,description="Emax growth model",
       psi0=matrix(c(20,300,20,0,0,0),ncol=3,byrow=TRUE,dimnames=list(NULL,
       c("E0","Emax","EC50"))), transform.par=c(1,1,1),
       covariate.model=matrix(c(0,0,1), ncol=3,byrow=TRUE),fixed.estim=c(1,1,1))

# SE not computed as not needed for the test
saemix.options<-list(algorithms=c(0,1,1),nb.chains=3,seed=765754,
       nbiter.saemix=c(500,300),save=FALSE,save.graphs=FALSE,displayProgress=FALSE)

fit1<-saemix(model1,saemix.data,saemix.options)
fit2<-saemix(model2,saemix.data,saemix.options)
wstat<-(-2)*(fit1["results"]["ll.is"]-fit2["results"]["ll.is"])

cat("LRT test for covariate effect on EC50: p-value=",1-pchisq(wstat,1),"\n")
```

---

plot,SaemixModel,ANY-method

Plot model predictions using an SaemixModel object

Description

This function will plot predictions obtained from an SaemixModel object over a given range of X. Additional predictors may be passed on to the function using the predictors argument.

Usage

```
## S4 method for signature 'SaemixModel,ANY'
plot(x, y, range = c(0, 1), psi = NULL, predictors = NULL, ...)
```

Arguments

x	an SaemixData object or an SaemixSimData object
y	unused, present for compatibility with base plot function

range	range of X over which the model is to be plotted. Important note: the *first* predictor will be used for the X-axis, the other predictors when present need to be passed sequentially in the predictors argument, in the order in which they appear in the model Less important note: please use explicitely range=XXX where XXX is of the form c(a,b) to pass the plotting range on the X-axis)
psi	parameters of the model
predictors	additional predictors needed to pass on to the model
...	additional arguments to be passed on to plot (titles, legends, ...). Use verbose=TRUE to print some messages concerning the characteristics of the plot

Examples

```
# Note that we have written the PK model so that time is the first predictor (xidep[,1])
# and dose the second
model1cpt<-function(psi,id,xidep) {
     tim<-xidep[,1]
     dose<-xidep[,2]
     ka<-psi[id,1]
     V<-psi[id,2]
     CL<-psi[id,3]
     k<-CL/V
     ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
     return(ypred)
     }
 x<-saemixModel(model=model1cpt,description="One-compartment model with first-order absorption",
          psi0=matrix(c(1.5,30,1), ncol=3,byrow=TRUE, dimnames=list(NULL, c("ka","V","CL"))))
 # Plot the model over 0-24h, using the parameters given in psi0 and a dose of 300
 plot(x, range=c(0,24), predictors=300, verbose=TRUE)
 # Plot the model over 0-24h, using another set of parameters and a dose of 350
 plot(x, range=c(0,24), psi=c(1.5,20,2), predictors=350, verbose=TRUE)
```

`plot,SaemixModel,SaemixData-method`

Plot model predictions for a new dataset. If the dataset is large, only the first 20 subjects (id's) will be shown.

Description

Plot model predictions for a new dataset. If the dataset is large, only the first 20 subjects (id's) will be shown.

Usage

```
## S4 method for signature 'SaemixModel,SaemixData'
plot(x, y, ...)
```

Arguments

x	an SaemixModel object
y	an SaemixData object
...	additional arguments. Passing psi=X where X is a vector or a dataframe will allow changing the parameters for which predictions are to be computed (defaults to the population parameters defined by the psi element of x) (see details)

Details

The function uses the model slot of the SaemixModel object to obtain predictions, using the dataset contained in the SaemixData object. The user is responsible for making sure data and model match. If psi is not given, the predictions will be computed for the population parameters (first line of the psi0 slot) of the object. If psi is given, the number of columns in psi (or the number of elements of psi, if psi is given as a vector) should match the number of parameters in the model, otherwise an error message will be shown and the function will return empty. If psi is a dataframe, each line will be used for a separate subject of the smx.data object. Elements of psi will be recycled if psi has less lines than the number of subjects in the dataset.

Currently this function only works for models defined as 'structural'.

Value

a ggplot object

Examples

```
data(theo.saemix)
saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
   name.group=c("Id"),name.predictors=c("Dose","Time"),
   name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
   units=list(x="hr",y="mg/L", covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,modeltype="structural",
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")
```

```
plot(saemix.model, saemix.data)
plot(saemix.model, saemix.data, psi=c(2, 40, 3))
indpsi<-data.frame(ka=2, V=seq(25,47,2), CL=seq(2.5,4.7, 0.2))
plot(saemix.model, saemix.data, psi=indpsi)
```

`plot,SaemixObject,ANY-method`

General plot function from SAEM

Description

Several plots (selectable by the type argument) are currently available: convergence plot, individual plots, predictions versus observations, distribution plots, VPC, residual plots, mirror.

Usage

```
## S4 method for signature 'SaemixObject,ANY'
plot(x, y, ...)
```

Arguments

x	an object returned by the saemix function
y	empty
...	optional arguments passed to the plots

Details

This is the generic plot function for an SaemixObject object, which implements different graphs related to the algorithm (convergence plots, likelihood estimation) as well as diagnostic graphs. A description is provided in the PDF documentation. Arguments such as main, xlab, etc... that can be given to the generic plot function may be used, and will be interpreted according to the type of plot that is to be drawn.

A special argument plot.type can be set to determine the type of plot; it can be one of:

data: A spaghetti plot of the data,displaying the observed data y as a function of the regression variable (time for a PK application)

convergence: For each parameter in the model, this plot shows the evolution of the parameter estimate versus the iteration number

likelihood: Graph showing the evolution of the log-likelihood during the estimation by importance sampling

observations.vs.predictions: Plot of the predictions computed with the population parameters versus the observations (left), and plot of the predictions computed with the individual parameters versus the observations (right)

residuals.scatter: Scatterplot of the residuals versus the predictor (top) and versus predictions (bottom), for weighted residuals (population residuals, left), individual weighted residuals (middle) and npde (right).

residuals.distribution: Distribution of the residuals, plotted as histogram (top) and as a QQ-plot (bottom), for weighted residuals (population residuals, left), individual weighted residuals (middle) and npde (right).

individual.fit: Individual fits are obtained using the individual parameters with the individual covariates

population.fit: Population fits are obtained using the population parameters with the individual covariates

both.fit: Individual fits, superposing fits obtained using the population parameters with the individual covariates (red) and using the individual parameters with the individual covariates (green)

mirror: Mirror plots assessing the compatibility of simulated data compared to the original

marginal.distribution: Distribution of the parameters (conditional on covariates when some are included in the model). A histogram of individual parameter estimates can be overlayed on the plot, but it should be noted that the histogram does not make sense when there are covariates influencing the parameters and a warning will be displayed

random.effects: Boxplot of the random effects

correlations: Correlation between the random effects

parameters.vs.covariates: Plots of the estimates of the individual parameters versus the covariates, using scatterplot for continuous covariates, boxplot for categorical covariates

randeff.vs.covariates: Plots of the estimates of the random effects versus the covariates, using scatterplot for continuous covariates, boxplot for categorical covariates

npde: Plots 4 graphs to evaluate the shape of the distribution of the normalised prediction distribution errors (npde)

vpc: Visual Predictive Check, with options to include the prediction intervals around the boundaries of the selected interval as well as around the median (50th percentile of the simulated data).

In addition, the following values for plot.type produce a series of plots:

reduced: produces the following plots: plot of the data, convergence plots, plot of the likelihood by importance sampling (if computed), plots of observations versus predictions. This is the default behaviour of the plot function applied to an SaemixObject object

full: produces the following plots: plot of the data, convergence plots, plot of the likelihood by importance sampling (if computed), plots of observations versus predictions, scatterplots and distribution of residuals, VPC, npde, boxplot of the random effects, distribution of the parameters, correlations between random effects, plots of the relationships between individually estimated parameters and covariates, plots of the relationships between individually estimated random effects and covariates

Each plot can be customised by modifying options, either through a list of options set by the `saemix.plot.setoptions` function, or on the fly by passing an option in the call to the plot (see examples).

Value

None

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixObject,saemix, saemix.plot.setoptions, saemix.plot.select, saemix.plot.data

Examples

```
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE)

# Not run (strict time constraints for CRAN)
# saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)

# Set of default plots
```

```
# plot(saemix.fit)

# Data
# plot(saemix.fit,plot.type="data")

# Convergence
# plot(saemix.fit,plot.type="convergence")

# Individual plot for subject 1, smoothed
# plot(saemix.fit,plot.type="individual.fit",ilist=1,smooth=TRUE)

# Individual plot for subject 1 to 12, with ask set to TRUE
# (the system will pause before a new graph is produced)
# plot(saemix.fit,plot.type="individual.fit",ilist=c(1:12),ask=TRUE)

# Diagnostic plot: observations versus population predictions
# par(mfrow=c(1,1))
# plot(saemix.fit,plot.type="observations.vs.predictions",level=0,new=FALSE)

# LL by Importance Sampling
# plot(saemix.fit,plot.type="likelihood")

# Scatter plot of residuals
# Data will be simulated to compute weighted residuals and npde
# the results shall be silently added to the object saemix.fit
# plot(saemix.fit,plot.type="residuals.scatter")

# Boxplot of random effects
# plot(saemix.fit,plot.type="random.effects")

# Relationships between parameters and covariates
# plot(saemix.fit,plot.type="parameters.vs.covariates")

# Relationships between parameters and covariates, on the same page
# par(mfrow=c(3,2))
# plot(saemix.fit,plot.type="parameters.vs.covariates",new=FALSE)

# VPC
# Not run (time constraints for CRAN)
# plot(saemix.fit,plot.type="vpc")
```

`plot-methods` *Methods for Function plot*

Description

Methods for function `plot`

Methods

list("signature(x = \"ANY\")") default plot function ?

list("signature(x = \"SaemixData\")") Plots the data. Defaults to a spaghetti plot of response versus predictor, with lines joining the data for one individual.

list("signature(x = \"SaemixModel\")") Plots prediction of the model

list("signature(x = \"SaemixObject\")") This method gives access to a number of plots that can be performed on a SaemixObject

list("signature(x = \"SaemixSimData\")") Plots simulated datasets

---

plot.SaemixData *Plot of longitudinal data*

Description

This function will plot a longitudinal dataframe contained in an SaemixData object. By default it produces a spaghetti plot, but arguments can be passed on to modify this behaviour.

Usage

```
## S3 method for class 'SaemixData'
plot(x, y, ...)

## S3 method for class 'SaemixSimData'
plot(x, y, irep = -1, prediction = FALSE, ...)
```

Arguments

x	an SaemixData object or an SaemixSimData object
y	unused, present for compatibility with base plot function
...	additional arguments to be passed on to plot (titles, legends, ...)
irep	which replicate datasets to use in the mirror plot (defaults to -1, causing a random simulated dataset to be sampled from the nsim simulated datasets)
prediction	if TRUE, plot the predictions without residual variability (ypred instead of ysim). Defaults to FALSE.

Details

this function can also be used to visualise the predictions for simulated values of the individual parameters, using the ypred element instead of the ysim element normally used here

plotDiscreteData *Plot non Gaussian data*

Description

This function provides exploration plots for non Gaussian longitudinal data (work in progress, doesn't work yet for RTTE)

Usage

```
plotDiscreteData(object, outcome = "continuous", verbose = FALSE, ...)

plotDiscreteDataElement(
  object,
  outcome = "categorical",
  mirror = FALSE,
  irep = 1,
  verbose = FALSE,
  ...
)
```

Arguments

object	an SaemixData object returned by the saemixData function. For plotDiscreteDataElement, an SaemixObject object returned by the saemix function
outcome	type of outcome (valid types are "TTE", "binary", "categorical", "count")
verbose	whether to print messages (defaults to FALSE)
...	additional arguments, used to pass graphical options (to be implemented, currently not available)
mirror	if TRUE, plots a mirror plot of the same type as the data (the object must include simulated data)
irep	number of the replication to use in the mirror plot

Details

This function is a very rough first attempt at automatically creating plots to explore discrete longitudinal data.

- for TTE data, a KM plot will be produced
- for count, categorical and binary data, a plot showing the proportion of each score/category across time will be shown These plots can be stratified over a covariate in the data set (currently only categorical covariates) by passing an argument which.cov='name' to the call #'

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr

References

Brendel, K, Comets, E, Laffont, C, Laveille, C, Mentre, F. Metrics for external model evaluation with an application to the population pharmacokinetics of gliclazide, Pharmaceutical Research 23 (2006), 2036-2049.

Holford, N. The Visual Predictive Check: superiority to standard diagnostic (Rorschach) plots (Abstract 738), in: 14th Meeting of the Population Approach Group in Europe, Pamplona, Spain, 2005.

Ron Keizer, tutorials on VPC TODO

See Also

SaemixObject, saemix, saemix.plot.vpc, simulateDiscreteSaemix

Examples

```
# Time-to-event data
data(lung.saemix)

saemix.data<-saemixData(name.data=lung.saemix,header=TRUE,name.group=c("id"),
name.predictors=c("time","status","cens"),name.response=c("status"),
name.covariates=c("age", "sex", "ph.ecog", "ph.karno", "pat.karno", "wt.loss","meal.cal"),
units=list(x="days",y="",covariates=c("yr","","-","%","%","cal","pounds")))

# Plots a KM survival plot
plotDiscreteData(saemix.data, outcome="TTE")
# Plots a KM survival plot, stratified by sex
plotDiscreteData(saemix.data, outcome="TTE", which.cov="sex")

# Count data
data(rapi.saemix)
saemix.data<-saemixData(name.data=rapi.saemix, name.group=c("id"),
                name.predictors=c("time","rapi"),name.response=c("rapi"),
               name.covariates=c("gender"),units=list(x="months",y="",covariates=c("")))

# Plots a histogram of the counts
plotDiscreteData(saemix.data, outcome="count")
```

`predict-methods` *Methods for Function predict*

Description

Methods for function `predict`

Usage

```
## S4 method for signature 'SaemixObject'
predict(
  object,
  newdata = NULL,
  type = c("ipred", "ypred", "ppred", "icpred"),
  se.fit = FALSE,
  ...
)
```

Arguments

object	an SaemixObject
newdata	an optional dataframe for which predictions are desired. If newdata is given, it must contain the predictors needed for the model in object
type	the type of predictions (ipred= individual, ppred=population predictions obtained with the population estimates, ypred=mean of the population predictions, icpred=conditional predictions). With newdata, individual parameters can be estimated if the new data contains observations; otherwise, predictions correspond to the population predictions ppred, and type is ignored.
se.fit	whether the SE are to be taken into account in the model predictions
...	additional arguments passed on to fitted()

Value

a vector or a dataframe (if more than one type) with the corresponding predictions for each observation in the dataframe

Methods

list("signature(object = \"ANY\")") Default predict functions

list("signature(object = \"SaemixObject\")") Computes predictions using the results of an SAEM fit

---

predict.SaemixModel *Predictions for a new dataset*

Description

Predictions for a new dataset

Usage

```
## S3 method for class 'SaemixModel'
predict(object, predictors, psi = c(), id = c(), ...)
```

Arguments

object	an SaemixModel object
predictors	a dataframe with the predictors for the model (must correspond to the predictors used by the model function), or an SaemixData object (the predictors will then be extracted from the object).
psi	a vector or a dataframe giving the parameters for which predictions are to be computed (defaults to empty). The number of columns in psi (or the number of elements of psi, if psi is given as a vector) should match the number of parameters in the model, otherwise an error message will be shown and the function will return empty. If psi is NA, the predictions are computed for the population parameters in the model (first line of the psi0 slot). Covariates are not taken into account in the prediction. If psi is a dataframe, each line will be used for a separate 'subject' in the predictors dataframe, as indicated by the id argument; if id is not given, only the first line of psi will be used.
id	a vector of indices of length equal to the number of lines in predictors, matching each line of predictors to the corresponding line in psi, ie the parameters for this predictors (defaults to empty). If id is given, the unique values in id must be equal to the number of lines in psi, otherwise id will be set to 1. If id is given and its values do not take the consecutive values 1:N, the indices will be matched to 1:N to follow the lines in psi.
...	unused argument, for consistency with the generic

Details

The function uses the model slot of the SaemixModel object to obtain predictions, using the predictors object. The user is responsible for giving all the predictors needed by the model function. if psi is not given, the predictions will be computed for the population parameters (first line of the psi0 slot) of the object.

The predictions correspond to the structure of the model; for models defined in terms of their likelihood, the predictions are the log-pdf of the model (see documentation for details).

Warning: this function is currently under development and the output may change in future versions of the package to conform to the usual predict functions.

Value

a list with two components

param a dataframe with the estimated parameters

predictions a dataframe with the population predictions

Examples

```
data(theo.saemix)
xpred<-theo.saemix[,c("Dose","Time")]

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
```

```
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,modeltype="structural",
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

head(predict(saemix.model, xpred)$predictions)
head(predict(saemix.model, xpred, psi=c(2, 40, 0.5))$predictions)
indpsi<-data.frame(ka=2, V=seq(25,47,2), CL=seq(2.5,4.7, 0.2))
head(predict(saemix.model, xpred, psi=indpsi)$predictions)
```

`print-methods` *Methods for Function print*

Description

Prints a summary of an object

Usage

```
## S4 method for signature 'SaemixData'
print(x, nlines = 10, ...)

## S4 method for signature 'SaemixModel'
print(x, ...)

## S4 method for signature 'SaemixRes'
print(x, digits = 2, map = FALSE, ...)

## S4 method for signature 'SaemixObject'
print(x, nlines = 10, ...)
```

Arguments

`x`	an object of type SaemixData, SaemixModel, SaemixRes or SaemixObject
`nlines`	maximum number of lines of data to print (defaults to 10)

...	additional arguments passed on the print function
digits	number of digits to use for pretty printing
map	when map is TRUE the individual parameter estimates are shown (defaults to FALSE)

Methods

list("signature(x = \"ANY\")") Default print function

list("signature(x = \"SaemixData\")") Prints a summary of a SaemixData object

list("signature(x = \"SaemixModel\")") Prints a summary of a SaemixModel object

list("signature(x = \"SaemixObject\")") Prints a summary of the results from a SAEMIX fit

list("signature(x = \"SaemixRes\")") Not user-level

---

psi-methods *Functions to extract the individual estimates of the parameters and random effects*

Description

These three functions are used to access the estimates of individual parameters and random effects.

Usage

```
psi(object, type = c("mode", "mean"))

phi(object, type = c("mode", "mean"))

eta(object, type = c("mode", "mean"))

## S4 method for signature 'SaemixObject'
psi(object, type = c("mode", "mean"))

## S4 method for signature 'SaemixObject'
phi(object, type = c("mode", "mean"))

## S4 method for signature 'SaemixObject'
eta(object, type = c("mode", "mean"))
```

Arguments

object	an SaemixObject object returned by the saemix function
type	a string specifying whether to use the MAP (type="mode") or the mean (type="mean") of the conditional distribution of the individual parameters. Defaults to mode

Details

The psi_i represent the individual parameter estimates. In the SAEM algorithm, these parameters are assumed to be a transformation of a Gaussian random vector phi_i, where the phi_i can be written as a function of the individual random effects (eta_i), the covariate matrix (C_i) and the vector of fixed effects (mu):

phi_i = C_i mu + eta_i

More details can be found in the PDF documentation.

Value

a matrix with the individual parameters (psi/phi) or the random effects (eta). These functions are used to access and output the estimates of parameters and random effects. When the object passed to the function does not contain these estimates, they are automatically computed. The object is then returned (invisibly) with these estimates added to the results.

Methods

list("signature(object = \"SaemixObject\")") please refer to the PDF documentation for the models

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixData,SaemixModel, SaemixObject, saemixControl, plot.saemix

Examples

```
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
```

```
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

saemix.options<-list(algorithm=c(1,0,0),seed=632545,save=FALSE,save.graphs=FALSE,
displayProgress=FALSE)

# Not run (strict time constraints for CRAN)

saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)
psi(saemix.fit)
phi(saemix.fit)
eta(saemix.fit,type="mean")
```

`rapi.saemix` *Rutgers Alcohol Problem Index*

Description

The RAPI data studies gender differences across two years in alcohol-related problems, as measured by the Rutgers Alcohol Problem Index (RAPI; White & Labouvie, 1989). The dataset includes 3,616 repeated measures of counts representing the number of alcohol problems reported over six months period, across five time points from 818 individuals.

Format

This data frame contains the following columns:

id subject identification number

time time since the beginning of the study (months)

rapi the number of reported alcohol problems during a six-months period

gender gender (0=Men, 1=Women

Source

David Atkins, University of Washington

References

D Atkins, S Baldwin, C Zheng, R Gallop, C Neighbors C (2013). A tutorial on count regression and zero-altered count models for longitudinal substance use data. Psychology of Addictive Behaviors, 27(1):166–177.

C Neighbors, Lewis MA, Atkins D, Jensen MM, Walter T, Fossos N, Lee C, Larimer M (2010). Efficacy of web-based personalized normative feedback: A two-year randomized controlled trial. Journal of Consulting and Clinical Psychology 78(6):898-911.

C Neighbors, N Barnett, D Rohsenow, S Colby, P Monti (2010). Cost-Effectiveness of a Motivational lntervention for Alcohol-Involved Youth in a Hospital Emergency Department. Journal of Studies on Alcohol and Drugs 71(3):384-394.

Examples

```
 data(rapi.saemix)
 saemix.data<-saemixData(name.data=rapi.saemix, name.group=c("id"),
                      name.predictors=c("time","rapi"),name.response=c("rapi"),
                  name.covariates=c("gender"),units=list(x="months",y="",covariates=c("")))
hist(rapi.saemix$rapi, main="", xlab="RAPI score", breaks=30)


# Fitting a Poisson model
count.poisson<-function(psi,id,xidep) {
  time<-xidep[,1]
  y<-xidep[,2]
  intercept<-psi[id,1]
  slope<-psi[id,2]
  lambda<- exp(intercept + slope*time)
  logp <- -lambda + y*log(lambda) - log(factorial(y))
  return(logp)
}
countsimulate.poisson<-function(psi, id, xidep) {
   time<-xidep[,1]
   y<-xidep[,2]
   ymax<-max(y)
   intercept<-psi[id,1]
   slope<-psi[id,2]
   lambda<- exp(intercept + slope*time)
   y<-rpois(length(time), lambda=lambda)
  y[y>ymax]<-ymax+1 # truncate to maximum observed value to avoid simulating aberrant values
   return(y)
}
# Gender effect on intercept and slope
rapimod.poisson<-saemixModel(model=count.poisson, simulate.function=countsimulate.poisson,
   description="Count model Poisson",modeltype="likelihood",
  psi0=matrix(c(log(5),0.01),ncol=2,byrow=TRUE,dimnames=list(NULL, c("intercept","slope"))),
   transform.par=c(0,0), omega.init=diag(c(0.5, 0.5)),
    covariance.model =matrix(data=1, ncol=2, nrow=2),
```

```
    covariate.model=matrix(c(1,1), ncol=2, byrow=TRUE))
saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE, displayProgress=FALSE, fim=FALSE)
poisson.fit<-saemix(rapimod.poisson,saemix.data,saemix.options)

# Fitting a ZIP model
count.poissonzip<-function(psi,id,xidep) {
  time<-xidep[,1]
  y<-xidep[,2]
  intercept<-psi[id,1]
  slope<-psi[id,2]
  p0<-psi[id,3] # Probability of zero's
  lambda<- exp(intercept + slope*time)
  logp <- log(1-p0) -lambda + y*log(lambda) - log(factorial(y)) # Poisson
  logp0 <- log(p0+(1-p0)*exp(-lambda)) # Zeroes
  logp[y==0]<-logp0[y==0]
  return(logp)
}
countsimulate.poissonzip<-function(psi, id, xidep) {
  time<-xidep[,1]
  y<-xidep[,2]
  ymax<-max(y)
  intercept<-psi[id,1]
  slope<-psi[id,2]
  p0<-psi[id,3] # Probability of zero's
  lambda<- exp(intercept + slope*time)
  prob0<-rbinom(length(time), size=1, prob=p0)
  y<-rpois(length(time), lambda=lambda)
  y[prob0==1]<-0
 y[y>ymax]<-ymax+1 # truncate to maximum observed value to avoid simulating aberrant values
  return(y)
}
rapimod.zip<-saemixModel(model=count.poissonzip, simulate.function=countsimulate.poissonzip,
   description="count model ZIP",modeltype="likelihood",
   psi0=matrix(c(1.5, 0.01, 0.2),ncol=3,byrow=TRUE,
   dimnames=list(NULL, c("intercept", "slope","p0"))),
  transform.par=c(0,0,3), covariance.model=diag(c(1,1,0)), omega.init=diag(c(0.5,0.3,0)),
   covariate.model = matrix(c(1,1,0),ncol=3, byrow=TRUE))
zippoisson.fit<-saemix(rapimod.zip,saemix.data,saemix.options)

# Using simulations to compare the predicted proportion of 0's in the two models
nsim<-100
yfit1<-simulateDiscreteSaemix(poisson.fit,  nsim=nsim)
yfit2<-simulateDiscreteSaemix(zippoisson.fit,  nsim=100)
{
nobssim<-length(yfit1@sim.data@datasim$ysim)
cat("Observed proportion of 0's",
   length(yfit1@data@data$rapi[yfit1@data@data$rapi==0])/yfit1@data@ntot.obs,"\n")
cat("      Poisson model, p=",
   length(yfit1@sim.data@datasim$ysim[yfit1@sim.data@datasim$ysim==0])/nobssim,"\n")
cat("          ZIP model, p=",
   length(yfit2@sim.data@datasim$ysim[yfit2@sim.data@datasim$ysim==0])/nobssim,"\n")
}
```

readSaemix,SaemixData-method

Create a longitudinal data structure from a file or a dataframe Helper function not intended to be called by the user

Description

Create a longitudinal data structure from a file or a dataframe

Helper function not intended to be called by the user

Usage

```
## S4 method for signature 'SaemixData'
readSaemix(object, dat = NULL)
```

Arguments

object	an SaemixData object
dat	the name of a dataframe in the R environment, defaults to NULL; if NULL, the function will attempt to read the file defined by the slot name.data.

replaceData

Replace the data element in an SaemixObject object

Description

Returns an SaemixObject object where the data object has been replaced by the data provided in a dataframe

Usage

```
replaceData.saemixObject(saemixObject, newdata)
```

Arguments

saemixObject	an SaemixObject object
newdata	a dataframe containing data

Value

an object of class "SaemixObject". The population parameters are retained but all the predictions, individual parameters and statistical criteria are removed. The function attempts to extract the elements entering the statistical model (subject id, predictors, covariates and response).

Examples

```
# TODO
```

`resid.saemix` *Extract Model Residuals*

Description

residuals is a generic function which extracts model residuals from objects returned by modelling functions. The abbreviated form resid is an alias for residuals

Usage

```
## S3 method for class 'SaemixRes'
resid(object, type = c("ires", "wres", "npde", "pd", "iwres", "icwres"), ...)

## S3 method for class 'SaemixObject'
resid(object, type = c("ires", "wres", "npde", "pd", "iwres", "icwres"), ...)
```

Arguments

`object`	an SaemixRes or an SaemixObject object
`type`	string determining which residuals are extracted. Possible values are: "ires" (individual residuals, default), "wres" (weighted population residuals), "npde" (normalised prediction distribution errors), "pd" (prediction discrepancies), "iwres" (individual weighted residuals) and "icwres" (conditional individual weighted residuals). See user guide for details.
`...`	further arguments to be passed to or from other methods

Value

By default, individual residuals are extracted from the model object

---

`saemix` *Stochastic Approximation Expectation Maximization (SAEM) algorithm*

Description

SAEM algorithm perform parameter estimation for nonlinear mixed effects models without any approximation of the model (linearization, quadrature approximation, . . .)

Usage

```
saemix(model, data, control = list())
```

Arguments

model	an object of class SaemixModel, created by a call to the function saemixModel
data	an object of class SaemixData, created by a call to the function saemixData
control	a list of options, see saemixControl

Details

The SAEM algorithm is a stochastic approximation version of the standard EM algorithm proposed by Kuhn and Lavielle (see reference). Details of the algorithm can be found in the pdf file accompanying the package.

Value

An object of class SaemixObject containing the results of the fit of the data by the non-linear mixed effect model. A summary of the results is printed out to the terminal, and, provided the appropriate options have not been changed, numerical and graphical outputs are saved in a directory.

Author(s)

Emmanuelle Comets <emmanuelle.comets@inserm.fr>

Audrey Lavenu

Marc Lavielle

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixData,SaemixModel, SaemixObject, saemixControl, plot.saemix

Examples

```
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
   name.group=c("Id"),name.predictors=c("Dose","Time"),
   name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
   units=list(x="hr",y="mg/L", covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
```

```
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,modeltype="structural",
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")


# Not run (strict time constraints for CRAN)
 saemix.fit<-saemix(saemix.model,saemix.data,list(seed=632545,directory="newtheo",
 save=FALSE,save.graphs=FALSE, print=FALSE))

# Prints a summary of the results
print(saemix.fit)

# Outputs the estimates of individual parameters
psi(saemix.fit)

# Shows some diagnostic plots to evaluate the fit
plot(saemix.fit)
```

`saemix.bootstrap` *Bootstrap for saemix fits*

Description

This function provides bootstrap estimates for a saemixObject run. Different bootstrap approaches have been implemented (see below for details), with the default method being the conditional non-parametric bootstrap ()

Usage

```
saemix.bootstrap(
  saemixObject,
  method = "conditional",
  nboot = 200,
  nsamp = 100,
  saemix.options = NULL
)
```

Arguments

saemixObject	an object returned by the saemix function
method	the name of the bootstrap algorithm to use (one of: "case", "residual", "parametric" or "conditional") (defaults to "conditional")
nboot	number of bootstrap samples
nsamp	number of samples from the conditional distribution (for method="conditional")
saemix.options	list of options to run the saemix algorithm. Defaults to the options in the object, but suppressing the estimation of individual parameters (map=FALSE) and likelihood (ll.is=FALSE), and the options to print out intermediate results and graphs (displayProgress=FALSE,save.graphs=FALSE,print=FALSE)

Details

Different bootstrap algorithms have been proposed for non-linear mixed effect models, to account for the hierarchical nature of these models (see review in Thai et al. 2013, 2014) In saemix, we implemented the following bootstrap algorithms, which can be selected using the "method" argument:

- case refers to case bootstrap, where resampling is performed at the level of the individual and the bootstrap sample consists in sampling with replacement from the observed individuals
- residual refers to non-parametric residual bootstrap, where for each individual the residuals for the random effects and for the residual error are resampled from the individual estimates after the fit
- parametric refers to parametric residual bootstrap, where these residuals are sampled from their theoretical distributions. In saemix, random effects are drawn from a normal distribution using the estimated variance in saemixObject, and transformed to the distribution specified by the transform.par component of the model, and the residual error is drawn from a normal distribution using the estimated residual variance.
- conditional refers to the conditional non-parametric bootstrap, where the random effects are resampled from the individual conditional distributions as in (Comets et al. 2021) and the residual errors resampled from the corresponding residuals

Important note: for discrete data models, all residual-based bootstraps (residual, parametric and conditional) need a simulate.function slot to be included in the model object, as the algorithm will need to generate predictions with the resampled individual parameters in order to generate bootstrap samples. See SaemixModel and the examples provided as notebooks for details.

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr

References

Thai H, Mentré F, Holford NH, Veyrat-Follet C, Comets E. A comparison of bootstrap approaches for estimating uncertainty of parameters in linear mixed-effects models. Pharmaceutical Statistics, 2013 ;12:129–40.

Thai H, Mentré F, Holford NH, Veyrat-Follet C, Comets E. Evaluation of bootstrap methods for estimating uncertainty of parameters in nonlinear mixed-effects models : a simulation study in population pharmacokinetics. Journal of Pharmacokinetics and Pharmacodynamics, 2014; 41:15–33.

Comets E, Rodrigues C, Jullien V, Moreno U. Conditional non-parametric bootstrap for non-linear mixed effect models. Pharmaceutical Research, 2021; 38, 1057–66.

Examples

```
# Bootstrap for the theophylline data
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

saemix.options<-list(algorithm=c(1,0,0),seed=632545,save=FALSE,save.graphs=FALSE,
displayProgress=FALSE)

# Not run (strict time constraints for CRAN)

saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)
# Only 10 bootstrap samples here for speed, please increase to at least 200
theo.case <- saemix.bootstrap(saemix.fit, method="case", nboot=10)
theo.cond <- saemix.bootstrap(saemix.fit, nboot=10)


# Bootstrap for the toenail data
data(toenail.saemix)
saemix.data<-saemixData(name.data=toenail.saemix,name.group=c("id"), name.predictors=c("time","y"),
 name.response="y", name.covariates=c("treatment"),name.X=c("time"))

binary.model<-function(psi,id,xidep) {
  tim<-xidep[,1]
  y<-xidep[,2]
  inter<-psi[id,1]
  slope<-psi[id,2]
```

```
  logit<-inter+slope*tim
  pevent<-exp(logit)/(1+exp(logit))
  pobs = (y==0)*(1-pevent)+(y==1)*pevent
  logpdf <- log(pobs)
  return(logpdf)
}
simulBinary<-function(psi,id,xidep) {
    tim<-xidep[,1]
    y<-xidep[,2]
    inter<-psi[id,1]
    slope<-psi[id,2]
    logit<-inter+slope*tim
    pevent<-1/(1+exp(-logit))
    ysim<-rbinom(length(tim),size=1, prob=pevent)
    return(ysim)
    }

saemix.model<-saemixModel(model=binary.model,description="Binary model",
     modeltype="likelihood", simulate.function=simulBinary,
    psi0=matrix(c(-5,-.1,0,0),ncol=2,byrow=TRUE,dimnames=list(NULL,c("inter","slope"))),
     transform.par=c(0,0))


saemix.options<-list(seed=1234567,save=FALSE,save.graphs=FALSE, displayProgress=FALSE,
   nb.chains=10, fim=FALSE)
binary.fit<-saemix(saemix.model,saemix.data,saemix.options)
# Only 10 bootstrap samples here for speed, please increase to at least 200
toenail.case <- saemix.bootstrap(binary.fit, method="case", nboot=10)
toenail.cond <- saemix.bootstrap(binary.fit, nboot=10)
```

`saemix.plot.data` *Functions implementing each type of plot in SAEM*

Description

Several plots (selectable by the type argument) are currently available: convergence plot, individual plots, predictions versus observations, distribution plots, VPC, residual plots, and mirror plots.

Usage

```
saemix.plot.data(saemixObject, ...)
```

Arguments

`saemixObject`	an object returned by the `saemix` function
`...`	optional arguments passed to the plots

Details

These functions implement plots different graphs related to the algorithm (convergence plots, likelihood estimation) as well as diagnostic graphs. A description is provided in the PDF documentation.

saemix.plot.parcov.aux, compute.sres and compute.eta.map are helper functions, not intended to be called by the user directly.

By default, the following plots are produced:

saemix.plot.data: A spaghetti plot of the data, displaying the observed data y as a function of the regression variable (time for a PK application)

saemix.plot.convergence: For each parameter in the model, this plot shows the evolution of the parameter estimate versus the iteration number

saemix.plot.llis: Graph showing the evolution of the log-likelihood during the estimation by importance sampling

saemix.plot.obsvspred: Plot of the predictions computed with the population parameters versus the observations (left), and plot of the predictions computed with the individual parameters versus the observations (right)

saemix.plot.scatterresiduals: Scatterplot of the residuals versus the predictor (top) and versus predictions (bottom), for weighted residuals (population residuals, left), individual weighted residuals (middle) and npde (right).

saemix.plot.distribresiduals: Distribution of the residuals, plotted as histogram (top) and as a QQ-plot (bottom), for weighted residuals (population residuals, left), individual weighted residuals (middle) and npde (right).

saemix.plot.fits: Model fits. Individual fits are obtained using the individual parameters with the individual covariates. Population fits are obtained using the population parameters with the individual covariates (red) and the individual parameters with the individual covariates (green). By default the individual plots are displayed.

saemix.plot.distpsi: Distribution of the parameters (conditional on covariates when some are included in the model). A histogram of individual parameter estimates can be overlayed on the plot, but it should be noted that the histogram does not make sense when there are covariates influencing the parameters and a warning will be displayed

saemix.plot.randeff: Boxplot of the random effects

saemix.plot.correlations: Correlation between the random effects

saemix.plot.parcov: Plots of the estimates of the individual parameters versus the covariates, using scatterplot for continuous covariates, boxplot for categorical covariates

saemix.plot.randeffcov: Plots of the estimates of the random effects versus the covariates, using scatterplot for continuous covariates, boxplot for categorical covariates

saemix.plot.npde: Plots 4 graphs to evaluate the shape of the distribution of the normalised prediction distribution errors (npde)

saemix.plot.vpc: Visual Predictive Check, with options to include the prediction intervals around the boundaries of the selected interval as well as around the median (50th percentile of the simulated data). Several methods are available to define binning on the X-axis (see methods in the PDF guide).

saemix.plot.mirror: When simulated data is available in the object (component sim.dat, which can be filled by a call to simulate), this function plots the original data as spaghetti plot and compares it to several simulated datasets under the fitted model.

Each plot can be customised by modifying options, either through a list of options set by the saemix.plot.setoptions function, or on the fly by passing an option in the call to the plot (see examples).

Value

None

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixObject,saemix, saemix.plot.setoptions, saemix.plot.select, plot.saemix

Examples

```
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,
```

```
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE)

# Not run (strict time constraints for CRAN)
saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)

# Simulate data and compute weighted residuals and npde
saemix.fit<-compute.sres(saemix.fit)

# Data
saemix.plot.data(saemix.fit)

# Convergence
saemix.plot.convergence(saemix.fit)

# Individual plot for subject 1, smoothed
saemix.plot.fits(saemix.fit,ilist=1,smooth=TRUE)

# Individual plot for subject 1 to 12, with ask set to TRUE
# (the system will pause before a new graph is produced)
saemix.plot.fits(saemix.fit,ilist=c(1:12),ask=TRUE)

# Mirror plots (plots of simulated data compared to the original)
saemix.plot.mirror(saemix.fit)

# Diagnostic plot: observations versus population predictions
par(mfrow=c(1,1))
saemix.plot.obsvspred(saemix.fit,level=0,new=FALSE)

# LL by Importance Sampling
saemix.plot.llis(saemix.fit)

# Boxplot of random effects
saemix.plot.randeff(saemix.fit)

# Relationships between parameters and covariates
saemix.plot.parcov(saemix.fit)

# Relationships between parameters and covariates, on the same page
par(mfrow=c(3,2))
saemix.plot.parcov(saemix.fit,new=FALSE)

# Scatter plot of residuals
# Not run
# Works interactively but not in the contained environment of CRAN (it looks for a datafile
# instesad of finding the dataset in the environment)
# saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)
```

```
# npde.obj<-npdeSaemix(saemix.fit)
# plot(npde.obj)
# VPC, default options (10 bins, equal number of observations in each bin)
# plot(npde.obj, plot.type="vpc")
# plot(npde.obj, plot.type="covariates")
# plot(npde.obj, plot.type="cov.x.scatter")
# plot(npde.obj, plot.type="cov.ecdf")
# VPC, user-defined breaks for binning
# plot(npde.obj, plot.type="vpc", bin.method="user", bin.breaks=c(0.4,0.8,1.5,2.5,4,5.5,8,10,13))
```

`saemix.plot.select` *Plots of the results obtained by SAEM*

Description

Several plots (selectable by the type argument) are currently available: convergence plot, individual plots, predictions versus observations, distribution plots, residual plots, VPC.

Usage

```
saemix.plot.select(
  saemixObject,
  data = FALSE,
  convergence = FALSE,
  likelihood = FALSE,
  individual.fit = FALSE,
  population.fit = FALSE,
  both.fit = FALSE,
  observations.vs.predictions = FALSE,
  residuals.scatter = FALSE,
  residuals.distribution = FALSE,
  random.effects = FALSE,
  correlations = FALSE,
  parameters.vs.covariates = FALSE,
  randeff.vs.covariates = FALSE,
  marginal.distribution = FALSE,
  vpc = FALSE,
  npde = FALSE,
  ...
)
```

Arguments

`saemixObject`	an object returned by the `saemix` function
`data`	if TRUE, produce a plot of the data. Defaults to FALSE

`convergence` if TRUE, produce a convergence plot. Defaults to FALSE

`likelihood` if TRUE, produce a plot of the estimation of the LL by importance sampling. Defaults to FALSE

`individual.fit` if TRUE, produce individual fits with individual estimates. Defaults to FALSE

`population.fit` if TRUE, produce individual fits with population estimates. Defaults to FALSE

`both.fit` if TRUE, produce individual fits with both individual and population estimates. Defaults to FALSE

`observations.vs.predictions`
if TRUE, produce a plot of observations versus predictions. Defaults to FALSE

`residuals.scatter`
if TRUE, produce scatterplots of residuals versus predictor and predictions. Defaults to FALSE

`residuals.distribution`
if TRUE, produce plots of the distribution of residuals. Defaults to FALSE

`random.effects` if TRUE, produce boxplots of the random effects. Defaults to FALSE

`correlations` if TRUE, produce a matrix plot showing the correlation between random effects. Defaults to FALSE

`parameters.vs.covariates`
if TRUE, produce plots of the relationships between parameters and covariates, using the Empirical Bayes Estimates of individual parameters. Defaults to FALSE

`randeff.vs.covariates`
if TRUE, produce plots of the relationships between random effects and covariates, using the Empirical Bayes Estimates of individual random effects. Defaults to FALSE

`marginal.distribution`
if TRUE, produce plots of the marginal distribution of the random effects. Defaults to FALSE

`vpc` if TRUE, produce Visual Predictive Check plots. Defaults to FALSE (we suggest to use `npdeSaemix` instead)

`npde` if TRUE, produce plots of the npde. Defaults to FALSE (deprecated in 3.0, please use `npdeSaemix` instead)

`...` optional arguments passed to the plots

Details

This function plots different graphs related to the algorithm (convergence plots, likelihood estimation) as well as diagnostic graphs. A description is provided in the PDF documentation.

data A spaghetti plot of the data, displaying the observed data y as a function of the regression variable (eg time for a PK application)

convergence For each parameter in the model, this plot shows the evolution of the parameter estimate versus the iteration number

likelihood Estimation of the likelihood estimated by importance sampling, as a function of the number of MCMC samples

individual.fit Individual fits, using the individual parameters with the individual covariates

population.fit Individual fits, using the population parameters with the individual covariates

both.fit Individual fits, using the population parameters with the individual covariates and the individual parameters with the individual covariates

observations.vs.predictions Plot of the predictions computed with the population parameters versus the observations (left), and plot of the predictions computed with the individual parameters versus the observations (right)

residuals.scatter Scatterplot of standardised residuals versus the X predictor and versus predictions. These plots are shown for individual and population residuals, as well as for npde when they are available

residuals.distribution Distribution of standardised residuals, using histograms and QQ-plot. These plots are shown for individual and population residuals, as well as for npde when they are available

random.effects Boxplot of the random effects

correlations Correlation between the random effects, with a smoothing spline

parameters.versus.covariates Plots of the estimate of the individual parameters versus the covariates, using scatterplot for continuous covariates, boxplot for categorical covariates

randeff.versus.covariates Plots of the estimate of the individual random effects versus the covariates, using scatterplot for continuous covariates, boxplot for categorical covariates

marginal.distribution Distribution of each parameter in the model (conditional on covariates when some are included in the model)

npde Plot of npde as in package npde (deprecated in 3.0, please use `npdeSaemix` instead)

vpc Visual Predictive Check (we suggest to use `npdeSaemix` instead)

Value

None

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixObject,saemix, default.saemix.plots, saemix.plot.setoptions, saemix.plot.data, saemix.plot.convergence, saemix.plot.llis, saemix.plot.randeff, saemix.plot.obsvspred, saemix.plot.fits, saemix.plot.parcov, saemix.plot.randeffcov, saemix.plot.distpsi, npdeSaemix saemix.plot.scatterresiduals, saemix.plot.distribresiduals, saemix.plot.vpc

Examples

```
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")

saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE)

# Not run (strict time constraints for CRAN)
# saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)

# saemix.plot.select(saemix.fit,data=TRUE,main="Spaghetti plot of data")

# Putting several graphs on the same plot
# par(mfrow=c(2,2))
# saemix.plot.select(saemix.fit,data=TRUE,vpc=TRUE,observations.vs.predictions=TRUE, new=FALSE)
```

---

saemix.plot.setoptions

Function setting the default options for the plots in SAEM

Description

- ablinecol Color of the lines added to the plots (default: "DarkRed")
- ablinelty Type of the lines added to the plots. Defaults to 2 (dashed line)
- ablinelwd Width of the lines added to the plots (default: 2)
- ask A logical value. If TRUE, users will be prompted before each new plot. Defaults to FALSE
- cex A numerical value giving the amount by which plotting text and symbols should be magnified relative to the default. Defaults to 1 (no magnification)
- cex.axis Magnification to be used for axis annotation relative to the current setting of 'cex'. Defaults to 1 (no magnification)
- cex.main Magnification to be used for main titles relative to the current setting of 'cex'. Defaults to 1 (no magnification)
- cex.lab Magnification to be used for x and y labels relative to the current setting of 'cex'. Defaults to 1 (no magnification)
- col.fillmed For the VPC plots: color filling the prediction interval for the median. Defaults to "pink"
- col.fillpi For the VPC plots: color filling the prediction interval for the limits of the prediction interval. Defaults to "slategray1"
- col.lmed For the VPC plots: color of the line showing the median of the simulated data. Defaults to "indianred4"
- col.lobs For the VPC plots: color of the lines showing the median, 2.5 and 97.5th percentiles (for a 95% prediction interval). Defaults to "steelblue4"
- col.lpi For the VPC plots: color of the line showing the boundaries of the prediction intervals. Defaults to "slategray4"
- col.obs For the VPC plots: color used to plot the observations. Defaults to "steelblue4"
- cov.name Name of the covariate to be used in the plots. Defaults to the first covariate in the model
- cov.value Value of the covariate to be used in the plots. Defaults to NA, indicating that the median value of the covariate (for continuous covariates) or the reference category (for categorical covariates) will be used
- ilist List of indices of subjects to be included in the individual plots (defaults to all subjects)
- indiv.par a string, giving the type of the individual estimates ("map"= conditional mode, "eap"=conditional mean). Defaults to conditional mode
- lcol Main line color (default: black)
- line.smooth Type of smoothing when a smoothed line is used in the plot ("m": mean value, "l": linear regression; "s": natural splines). Several options may be combined, for instance "ls" will add both a linear regression line and a line representing the fit of a natural spline. Defaults to "s"
- lty Line type. Defaults to 1, corresponding to a straight line
- lty.lmed For the VPC plots: type of the line showing the median of the simulated data. Defaults to 2 (dashed)

- lty.obs For the VPC plots: type of the line showing the observed data. Defaults to 1
- lty.lpi For the VPC plots: type of the line showing the boundaries of the simulated data. Defaults to 2 (dashed)
- lwd Line width (default: 1)
- lwd.lmed For the VPC plots: thickness of the line showing the median of the simulated data. Defaults to 2
- lwd.obs For the VPC plots: thickness of the line showing the median and boundaries of the observed data. Defaults to 2
- lwd.lpi For the VPC plots: thickness of the line showing the boundaries of the simulated data. Defaults to 1
- par.name Name of the parameter to be used in the plots. Defaults to the first parameter in the model
- pch Symbol type. Defaults to 20, corresponding to small dots
- pcol Main symbol color (default: black)
- range Range (expressed in number of SD) over which to plot the marginal distribution. Defaults to 4, so that the random effects for the marginal distribution is taken over the range [-4 SD; 4 SD]
- res.plot Type of residual plot ("res.vs.x": scatterplot versus X, "res.vs.pred": scatterplot versus predictions, "hist": histogram, "qqplot": QQ-plot) (default: "res.vs.x")
- smooth When TRUE, smoothed lines are added in the plots of predictions versus observations (default: FALSE)
- tit Title of the graph (default: none)
- type Type of the plot (as in the *R* plot function. Defaults to "b", so that both lines and symbols are shown
- units Name of the predictor used in the plots (X). Defaults to the name of the first predictor in the model (saemix.data$names$predictors[1])
- vpc.bin Number of binning intervals when plotting the VPC (the (vpc.bin-1) breakpoints are taken as the empirical quantiles of the X data). Defaults to 10
- vpc.interval Size of the prediction intervals.Defaults to 0.95 for the 95\
- vpc.obs Should the observations be overlayed on the VPC plot. Defaults to TRUE
- vpc.pi Should prediction bands be computed around the median and the bounds of the prediction intervals for the VPC. Defaults to TRUE
- xlab Label for the X-axis. Defaults to the name of the X predictor followed by the unit in bracket (eg "Time (hr)")
- xlim Range for the X-axis. Defaults to NA, indicating that the range is to be set by the plot function
- xlog A logical value. If TRUE, a logarithmic scale is in use. Defaults to FALSE
- xname Name of the predictor used in the plots (X)
- ylab Label for the Y-axis. Defaults to the name of the response followed by the unit in bracket (eg "Concentration (mg/L)" (Default: none)
- ylim Range for the Y-axis. Defaults to NA, indicating that the range is to be set by the plot function

- ylog A logical value. If TRUE, a logarithmic scale is in use. Defaults to FALSE

Plotting a SaemixData object also allows the following options:

individual if TRUE, plots separate plots for each individual, otherwise plots a spaghetti plot of all the data. Defaults to FALSE

limit for individual plots, plots only a limited number of subjets (nmax). Defaults to TRUE

nmax for individual plots, when limit is TRUE, the maximum number of plots to produce. Defaults to 12

sample for individual plots, if TRUE, randomly samples nmax different subjects to plot. Defaults to FALSE (the first nmax subjects are used in the plots)

Usage

```
saemix.plot.setoptions(saemixObject)
```

Arguments

`saemixObject`	an object returned by the `saemix` function

Details

This function can be used to create a list containing the default options and arguments used by the plot functions.

A more detailed description of the options set via these lists is provided in the PDF documentation. The "replace" functions are helper functions used within the plot functions. saemix.plot.setoptions has more available options than saemix.data.setoptions since it applies to all possible plots while the latter only applies to data.

Value

A list containing the options set at their default value. This list can be stored in an object and its elements modified to provide suitable graphs.

Author(s)

Emmanuelle Comets <emmanuelle.comets@inserm.fr>, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixObject,saemix, saemix.plot.data, saemix.plot.convergence, saemix.plot.llis, saemix.plot.randeff, saemix.plot.obsvspred, saemix.plot.fits, saemix.plot.parcov, saemix.plot.distpsi, saemix.plot.scatterres saemix.plot.vpc, saemix.plot.mirror

Examples

```
# Theophylline example, after a call to fit.saemix (see examples)
# Not run
# sopt<-saemix.plot.setoptions(saemix.fit)
# sopt$ask<-TRUE
```

---

saemix.predict *Compute model predictions after an saemix fit*

Description

In nonlinear mixed effect models, different types of predictions may be obtained, including individual predictions and population predictions. This function takes an SaemixObject and adds any missing predictions for maximum a posteriori and conditional mean estimations of the individual parameters, and for the different types of individual and population predictions for the response variable.

Usage

```
saemix.predict(object, type = c("ipred", "ypred", "ppred", "icpred"))
```

Arguments

object	an SaemixObject object
type	the type of predictions (ipred= individual, ppred=population predictions obtained with the population estimates, ypred=mean of the population predictions, icpred=conditional predictions). By default, computes all the predictions and residuals, along with the corresponding parameter estimates

Details

This function is used internally by saemix to automatically compute a number of elements needed for diagnostic plots. It is normally executed directly during a call to saemix() but can be called to add residuals

Value

an updated SaemixObject object

saemixControl *List of options for running the algorithm SAEM*

Description

List containing the variables relative to the optimisation algorithm. All these elements are optional and will be set to default values when running the algorithm if they are not specified by the user.

Usage

```
saemixControl(
  map = TRUE,
  fim = TRUE,
  ll.is = TRUE,
  ll.gq = FALSE,
  nbiter.saemix = c(300, 100),
  nbiter.sa = NA,
  nb.chains = 1,
  fix.seed = TRUE,
  seed = 23456,
  nmc.is = 5000,
  nu.is = 4,
  print.is = FALSE,
  nbdisplay = 100,
  displayProgress = FALSE,
  nbiter.burn = 5,
  nbiter.map = 5,
  nbiter.mcmc = c(2, 2, 2, 0),
  proba.mcmc = 0.4,
  stepsize.rw = 0.4,
  rw.init = 0.5,
  alpha.sa = 0.97,
  nnodes.gq = 12,
  nsd.gq = 4,
  maxim.maxiter = 100,
  nb.sim = 1000,
  nb.simpred = 100,
  ipar.lmcmc = 50,
  ipar.rmcmc = 0.05,
  print = TRUE,
  save = TRUE,
  save.graphs = TRUE,
  directory = "newdir",
  warnings = FALSE
)
```

Arguments

map	a boolean specifying whether to estimate the individual parameters (MAP estimates). Defaults to TRUE
fim	a boolean specifying whether to estimate the Fisher Information Matrix and derive the estimation errors for the parameters. Defaults to TRUE. The linearised approximation to the log-likelihood is also computed in the process
ll.is	a boolean specifying whether to estimate the log-likelihood by importance sampling. Defaults to TRUE
ll.gq	a boolean specifying whether to estimate the log-likelihood by Gaussian quadrature. Defaults to FALSE
nbiter.saemix	nb of iterations in each step (a vector containing 2 elements, nbiter.saemix[1] for the exploration phase of the algorithm (K1) and nbiter.saemix[2] for the smoothing phase (K2))
nbiter.sa	nb of iterations subject to simulated annealing (defaults to nbiter.saemix[1]/2, will be cut down to K1=nbiter.saemix[1] if greater than that value). We recommend to stop simulated annealing before the end of the exploration phase (nbiter.saemix[1]).
nb.chains	nb of chains to be run in parallel in the MCMC algorithm (defaults to 1)
fix.seed	TRUE (default) to use a fixed seed for the random number generator. When FALSE, the random number generator is initialised using a new seed, created from the current time. Hence, different sessions started at (sufficiently) different times will give different simulation results. The seed is stored in the element seed of the options list.
seed	seed for the random number generator. Defaults to 123456
nmc.is	nb of samples used when computing the likelihood through importance sampling
nu.is	number of degrees of freedom of the Student distribution used for the estimation of the log-likelihood by Importance Sampling. Defaults to 4
print.is	when TRUE, a plot of the likelihood as a function of the number of MCMC samples when computing the likelihood through importance sampling is produced and updated every 500 samples. Defaults to FALSE
nbdisplay	nb of iterations after which to display progress
displayProgress	when TRUE, the convergence plots are plotted after every nbdisplay iteration, and a dot is written in the terminal window to indicate progress. When FALSE, plots are not shown and the algorithm runs silently. Defaults to FALSE
nbiter.burn	nb of iterations for burning
nbiter.map	nb of iterations of the MAP kernel (4th kernel) to run at the beginning of the estimation process (defaults to nbiter.saemix[1]/10 if nbiter.mcmc[4] is more than 0) (EXPERIMENTAL, see Karimi et al. 2019 for details)
nbiter.mcmc	nb of iterations in each kernel during the MCMC step
proba.mcmc	probability of acceptance
stepsize.rw	stepsize for kernels q2 and q3 (defaults to 0.4)

rw.init	initial variance parameters for kernels (defaults to 0.5)
alpha.sa	parameter controlling cooling in the Simulated Annealing algorithm (defaults to 0.97)
nnodes.gq	number of nodes to use for the Gaussian quadrature when computing the likelihood with this method (defaults to 12)
nsd.gq	span (in SD) over which to integrate when computing the likelihood by Gaussian quadrature. Defaults to 4 (eg 4 times the SD)
maxim.maxiter	Maximum number of iterations to use when maximising the fixed effects in the algorithm. Defaults to 100
nb.sim	number of simulations to perform to produce the VPC plots or compute npde. Defaults to 1000
nb.simpred	number of simulations used to compute mean predictions (ypred element), taken as a random sample within the nb.sim simulations used for npde
ipar.lmcmc	number of iterations required to assume convergence for the conditional estimates. Defaults to 50
ipar.rmcmc	confidence interval for the conditional mean and variance. Defaults to 0.95
print	whether the results of the fit should be printed out. Defaults to TRUE
save	whether the results of the fit should be saved to a file. Defaults to TRUE
save.graphs	whether diagnostic graphs and individual graphs should be saved to files. Defaults to TRUE
directory	the directory in which to save the results. Defaults to "newdir" in the current directory
warnings	whether warnings should be output during the fit. Defaults to FALSE

Details

All the variables are optional and will be set to their default value when running saemix.

The function saemix returns an object with an element options containing the options used for the algorithm, with defaults set for elements which have not been specified by the user.

These elements are used in subsequent functions and are not meant to be used directly.

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

B Karimi, M Lavielle , E Moulines E (2019). f-SAEM: A fast Stochastic Approximation of the EM algorithm for nonlinear mixed effects models. Computational Statistics & Data Analysis, 141:123-38

See Also

SaemixData,SaemixModel, SaemixObject, saemix

Examples

```
# All default options
saemix.options<-saemixControl()

# All default options, changing seed
saemix.options<-saemixControl(seed=632545)
```

`saemixData` *Function to create an SaemixData object*

Description

This function creates an SaemixData object. The only mandatory argument is the name of the dataset. If the dataset has a header (or named columns), the program will attempt to detect which column correspond to ID, predictor(s) and response. Warning messages will be printed during the object creation and should be read for details.

Usage

```
saemixData(
  name.data,
  header,
  sep,
  na,
  name.group,
  name.predictors,
  name.response,
  name.X,
  name.covariates = c(),
  name.genetic.covariates = c(),
  name.mdv = "",
  name.cens = "",
  name.occ = "",
  name.ytype = "",
  units = list(x = "", y = "", covariates = c()),
  verbose = TRUE,
  automatic = TRUE
)
```

Arguments

name.data	name of the dataset (can be a character string giving the name of a file on disk or of a dataset in the R session, or the name of a dataset
header	whether the dataset/file contains a header. Defaults to TRUE
sep	the field separator character. Defaults to any number of blank spaces ("")
na	a character vector of the strings which are to be interpreted as NA values. Defaults to c(NA)
name.group	name (or number) of the column containing the subject id
name.predictors	name (or number) of the column(s) containing the predictors (the algorithm requires at least one predictor x)
name.response	name (or number) of the column containing the response variable y modelled by predictor(s) x
name.X	name of the column containing the regression variable to be used on the X axis in the plots (defaults to the first predictor)
name.covariates	name (or number) of the column(s) containing the covariates, if present (otherwise missing)
name.genetic.covariates	name (or number) of the column(s) containing the covariates, if present (otherwise missing)
name.mdv	name of the column containing the indicator for missing variable
name.cens	name of the column containing the indicator for censoring
name.occ	name of the column containing the occasion
name.ytype	name of the column containing the index of the response
units	list with up to three elements, x, y and optionally covariates, containing the units for the X and Y variables respectively, as well as the units for the different covariates (defaults to empty)
verbose	a boolean indicating whether messages should be printed out during the creation of the object
automatic	a boolean indicating whether to attempt automatic name recognition when some colum names are missing or wrong (defaults to TRUE)

Details

This function is the user-friendly constructor for the SaemixData object class. The read.saemixData is a helper function, used to read the dataset, and is not intended to be called directly.

This function is the user-friendly constructor for the SaemixData object class. The read is a helper function, used to read the dataset, and is not intended to be called directly.

Value

An SaemixData object (see saemixData).

Author(s)

Emmanuelle Comets <emmanuelle.comets@inserm.fr>, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixData,SaemixModel, saemixControl,saemix

Examples

```
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

print(saemix.data)

plot(saemix.data)
```

SaemixData-class *Class "SaemixData"*

Description

An object of the SaemixData class, representing a longitudinal data structure, used by the SAEM algorithm.

Slots

`name.data` Object of class `"character"`: name of the dataset

`header` Object of class `"logical"`: whether the dataset/file contains a header. Defaults to TRUE

`sep` Object of class `"character"`: the field separator character

`na` Object of class `"character"`: a character vector of the strings which are to be interpreted as NA values

`messages` Object of class `"logical"`: if TRUE, the program will display information about the creation of the data object

`automatic` Object of class `"logical"`: if TRUE, automatic name recognition is on (used at the creation of the object)

`name.group` Object of class `"character"`: name of the column containing the subject id

`name.predictors` Object of class `"character"`: name of the column(s) containing the predictors

`name.response` Object of class `"character"`: name of the column containing the response variable y modelled by predictor(s) x

`name.covariates` Object of class `"character"`: name of the column(s) containing the covariates, if present (otherwise empty)

`name.X` Object of class `"character"`: name of the column containing the regression variable to be used on the X axis in the plots

`name.mdv` Object of class `"character"`: name of the column containing the indicator variable denoting missing data

`name.cens` Object of class `"character"`: name of the column containing the indicator variable denoting censored data (the value in the name.response column will be taken as the censoring value)

`name.occ` Object of class `"character"`: name of the column containing the value of the occasion

`name.ytype` Object of class `"character"`: name of the column containing the response number

`trans.cov` Object of class `"list"`: the list of transformations leading to the new covariate # note: there could be a covariate class

`units` Object of class `"list"`: list with up to three elements, x, y and optionally covariates, containing the units for the X and Y variables respectively, as well as the units for the different covariates

`data` Object of class `"data.frame"`: dataframe containing the data, with columns for id (name.group), predictors (name.predictors), response (name.response), and covariates if present in the dataset (name.covariates). A column "index" contains the subject index (used to map the subject id). The column names, except for the additional column index, correspond to the names in the original dataset.

`N` Object of class `"numeric"`: number of subjects

`yorig` Object of class `"numeric"`: response data, on the original scale. Used when the error model is exponential

`ocov` Object of class `"data.frame"`: original covariate data (before transformation in the algorithm)

`ind.gen` Object of class `"logical"`: indicator for genetic covariates (internal)

`ntot.obs` Object of class `"numeric"`: total number of observations

`nind.obs` Object of class `"numeric"`: vector containing the number of observations for each subject

Objects from the Class

An object of the SaemixData class can be created by using the function `saemixData` and contain the following slots:

Methods

[<- `signature(x = "SaemixData")`: replace elements of object

[`signature(x = "SaemixData")`: access elements of object

initialize `signature(.Object = "SaemixData")`: internal function to initialise object, not to be used

plot `signature(x = "SaemixData")`: plot the data

print `signature(x = "SaemixData")`: prints details about the object (more extensive than show)

read `signature(object = "SaemixData")`: internal function, not to be used

showall `signature(object = "SaemixData")`: shows all the elements in the object

show `signature(object = "SaemixData")`: prints details about the object

summary `signature(object = "SaemixData")`: summary of the data. Returns a list with a number of elements extracted from the dataset (N: the number of subjects; nobs: the total number of observations; nind.obs: a vector giving the number of observations for each subject; id: subject ID; x: predictors; y: response, and, if present in the data, covariates: the covariates (as many lines as observations) and ind.covariates: the individual covariates (one line per individual).

subset `signature(object = "SaemixData")`: extract part of the data; this function will operate on the rows of the dataset (it can be used for instance to extract the data corresponding to the first ten subjects)

Author(s)

Emmanuelle Comets <emmanuelle.comets@inserm.fr>

Audrey Lavenu

Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

`saemixData` `SaemixModel` `saemixControl` `saemix`

Examples

```
showClass("SaemixData")

# Specifying column names
data(theo.saemix)
saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

# Specifying column numbers
data(theo.saemix)
saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=1,name.predictors=c(2,3),name.response=c(4), name.covariates=5:6,
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

# No column names specified, using automatic recognition of column names
data(PD1.saemix)
saemix.data<-saemixData(name.data=PD1.saemix,header=TRUE,
  name.covariates=c("gender"),units=list(x="mg",y="-",covariates=c("-")))
```

---

saemixModel *Function to create an SaemixModel object*

Description

This function creates an SaemixModel object. The two mandatory arguments are the name of a R function computing the model in the SAEMIX format (see details and examples) and a matrix psi0 giving the initial estimates of the fixed parameters in the model, with one row for the population mean parameters and one row for the covariate effects (see documentation).

Usage

```
saemixModel(
  model,
  psi0,
  description = "",
  modeltype = "structural",
  name.response = "",
  name.sigma = character(),
  error.model = character(),
  transform.par = numeric(),
  fixed.estim = numeric(),
  covariate.model = matrix(nrow = 0, ncol = 0),
  covariance.model = matrix(nrow = 0, ncol = 0),
  omega.init = matrix(nrow = 0, ncol = 0),
  error.init = numeric(),
```

```
  name.modpar = character(),
  simulate.function = NULL,
  verbose = TRUE
)
```

Arguments

model	name of the function used to compute the structural model. The function should return a vector of predicted values given a matrix of individual parameters, a vector of indices specifying which records belong to a given individual, and a matrix of dependent variables (see example below).
psi0	a matrix with a number of columns equal to the number of parameters in the model, and one (when no covariates are available) or two (when covariates enter the model) giving the initial estimates for the fixed effects. The column names of the matrix should be the names of the parameters in the model, and will be used in the plots and the summaries. When only the estimates of the mean parameters are given, psi0 may be a named vector.
description	a character string, giving a brief description of the model or the analysis
modeltype	a character string, giving model type (structural or likelihood)
name.response	the name of the dependent variable
name.sigma	a vector of character string giving the names of the residual error parameters
error.model	type of residual error model (valid types are constant, proportional, combined and exponential). Defaults to constant
transform.par	the distribution for each parameter (0=normal, 1=log-normal, 2=probit, 3=logit). Defaults to a vector of 1s (all parameters have a log-normal distribution)
fixed.estim	whether parameters should be estimated (1) or fixed to their initial estimate (0). Defaults to a vector of 1s
covariate.model	a matrix giving the covariate model. Defaults to no covariate in the model
covariance.model	a square matrix of size equal to the number of parameters in the model, giving the variance-covariance matrix of the model: 1s correspond to estimated variances (in the diagonal) or covariances (off-diagonal elements). Defaults to the identity matrix
omega.init	a square matrix of size equal to the number of parameters in the model, giving the initial estimate for the variance-covariance matrix of the model.
error.init	a vector of size 2 giving the initial value of a and b in the error model. Defaults to 1 for each estimated parameter in the error model
name.modpar	names of the model parameters, if they are not given as the column names (or names) of psi0
simulate.function	for non-Gaussian data models, defined as modeltype='likelihood', the name of the function used to simulate from the structural model. The function should have the same header as the model function, and should return a vector of simulated values given a matrix of individual parameters, a vector of indices specifying which records belong to a given individual, and a matrix of dependent variables (see example in the documentation, section discrete data examples)

verbose — a boolean, controlling whether information about the created should be printed out. Defaults to TRUE

Details

This function is the user-friendly constructor for the SaemixModel object class.

Value

An SaemixModel object (see saemixModel).

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixData,SaemixModel, saemixControl,saemix

Examples

```
model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}

saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")
```

SaemixModel-class *Class "SaemixModel"*

Description

An object of the SaemixModel class, representing a nonlinear mixed-effect model structure, used by the SAEM algorithm.

Objects from the Class

An object of the SaemixModel class can be created by using the function saemixModel and contain the following slots:

`model`: Object of class `"function"`: name of the function used to get predictions from the model (see the User Guide and the online examples for the format and what this function should return).

`description`: Object of class `"character"`: an optional text description of the model

`psi0`: Object of class `"matrix"`: a matrix with named columns containing the initial estimates for the parameters in the model (first line) and for the covariate effects (second and subsequent lines, optional). The number of columns should be equal to the number of parameters in the model.

`simulate.function`: Object of class `"function"`: for non-Gaussian data models, name of the function used to simulate from the model.

`transform.par`: Object of class `"numeric"`: vector giving the distribution for each model parameter (0: normal, 1: log-normal, 2: logit, 3: probit). Its length should be equal to the number of parameters in the model.

`fixed.estim`: Object of class `"numeric"`: for each parameter, 0 if the parameter is fixed and 1 if it should be estimated. Defaults to a vector of 1 (all parameters are estimated). Its length should be equal to the number of parameters in the model.

`error.model`: Object of class `"character"`: name of the error model. Valid choices are "constant" (default), "proportional" and "combined" (see equations in User Guide, except for combined which was changed to y = f + sqrt(a^2+b^2*f^2)*e)

`covariate.model`: Object of class `"matrix"`: a matrix of 0's and 1's, with a 1 indicating that a parameter-covariate relationship is included in the model (and an associated fixed effect will be estimated). The nmuber of columns should be equal to the number of parameters in the model and the number of rows to the number of covariates.

`covariance.model`: Object of class `"matrix"`: a matrix f 0's and 1's giving the structure of the variance-covariance matrix. Defaults to the Identity matrix (diagonal IIV, no correlations between parameters)

`omega.init`: Object of class `"matrix"`: a matrix giving the initial estimate for the variance-covariance matrix

`error.init`: Object of class `"numeric"`: a vector giving the initial estimate for the parameters of the residual error

Additional elements are added to the model object after a call to `saemix` and are used in the algorithm.

Methods

[<- `signature(x = "SaemixModel")`: replace elements of object

[`signature(x = "SaemixModel")`: access elements of object

initialize `signature(.Object = "SaemixModel")`: internal function to initialise object, not to be used

plot `signature(x = "SaemixModel")`: plot predictions from the model

print `signature(x = "SaemixModel")`: prints details about the object (more extensive than show)

showall `signature(object = "SaemixModel")`: shows all the elements in the object

show `signature(object = "SaemixModel")`: prints details about the object

Author(s)

Emmanuelle Comets <emmanuelle.comets@inserm.fr>

Audrey Lavenu

Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

`SaemixData` `SaemixObject` `saemixControl` `saemix` `plot.saemix`

Examples

```
showClass("SaemixModel")

# Model function for continuous data
## structural model: a one-compartment model with oral absorption
model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}
```

```
# Corresponding SaemixModel, assuming starting parameters ka=1, V=20, CL=0.5
# and log-normal distributions for the parameters
model1 <-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1,20,0.5),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1))

# Model function for discrete data
## logistic regression for the probability of the observed outcome
binary.model<-function(psi,id,xidep) {
  tim<-xidep[,1]
  y<-xidep[,2]
  inter<-psi[id,1]
  slope<-psi[id,2]
  logit<-inter+slope*tim
  pevent<-exp(logit)/(1+exp(logit))
  pobs = (y==0)*(1-pevent)+(y==1)*pevent
  logpdf <- log(pobs)
  return(logpdf)
}

## Corresponding SaemixModel, assuming starting parameters inter=-5, slope=-1
# and normal distributions for both parameters
# note that the modeltype argument is set to likelihood
saemix.model<-saemixModel(model=binary.model,description="Binary model",
     modeltype="likelihood",
    psi0=matrix(c(-5,-.1,0,0),ncol=2,byrow=TRUE,dimnames=list(NULL,c("inter","slope"))),
     transform.par=c(0,0))

## saemix cannot infer the distribution of the outcome directly from the model
## Here we therefore define a simulation function, needed for diagnostics
### Note the similarity and differences with the model function
simulBinary<-function(psi,id,xidep) {
    tim<-xidep[,1]
    y<-xidep[,2]
    inter<-psi[id,1]
    slope<-psi[id,2]
    logit<-inter+slope*tim
    pevent<-1/(1+exp(-logit))
    ysim<-rbinom(length(tim),size=1, prob=pevent)
    return(ysim)
    }

saemix.model<-saemixModel(model=binary.model,description="Binary model",
     modeltype="likelihood", simulate.function=simulBinary,
    psi0=matrix(c(-5,-.1,0,0),ncol=2,byrow=TRUE,dimnames=list(NULL,c("inter","slope"))),
     transform.par=c(0,0))
```

SaemixObject-class *Class "SaemixObject"*

Description

An object of the SaemixObject class, storing the input to saemix, and the results obtained by a call to the SAEM algorithm

Details

Details of the algorithm can be found in the pdf file accompanying the package.

Objects from the Class

An object of the SaemixObject class is created after a call to `saemix` and contain the following slots:

`data`: Object of class `"SaemixData"`: saemix dataset, created by a call to `saemixData`

`model`: Object of class `"SaemixModel"`: saemix model, created by a call to `saemixModel`

`results`: Object of class `"SaemixData"`: saemix dataset, created by a call to `saemixData`

`rep.data`: Object of class `"SaemixRepData"`: (internal) replicated saemix dataset, used the execution of the algorithm

`sim.data`: Object of class `"SaemixSimData"`: simulated saemix dataset

`options`: Object of class `"list"`: list of settings for the algorithm

`prefs`: Object of class `"list"`: list of graphical options for the graphs

Methods

[<- `signature(x = "SaemixObject")`: replace elements of object

[`signature(x = "SaemixObject")`: access elements of object

initialize `signature(.Object = "SaemixObject")`: internal function to initialise object, not to be used

plot `signature(x = "SaemixObject")`: plot the data

print `signature(x = "SaemixObject")`: prints details about the object (more extensive than show)

showall `signature(object = "SaemixObject")`: shows all the elements in the object

show `signature(object = "SaemixObject")`: prints details about the object

summary `signature(object = "SaemixObject")`: summary of the object. Returns a list with a number of elements extracted from the object.

Author(s)

Emmanuelle Comets <emmanuelle.comets@inserm.fr>

Audrey Lavenu

Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

SaemixData SaemixModel saemixControl saemix plot.saemix,

Examples

```
showClass("SaemixObject")
```

`saemixPredictNewdata` *Predictions for a new dataset*

Description

Predictions for a new dataset

Usage

```
saemixPredictNewdata(
  saemixObject,
  newdata,
  type = c("ipred", "ypred", "ppred", "icpred"),
  nsamp = 1,
  max.iter = NULL
)
```

Arguments

`saemixObject`	an SaemixObject from a fitted run
`newdata`	a dataframe containing the new data. The dataframe must contain the same information as the original dataset (column names, etc...)
`type`	one or several of "ipred" (individual predictions using the MAP estimates), "ppred" (population predictions obtained using the population parameters f(E(theta))), "ypred" (mean of the population predictions (E(f(theta)))), "icpred" (individual predictions using the conditional mean estimates). Defaults to "ppred".

`nsamp` an integer, ignored for other types than icpred; if icpred, returns both the mean of the conditional distribution and nsamp samples, with the corresponding predictions. Defaults to 1.

`max.iter` an integer, ignored for other types than icpred; for icpred, maximum number of iterations to run for the conditional distribution. Defaults to NULL, in which case the maximum number of iterations will be set to the total number of iterations in the population estimation algorithm. Note that a minimum of 50 iterations will be run by the algorithm if max.iter is set to a lower value than that.

Details

This function is the workhorse behind the predict method for SaemixObject. It computes predictions for a new dataframe based on the results of an saemix fit. Since the predict function only returns predicted values, this function is provided so that users can access other elements, for example the different types of parameter estimates (such as individual or population parameters) associated with the predictions For other purposes such as simply obtaining model predictions, we suggest using the predict() method with or without newdata.

The function uses estimateMeanParametersNewdata() to set the population estimates for the individual parameters taking into account the individual covariates and doses, and estimateIndividualParametersNewdata() to derive individual estimates by computing the mean of the conditional distributions (type="icpred") or the MAP estimate (type="ipred")

Value

a list with three components (five if type includes "icpred" and nsamp>1)

param a dataframe with the estimated parameters. The columns in the dataframe depend on which type of predictions were requested (argument type)

predictions a dataframe with the predictions. The columns in the dataframe depend on which type of predictions were requested (argument type)

saemixObject the SaemixObject with the data slot replaced by the new data. The elements of the results slot pertaining to individual (including population individual parameters) predictions and likelihood will have been removed, and only the elements computed within the function will have been replaced (eg individual estimated parameters and predictions for the new data)

parSample a dataframe with parameters sampled from the conditional distribution of the individual parameters (only present if type includes 'icpred' and nsamp>1)

predSample a dataframe with the predictions corresponding to the parameters sampled from the conditional distribution of the individual parameters (only present if type includes 'icpred' and nsamp>1)

Examples

```
# TODO
```

SaemixRes-class *Class "SaemixRes"*

Description

An object of the SaemixRes class, representing the results of a fit through the SAEM algorithm.

Slots

`modeltype` string giving the type of model used for analysis

`status` string indicating whether a model has been run successfully; set to "empty" at initialisation, used to pass on error messages or fit status

`name.fixed` a vector containing the names of the fixed parameters in the model

`name.random` a vector containing the names of the random parameters in the model

`name.sigma` a vector containing the names of the parameters of the residual error model

`npar.est` the number of parameters estimated (fixed, random and residual)

`nbeta.random` the number of estimated fixed effects for the random parameters in the model

`nbeta.fixed` the number of estimated fixed effects for the non random parameters in the model

`fixed.effects` a vector giving the estimated h(mu) and betas

`fixed.psi` a vector giving the estimated h(mu)

`betas` a vector giving the estimated mu

`betaC` a vector with the estimates of the fixed effects for covariates

`omega` the estimated variance-covariance matrix

`respar` the estimated parameters of the residual error model

`fim` the Fisher information matrix

`se.fixed` a vector giving the estimated standard errors of estimation for the fixed effect parameters

`se.omega` a vector giving the estimated standard errors of estimation for Omega

`se.cov` a matrix giving the estimated SE for each term of the covariance matrix (diagonal elements represent the SE on the variances of the random effects and off-diagonal elements represent the SE on the covariance terms)

`se.respar` a vector giving the estimated standard errors of estimation for the parameters of the residual variability

`conf.int` a dataframe containing the estimated parameters, their estimation error (SE), coefficient of variation (CV), and the associated confidence intervals; the variabilities for the random effects are presented first as estimated (variances) then converted to standard deviations (SD), and the correlations are computed. For SD and correlations, the SE are estimated via the delta-method

`parpop` a matrix tracking the estimates of the population parameters at each iteration

`allpar` a matrix tracking the estimates of all the parameters (including covariate effects) at each iteration

indx.fix the index of the fixed parameters (used in the estimation algorithm)

indx.cov the index of the covariance parameters (used in the estimation algorithm)

indx.omega the index of the random effect parameters (used in the estimation algorithm)

indx.res the index of the residual error model parameters (used in the estimation algorithm)

MCOV a matrix of covariates (used in the estimation algorithm)

cond.mean.phi a matrix giving the conditional mean estimates of phi (estimated as the mean of the conditional distribution)

cond.mean.psi a matrix giving the conditional mean estimates of psi (h(cond.mean.phi))

cond.var.phi a matrix giving the variance on the conditional mean estimates of phi (estimated as the variance of the conditional distribution)

cond.mean.eta a matrix giving the conditional mean estimates of the random effect eta

cond.shrinkage a vector giving the shrinkage on the conditional mean estimates of eta

mean.phi a matrix giving the population estimate (Ci*mu) including covariate effects, for each subject

map.psi a dataframe giving the MAP estimates of individual parameters

map.phi a dataframe giving the MAP estimates of individual phi

map.eta a matrix giving the individual estimates of the random effects corresponding to the MAP estimates

map.shrinkage a vector giving the shrinkage on the MAP estimates of eta

phi individual parameters, estimated at the end of the estimation process as the average over the chains of the individual parameters sampled during the successive E-steps

psi.samp a three-dimensional array with samples of psi from the conditional distribution

phi.samp a three-dimensional array with samples of phi from the conditional distribution

phi.samp.var a three-dimensional array with the variance of phi

ll.lin log-likelihood computed by lineariation

aic.lin Akaike Information Criterion computed by linearisation

bic.lin Bayesian Information Criterion computed by linearisation

bic.covariate.lin Specific Bayesian Information Criterion for covariate selection computed by linearisation

ll.is log-likelihood computed by Importance Sampling

aic.is Akaike Information Criterion computed by Importance Sampling

bic.is Bayesian Information Criterion computed by Importance Sampling

bic.covariate.is Specific Bayesian Information Criterion for covariate selection computed by Importance Sampling

LL a vector giving the conditional log-likelihood at each iteration of the algorithm

ll.gq log-likelihood computed by Gaussian Quadrature

aic.gq Akaike Information Criterion computed by Gaussian Quadrature

bic.gq Bayesian Information Criterion computed by Gaussian Quadrature

`bic.covariate.gq` Specific Bayesian Information Criterion for covariate selection computed by Gaussian Quadrature

`predictions` a data frame containing all the predictions and residuals in a table format

`ppred` a vector giving the population predictions obtained with the population estimates

`ypred` a vector giving the mean population predictions

`ipred` a vector giving the individual predictions obtained with the MAP estimates

`icpred` a vector giving the individual predictions obtained with the conditional estimates

`ires` a vector giving the individual residuals obtained with the MAP estimates

`iwres` a vector giving the individual weighted residuals obtained with the MAP estimates

`icwres` a vector giving the individual weighted residuals obtained with the conditional estimates

`wres` a vector giving the population weighted residuals

`npde` a vector giving the normalised prediction distribution errors

`pd` a vector giving the prediction discrepancies

Objects from the Class

An object of the SaemixData class can be created by using the function `saemixData` and contain the following slots:

Methods

[<- `signature(x = "SaemixRes")`: replace elements of object

[`signature(x = "SaemixRes")`: access elements of object

initialize `signature(.Object = "SaemixRes")`: internal function to initialise object, not to be used

print `signature(x = "SaemixRes")`: prints details about the object (more extensive than show)

read `signature(object = "SaemixRes")`: internal function, not to be used

showall `signature(object = "SaemixRes")`: shows all the elements in the object

show `signature(object = "SaemixRes")`: prints details about the object

summary `signature(object = "SaemixRes")`: summary of the results. Returns a list with a number of elements extracted from the results ().

Author(s)

Emmanuelle Comets <emmanuelle.comets@inserm.fr>

Audrey Lavenu

Marc Lavielle.

References

E Comets, A Lavenu, M Lavielle M (2017). Parameter estimation in nonlinear mixed effect models using saemix, an R implementation of the SAEM algorithm. Journal of Statistical Software, 80(3):1-41.

E Kuhn, M Lavielle (2005). Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis, 49(4):1020-1038.

E Comets, A Lavenu, M Lavielle (2011). SAEMIX, an R version of the SAEM algorithm. 20th meeting of the Population Approach Group in Europe, Athens, Greece, Abstr 2173.

See Also

saemixData SaemixModel saemixControl saemix

Examples

```
methods(class="SaemixRes")

showClass("SaemixRes")
```

---

show-methods *Methods for Function show*

Description

Prints a short summary of an object

Usage

```
## S4 method for signature 'SaemixData'
show(object)

## S4 method for signature 'SaemixRepData'
show(object)

## S4 method for signature 'SaemixSimData'
show(object)

## S4 method for signature 'SaemixModel'
show(object)

## S4 method for signature 'SaemixRes'
show(object)

## S4 method for signature 'SaemixObject'
show(object)
```

Arguments

object	an object of type SaemixData, SaemixModel, SaemixRes or SaemixObject

Methods

list("signature(x = \"ANY\")") Default show function

list("signature(x = \"SaemixData\")") Prints a short summary of a SaemixData object

list("signature(x = \"SaemixModel\")") Prints a short summary of a SaemixModel object

list("signature(x = \"SaemixObject\")") Prints a short summary of the results from a SAEMIX fit

list("signature(x = \"SaemixRes\")") Not user-level

list("signature(object = \"SaemixRepData\")") Prints a short summary of a SaemixRepData object

list("signature(object = \"SaemixSimData\")") Prints a short summary of a SaemixSimData object

---

showall-methods *Methods for Function showall*

Description

This function is used to visualise the majority of the elements of an object

Usage

```
showall(object)

## S4 method for signature 'SaemixData'
showall(object)

## S4 method for signature 'SaemixModel'
showall(object)

## S4 method for signature 'SaemixRes'
showall(object)

## S4 method for signature 'SaemixObject'
showall(object)
```

Arguments

object	showall methods are available for objects of type SaemixData, SaemixModel and SaemixObject

Value

None

Methods

list("signature(x = \"SaemixData\")") Prints a extensive summary of a SaemixData object

list("signature(x = \"SaemixModel\")") Prints a extensive summary of a SaemixModel object

list("signature(x = \"SaemixObject\")") Prints a extensive summary of the results from a SAEMIX fit

list("signature(x = \"SaemixRes\")") Not user-level

See Also

SaemixData,SaemixModel, SaemixObject

Examples

```
# A SaemixData object
data(theo.saemix)
saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")
showall(saemix.data)

# A SaemixModel object
model1cpt<-function(psi,id,xidep) {
  dose<-xidep[,1]
  tim<-xidep[,2]
  ka<-psi[id,1]
  V<-psi[id,2]
  CL<-psi[id,3]
  k<-CL/V
  ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
  return(ypred)
}
saemix.model<-saemixModel(model=model1cpt,
  description="One-compartment model with first-order absorption",
  psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3, byrow=TRUE,
  dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
  covariate.model=matrix(c(0,1,0,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
  covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),
  omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="constant")
showall(saemix.model)
```

`simulate.SaemixObject` *Perform simulations under the model for an saemixObject object*

Description

This function is used to simulate data under the empirical design, using the model and estimated parameters from a fit.

Usage

```
## S3 method for class 'SaemixObject'
simulate(
  object,
  nsim,
  seed,
  predictions,
  outcome = "continuous",
  res.var = TRUE,
  uncertainty = FALSE,
  ...
)
```

Arguments

`object`	an saemixObject object returned by the `saemix` function
`nsim`	Number of simulations to perform. Defaults to the nb.sim element in options
`seed`	if non-null, seed used to initiate the random number generator (defaults to NULL)
`predictions`	Whether the simulated parameters should be used to compute predictions. Defaults to TRUE for continuous data, and to FALSE for non-Gaussian data models. If FALSE, only individual parameters are simulated.
`outcome`	the type of outcome (used to specify TTE or RTTE models)
`res.var`	Whether residual variability should be added to the predictions. Defaults to TRUE
`uncertainty`	Uses uncertainty (currently not implemented). Defaults to FALSE
`...`	additional arguments, unused (included for compatibility with the generic)

Details

The simulated data can then be used to produce Visual Predictive Check graphs, as well as to compute the normalised prediction distribution errors (npde).

This function replaces the previous function (simul.saemix), which will be deprecated in future versions but can still be called as previously for compatibility purposes.

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Audrey Lavenu, Marc Lavielle.

References

Brendel, K, Comets, E, Laffont, C, Laveille, C, Mentre, F. Metrics for external model evaluation with an application to the population pharmacokinetics of gliclazide, Pharmaceutical Research 23 (2006), 2036-2049.

Holford, N. The Visual Predictive Check: superiority to standard diagnostic (Rorschach) plots (Abstract 738), in: 14th Meeting of the Population Approach Group in Europe, Pamplona, Spain, 2005.

See Also

SaemixObject,saemix, saemix.plot.data, saemix.plot.convergence, saemix.plot.llis, saemix.plot.randeff, saemix.plot.obsvspred, saemix.plot.fits, saemix.plot.parcov, saemix.plot.distpsi, saemix.plot.scatterres saemix.plot.vpc

---

simulateDiscreteSaemix

Perform simulations under the model for an saemixObject object defined by its log-likelihood

---

Description

This function is used to simulate from discrete response models.

Usage

```
simulateDiscreteSaemix(
  object,
  nsim,
  seed,
  predictions = TRUE,
  uncertainty = FALSE,
  verbose = FALSE
)
```

Arguments

object	an saemixObject object returned by the saemix function. The model must contain a slot simulate.function, containing a function with the same structure as the model functions (arguments (psi, id, xidep)) which returns simulated responses when given a set of individual parameters (psi) and the corresponding predictors (xidep)
nsim	Number of simulations to perform. Defaults to the nb.simpred element in options
seed	if non-null, seed used to initiate the random number generator (defaults to NULL)
predictions	Whether the simulated parameters should be used to compute predictions. Defaults to TRUE. If FALSE, only individual parameters are simulated.
uncertainty	Uses uncertainty (currently not implemented). Defaults to FALSE
verbose	if TRUE, prints messages (defaults to FALSE)

Details

To call this function, the user needs to define a simulate.function matching the model function in the object. The function will then be used to simulate data under the empirical design, using the model and estimated parameters from a fit.

This function calls simulate.SaemixObject with the prediction=FALSE option to simulate individual parameters, then the simulate.function to obtain corresponding predictions.

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr

See Also

SaemixObject,saemix, simulate.SaemixObject

---

step.saemix *Stepwise procedure for joint selection of covariates and random effects*

Description

Joint selection of covariates and random effects in a nonlinear mixed effects model by a stepwise-type algorithm based on two different versions of BIC for covariate selection and random effects selection respectively. Selection is made among the covariates as such specified in the SaemixData object. Only uncorrelated random effects structures are considered.

Usage

```
step.saemix(
  saemixObject,
  trace = TRUE,
  direction = "forward",
  covariate.init = NULL
)
```

Arguments

saemixObject	An object returned by the saemix function
trace	If TRUE, a table summarizing the steps of the algorithm is printed. Default "TRUE"
direction	The mode of stepwise search, can be one of "both", "backward", or "forward", with a default of "forward".
covariate.init	A matrix specifying the initial covariate structure to be considered in the algorithm. covariate.init is only used if the direction argument is "both".

Value

An object of the SaemixObject class storing the covariate model and the covariance structure of random effects of the final model.

Author(s)

Maud Delattre

References

M Delattre, M Lavielle, MA Poursat (2014) A note on BIC in mixed effects models. Electronic Journal of Statistics 8(1) p. 456-475 M Delattre, MA Poursat (2017) BIC strategies for model choice in a population approach. (arXiv:1612.02405)

Examples

```
data(theo.saemix)

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")

# Definition of models to be compared
model1cpt<-function(psi,id,xidep) {
   dose<-xidep[,1]
   tim<-xidep[,2]
   ka<-psi[id,1]
   V<-psi[id,2]
   CL<-psi[id,3]
   k<-CL/V
   ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
   return(ypred)
}

saemix.model1<-saemixModel(model=model1cpt,modeltype="structural",
     description="One-compartment model with first-order absorption",
     psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3,byrow=TRUE,
     dimnames=list(NULL, c("ka","V","CL"))), transform.par=c(1,1,1),
     covariate.model=matrix(c(0,0,1,0,0,0),ncol=3,byrow=TRUE))

saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE)
saemix.fit1<-saemix(saemix.model1,saemix.data,saemix.options)

## Not run:
res.forward <- step.saemix(saemix.fit1, direction = "forward")
res.backward <- step.saemix(saemix.fit1, direction = "backward")
covariate.init <- matrix(c(1,0,0,0,1,0),ncol=3,nrow=2)
res.stepwise <- step.saemix(saemix.fit1, direction="both")

## End(Not run)
```

`stepwise.procedure` *Stepwise procedure for joint selection of covariates and random effects*

Description

Joint selection of covariates and random effects in a nonlinear mixed effects model by a stepwise-type algorithm based on two different versions of BIC for covariate selection and random effects selection respectively. Selection is made among the covariates as such specified in the SaemixData object. Only uncorrelated random effects structures are considered.

Usage

```
stepwise.procedure(saemixObject, covariate.init = NULL, trace = TRUE)
```

Arguments

`saemixObject`	An object returned by the `saemix` function
`covariate.init`	A matrix specifying the initial covariate structure to be considered in the algorithm.
`trace`	If TRUE, a table summarizing the steps of the algorithm is printed. Default "TRUE"

Value

An object of the SaemixObject class storing the covariate model and the covariance structure of random effects of the final model.

Author(s)

Maud Delattre

References

M Delattre, M Lavielle, MA Poursat (2014) A note on BIC in mixed effects models. Electronic Journal of Statistics 8(1) p. 456-475 M Delattre, MA Poursat (2017) BIC strategies for model choice in a population approach. (arXiv:1612.02405)

subset *Data subsetting*

Description

Return an SaemixData object containing the subset of data which meets conditions.

Usage

```
## S3 method for class 'SaemixData'
subset(x, subset, ...)
```

Arguments

x	saemixData object
subset	logical expression indicating elements or rows to keep: missing values are taken as false
...	additional parameters (ignored)

Value

an object of class "SaemixData"

Examples

```
# TODO
```

summary-methods *Methods for Function summary*

Description

Methods for function summary

summary method for class SaemixData

Usage

```
## S4 method for signature 'SaemixData'
summary(object, print = TRUE, ...)

## S4 method for signature 'SaemixModel'
summary(object, print = TRUE, ...)

## S4 method for signature 'SaemixRes'
summary(object, print = TRUE, ...)
```

```
## S4 method for signature 'SaemixObject'
summary(object, print = TRUE, ...)
```

Arguments

object	an object of class SaemixData
print	a boolean controlling whether to print the output or return it silently
...	additional arguments (ignored)

Value

a list with a number of elements extracted from the dataset

N number of subjects

nobs the total number of observations

nind.obs a vector giving the number of observations for each subject

id subject ID; x: predictors; y: response, and, if present in the data, covariates: the covariates (as many lines as observations) and ind.covariates: the individual covariates (one

Methods

list("signature(x = \"ANY\")") default summary function ?

list("signature(x = \"SaemixData\")") summary of the data

list("signature(x = \"SaemixModel\")") summary of the model

list("signature(x = \"SaemixObject\")") summary of an SaemixObject

---

testnpde *Tests for normalised prediction distribution errors*

Description

Performs tests for the normalised prediction distribution errors returned by npde

Usage

```
testnpde(npde)
```

Arguments

npde	the vector of prediction distribution errors

Details

Given a vector of normalised prediction distribution errors (npde), this function compares the npde to the standardised normal distribution N(0,1) using a Wilcoxon test of the mean, a Fisher test of the variance, and a Shapiro-Wilks test for normality. A global test is also reported.

The helper functions `kurtosis` and `skewness` are called to compute the kurtosis and skewness of the distribution of the npde.

Value

a list containing 4 components:

`Wilcoxon test of mean=0`
: compares the mean of the npde to 0 using a Wilcoxon test

`variance test`
: compares the variance of the npde to 1 using a Fisher test

`SW test of normality`
: compares the npde to the normal distribution using a Shapiro-Wilks test

`global test`
: an adjusted p-value corresponding to the minimum of the 3 previous p-values multiplied by the number of tests (3), or 1 if this p-value is larger than 1.

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr

References

K. Brendel, E. Comets, C. Laffont, C. Laveille, and F. Mentr\'e. Metrics for external model evaluation with an application to the population pharmacokinetics of gliclazide. *Pharmaceutical Research*, 23:2036–49, 2006.

See Also

`saemix`, `saemix.plot.npde`

---

theo.saemix *Pharmacokinetics of theophylline*

Description

The `theo.saemix` data frame has 132 rows and 6 columns of data from an experiment on the pharmacokinetics of theophylline. A column with gender was added to the original data for demo purposes, and contains simulated data.

Usage

```
theo.saemix
```

Format

This data frame contains the following columns:

Id an ordered factor with levels 1, ..., 12 identifying the subject on whom the observation was made. The ordering is by time at which the observation was made.

Dose dose of theophylline administered orally to the subject (mg/kg).

Time time since drug administration when the sample was drawn (hr).

Concentration theophylline concentration in the sample (mg/L).

Weight weight of the subject (kg).

Sex gender (simulated, 0=male, 1=female

Details

Boeckmann, Sheiner and Beal (1994) report data from a study by Dr. Robert Upton of the kinetics of the anti-asthmatic drug theophylline. Twelve subjects were given oral doses of theophylline then serum concentrations were measured at 11 time points over the next 25 hours.

These data are analyzed in Davidian and Giltinan (1995) and Pinheiro and Bates (2000) using a two-compartment open pharmacokinetic model.

These data are also available in the library `datasets` under the name `Theoph` in a slightly modified format and including the data at time 0. Here, we use the file in the format provided in the *NONMEM* installation path (see the User Guide for that software for details).

Source

AJ Boeckmann, LB Sheiner, SL Beal (1994), *NONMEM Users Guide: Part V*, NONMEM Project Group, University of California, San Francisco.

References

M Davidian, DM Giltinan (1995) *Nonlinear Models for Repeated Measurement Data*, Chapman & Hall (section 5.5, p. 145 and section 6.6, p. 176)

JC Pinheiro, DM Bates (2000) *Mixed-effects Models in S and S-PLUS*, Springer (Appendix A.29)

Examples

```
data(theo.saemix)

#Plotting the theophylline data
plot(Concentration~Time,data=theo.saemix,xlab="Time after dose (hr)",
ylab="Theophylline concentration (mg/L)")

saemix.data<-saemixData(name.data=theo.saemix,header=TRUE,sep=" ",na=NA,
  name.group=c("Id"),name.predictors=c("Dose","Time"),
  name.response=c("Concentration"),name.covariates=c("Weight","Sex"),
  units=list(x="hr",y="mg/L",covariates=c("kg","-")), name.X="Time")
  model1cpt<-function(psi,id,xidep) {
    dose<-xidep[,1]
    tim<-xidep[,2]
```

```
    ka<-psi[id,1]
    V<-psi[id,2]
    CL<-psi[id,3]
    k<-CL/V
    ypred<-dose*ka/(V*(ka-k))*(exp(-k*tim)-exp(-ka*tim))
    return(ypred)
    }
# Default model, no covariate
saemix.model<-saemixModel(model=model1cpt,
       description="One-compartment model with first-order absorption",
       psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3,byrow=TRUE,
       dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1))
 # Note: remove the options save=FALSE and save.graphs=FALSE
 # to save the results and graphs
 saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE, displayProgress=FALSE)

 # Not run (strict time constraints for CRAN)
 saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)

 # Model with covariates
 saemix.model<-saemixModel(model=model1cpt,
      description="One-compartment model with first-order absorption",
      psi0=matrix(c(1.,20,0.5,0.1,0,-0.01),ncol=3,byrow=TRUE,
      dimnames=list(NULL, c("ka","V","CL"))),transform.par=c(1,1,1),
      covariate.model=matrix(c(0,0,1,0,0,0),ncol=3,byrow=TRUE),fixed.estim=c(1,1,1),
      covariance.model=matrix(c(1,0,0,0,1,1,0,1,1),ncol=3,byrow=TRUE),
      omega.init=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,byrow=TRUE),error.model="combined")

 saemix.options<-list(seed=39546,save=FALSE,save.graphs=FALSE,displayProgress=FALSE)

 # Not run (strict time constraints for CRAN)
 saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)
```

`toenail.saemix` *Toenail data*

Description

The `toenail.saemix` data are from a multicenter study comparing two oral treatments for toe-nail infection, including information for 294 patients measured at 7 weeks, comprising a total of 1908 measurements. The outcome binary variable "onycholysis" indicates the degree of separation of the nail plate from the nail-bed (none or mild versus moderate or severe). Patients were evaluated at baseline (week 0) and at weeks 4, 8, 12, 24, 36, and 48 thereafter.

Usage

```
toenail.saemix
```

Format

This data frame contains the following columns:

id subject index in file

time time of measurement (in months)

y degree of onycholysis (0 if none or mild, 1 if moderate or severe)

treatment treatment indicator (1=Treatment A, 0= Treatment B)

visit visit number (visit numbers 1-7 correspond to scheduled visits at 0, 4, 8, 12, 24, 36, and 48 weeks)

#'

Details

The data in the `toenail.saemix` was copied from the Toenail dataset provided by the prLogistic package. Different models and analyses have been performed to describe this dataset in Molenberg and Verbeke (2000). Please refer to the PDF documentation (chapter 4, section 4.6) for more details on the analysis, including how to obtain diagnostic plots.

Source

prLogistic package in R

References

M De Backer, C De Vroey, E Lesaffre, I Scheys, P De Keyser (1998). Twelve weeks of continuous oral therapy for toenail onychomycosis caused by dermatophytes: A double-blind comparative trial of terbinafine 250 mg/day versus itraconazole 200 mg/day. Journal of the American Academy of Dermatology, 38:57-63.

E Lesaffre, B Spiessens (2001). On the effect of the number of quadrature points in a logistic random-effects model: An example. Journal of the Royal Statistical Society, Series C, 50:325-335.

G Verbeke, G Molenberghs (2000). Linear mixed models for longitudinal data, Springer, New York.

S Rabe-Hesketh, A Skrondal (2008). Multilevel and Longitudinal Modeling Using Stata. Mahwah, NJ: Lawrence Erlbaum Associates. Second Edition.

Examples

```
data(toenail.saemix)
saemix.data<-saemixData(name.data=toenail.saemix,name.group=c("id"), name.predictors=c("time","y"),
 name.response="y", name.covariates=c("treatment"),name.X=c("time"))
binary.model<-function(psi,id,xidep) {
  tim<-xidep[,1]
  y<-xidep[,2]
  inter<-psi[id,1]
  slope<-psi[id,2]
  logit<-inter+slope*tim
  pevent<-exp(logit)/(1+exp(logit))
  pobs = (y==0)*(1-pevent)+(y==1)*pevent
```

```
  logpdf <- log(pobs)
  return(logpdf)
}

saemix.model<-saemixModel(model=binary.model,description="Binary model",
     modeltype="likelihood",
   psi0=matrix(c(-5,-.1,0,0),ncol=2,byrow=TRUE,dimnames=list(NULL,c("theta1","theta2"))),
     transform.par=c(0,0), covariate.model=c(0,1),
     covariance.model=matrix(c(1,0,0,1),ncol=2))

saemix.options<-list(seed=1234567,save=FALSE,save.graphs=FALSE, displayProgress=FALSE,
   nb.chains=10, fim=FALSE)
binary.fit<-saemix(saemix.model,saemix.data,saemix.options)
plot(binary.fit, plot.type="convergence")
```

`transform` *Transform covariates*

Description

Transform and/or center a vector

Usage

```
## S3 method for class 'numeric'
transform(
  `_data`,
  transformation = function(x) x,
  centering = "median",
  verbose = FALSE,
  ...
)
```

Arguments

`_data`	a vector with values of type numeric
`transformation`	transformation function. Defaults to no transformation
`centering`	string, giving the value used to center the covariate; can be "mean" or "median", in which case this value will be computed from the data, 'none' or 0 for no centering, or a value given by the user. Defaults to the median value over the dataset.
`verbose`	a boolean, prints messages during the execution of the function if TRUE. Defaults to FALSE.
`...`	unused, for consistency with the generic method

Value

a vector

Examples

```
# TODO
```

`transformCatCov` *Transform categorical covariates*

Description

This function is used to automatically map categorical covariates to dummy binary covariates. It can also be used to regroup categorical covariates before the analysis, for example mapping a covariate initially in 5 categories to only 3 categories. For binary covariates, the covariate will simply be transformed to 0/1 with the reference category being 0.

Usage

```
transformCatCov(
  object,
  covariate,
  oldCat = NULL,
  newCat = NULL,
  newCatName = NULL,
  verbose = FALSE
)
```

Arguments

`object`	saemixData object
`covariate`	name of the covariate
`oldCat`	a vector giving the initial categories of the covariate (if NULL, defaults to the unique values of the covariate)
`newCat`	a vector of consecutive integers 1:N, giving the N categories to which the initial values of the covariate should be mapped, with 1 denoting the reference class. If the resulting covariate is binary, it will be stored as 0/1. If it has more than 2 categories, N dummy covariates 0/1 will be created. If not given, the initial categories of the covariate will be mapped to 1:N and transformed into N dummy covariates 0/1.
`newCatName`	the name of the new group. If NULL (default), binary covariates will be renamed as covariate.mod, while for categorical covariates, the new covariates will be named covariate.ref, covariate.G2, covariate.G3, etc...
`verbose`	a boolean, prints messages during the execution of the function if TRUE. Defaults to FALSE.

Details

For a binary covariate, a dummy covariate with values 0 for the reference class and 1 for the other category will be created and will replace the original dataset in the covariate For covariates with 3 categories or more (categorical covariates), dummy variables with values 0/1 will be created for the reference class and for each contrast with the reference class (category 2 versus reference, category 3 versus reference, etc...) If these covariates have units, the dummy covariates will have the same unit (which may not be appropriate and can then be changed by the user)

Value

an object of class "SaemixData"

Examples

```
data(cow.saemix)
saemix.data<-saemixData(name.data=cow.saemix,header=TRUE,name.group=c("cow"),
                  name.predictors=c("time"),name.response=c("weight"),
                  name.covariates=c("birthyear","twin","birthrank"),
                  units=list(x="days",y="kg",covariates=c("yr","-","-")))
unique(saemix.data@data$birthrank) # 5 categories, 3 4 5 6 7
# create 3 dummy variables regrouping 3 (reference), 4 and 5, and 6 and 7
cowt <- transformCatCov(saemix.data, covariate=birthrank, newCat=c(1,2,2,3,3), verbose=TRUE)
head(saemix.data@data) # the original covariate is birthrank
head(cowt@data)
# the new covariates are birthrank.ref (initially 3), birthrank.G2 (regrouping 4 and 5) and
# birthrank.G3 (6 and 7)
# only birthrank.G2 and birthrank.G3 are included in the name.covariates slot of the object cowt
cowt <- transformCatCov(cowt, covariate=birthrank, newCat=c(1,2,2,3,3),
   newCatName=c("ref","preg4-5","6-7"), verbose=TRUE)
head(cowt@data)
# new names can be assigned to the dichotomised covariates through the newCatName argument
# the new covariates are now "ref", preg4-5" (regrouping 4 and 5) and "6-7" (6 and 7)

# Changing the reference for a binary variable
cowt<-transformCatCov(cowt, covariate=twin, newCat=c(2,1), newCatName="Singleton", verbose=TRUE)
```

---

transformContCov *Transform continuous covariates*

Description

Transform and/or center continuous covariates

Usage

```
transformContCov(
  object,
  covariate,
```

```
  transformation = function(x) x,
  centering = "median",
  na.rm = TRUE,
  newCovName = "",
  verbose = FALSE
)
```

Arguments

object	saemixData object
covariate	the name of the covariate to transform (without quotes)
transformation	function used to apply the transformation (can be a name, eg log or exp, or a function, see examples). Defaults to no transformation
centering	string, giving the value used to center the covariate; can be "mean" or "median", in which case this value will be computed from the data, 'none' or 0 for no centering, or a value given by the user. Defaults to the median value over the entire dataset (including potential duplicates). The transformation will be applied as transformation(covariate)-transformation(centering value)
na.rm	a logical evaluating to TRUE or FALSE indicating whether NA values should be stripped before computing mean or median for centering (defaults to TRUE)
newCovName	the name of the transformed covariate (if not given, will be set to "the name of the covariate to transform"+".mod")
verbose	a boolean, prints messages during the execution of the function if TRUE. Defaults to FALSE.

Details

Transformations can only involve one covariate. More complex transformations should be performed by the user and integrated in the data object before a call to saemixData NA values are kept unchanged (values NA before will be NA after transformation) Using transformContCov on an saemixData object will create or modify the trans.cov slot, to store the transformation leading to the new covariate. Warning: the name of the covariate must be given without quotes, to match the behaviour of the usual transform function in R.

Value

an object of class "SaemixData"

Examples

```
# Log-transform variable birthyear and center it to the median
  x<-saemixData(name.data=cow.saemix,name.group="cow",name.predictors=c("time"),
      name.response="weight",
  name.covariates=c("birthyear","twin","birthrank"),
      units=list(x="d",y="kg",covariates=c("yr","-","-")),verbose=FALSE)
  x2<-transformContCov(x,birthyear,centering="median",transformation=log,verbose=FALSE,
      newCovName = "logYear")
  print(summary(x2@data$logYear))
```

```
# Should be the same as:
  print(summary(x@data$birthyear-median(x@data$birthyear)))
```

`validate.covariance.model` *Validate the structure of the covariance model*

Description

Check that a matrix corresponds to a structure defining a covariance model for a non-linear mixed effect model. Such a matrix should be composed of only 0s and 1s, with at least one element set to 1, and should be square and symmetrical. 1s on the diagonal indicate that the corresponding parameter has interindividual variability and that its variance will be estimated. 1s as off-diagonal elements indicate that a covariance between the two corresponding parameters will be estimated.

Usage

```
validate.covariance.model(x, verbose = TRUE)
```

Arguments

`x`	a matrix
`verbose`	a boolean indicating whether warnings should be output if x is not a valid covariance model

Value

a boolean, TRUE if x is an acceptable structure and FALSE if not. Messages will be output to describe why x isn't a valid covariance model if the argument verbose is TRUE.

Author(s)

Emmanuelle Comets emmanuelle.comets@inserm.fr, Belhal Karimi

See Also

`SaemixModel`

Examples

```
covarmodel<-diag(c(1,1,0))
validate.covariance.model(covarmodel) # should return TRUE
```

validate.names *Name validation (##)Helper function not intended to be called by the user)*

Description

Helper function, checks if the names given by the user match to the names in the dataset. If not, automatic recognition is attempted when automatic=TRUE.

Arguments

usernames	vector of strings
datanames	vector of strings
recognisednames	vector of strings, values for automatic recognition
verbose	logical, whether to print warning messages
automatic	a boolean indicating whether to attempt automatic name recognition when some colum names are missing or wrong (defaults to TRUE)

Value

a vector with valid names

Examples

```
# TODO
```

vcov *Extracts the Variance-Covariance Matrix for a Fitted Model Object*

Description

Returns the variance-covariance matrix of the main parameters of a fitted model object

Usage

```
## S3 method for class 'SaemixRes'
vcov(object, ...)

## S3 method for class 'SaemixObject'
vcov(object, ...)

## S3 method for class 'SaemixObject'
vcov(object, ...)
```

Arguments

object	a fitted object from a call to saemix
...	further arguments to be passed to or from other methods

Value

A matrix of the estimated covariances between the parameter estimates in model. In saemix, this matrix is obtained as the inverse of the Fisher Information Matrix computed by linearisation

---

xbinning *Internal functions used to produce prediction intervals (from the npde package)*

Description

Functions used by plot functions to define the boundaries of the bins on the X-axis

Usage

```
xbinning(xvec, plot.opt, verbose = FALSE)
```

Arguments

xvec	a vector of values for the X-axis of the plot
plot.opt	graphical options
verbose	boolean (defaults to FALSE). If TRUE, a table showing how the binning was performed

Details

These functions are normally not called by the end-user.

Value

a list with 3 elements, xgrp (the number of the bin associated with each element of xvec), xcent (a named vector containing the mean of the elements of xvec contained in each bin; the name of the bin is the interval), and xgroup (a vector with the group associated to each element of xvec after binning) If verbose is TRUE, a table showing the bins is shown, giving the interval of xvec associated with each bin, the mean value of xvec in each bin, and the number of observations

Author(s)

Emmanuelle Comets emmanuelle.comets@bichat.inserm.fr

References

K. Brendel, E. Comets, C. Laffont, C. Laveille, and F. Mentre. Metrics for external model evaluation with an application to the population pharmacokinetics of gliclazide. *Pharmaceutical Research*, 23:2036–49, 2006.

See Also

npde, autonpde

`yield.saemix` *Wheat yield in crops treated with fertiliser, in SAEM format*

Description

The`yield.saemix` contains data from winter wheat experiments.

Usage

```
yield.saemix
```

Format

This data frame contains the following columns:

site the site number

dose dose of nitrogen fertiliser (kg/ha)

yield grain yield (kg/ha)

soil.nitrogen end-of-winter mineral soil nitrogen (NO3- plus NH4+) in the 0 to 90 cm layer was measured on each site/year (kg/ha)

Details

The data in the `yield.saemix` comes from 37 winter wheat experiments carried out between 1990 and 1996 on commercial farms near Paris, France. Each experiment was from a different site. Two soil types were represented, a loam soil and a chalky soil. Common winter wheat varieties were used. Each experiment consisted of five to eight different nitrogen fertiliser rates, for a total of 224 nitrogen treatments. Nitrogen fertilizer was applied in two applications during the growing season. For each nitrogen treatment, grain yield (adjusted to 150 g.kg-1 grain moisture content) was measured. In addition, end-of-winter mineral soil nitrogen (NO3- plus NH4+) in the 0 to 90 cm layer was measured on each site-year during February when the crops were tillering. Yield and end-of-winter mineral soil nitrogen measurements were in the ranges 3.44-11.54 t.ha-1 , and 40-180 kg.ha-1 respectively.

Source

Makowski, D., Wallach, D., and Meynard, J.-M (1999). Models of yield, grain protein, and residual mineral nitrogen responses to applied nitrogen for winter wheat. Agronomy Journal 91: 377-385.

Examples

```
data(yield.saemix)
saemix.data<-saemixData(name.data=yield.saemix,header=TRUE,name.group=c("site"),
      name.predictors=c("dose"),name.response=c("yield"),
    name.covariates=c("soil.nitrogen"),units=list(x="kg/ha",y="t/ha",covariates=c("kg/ha")))

#  Model: linear + plateau
yield.LP<-function(psi,id,xidep) {
  x<-xidep[,1]
  ymax<-psi[id,1]
  xmax<-psi[id,2]
  slope<-psi[id,3]
  f<-ymax+slope*(x-xmax)
  #'  cat(length(f),"  ",length(ymax),"\n")
  f[x>xmax]<-ymax[x>xmax]
  return(f)
}
saemix.model<-saemixModel(model=yield.LP,description="Linear plus plateau model",
        psi0=matrix(c(8,100,0.2,0,0,0),ncol=3,byrow=TRUE,dimnames=list(NULL,
          c("Ymax","Xmax","slope"))),covariate.model=matrix(c(0,0,0),ncol=3,byrow=TRUE),
        transform.par=c(0,0,0),covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,
            byrow=TRUE),error.model="constant")

saemix.options<-list(algorithms=c(1,1,1),nb.chains=1,seed=666,
       save=FALSE,save.graphs=FALSE,displayProgress=FALSE)

# Plotting the data
plot(saemix.data,xlab="Fertiliser dose (kg/ha)", ylab="Wheat yield (t/ha)")


saemix.fit<-saemix(saemix.model,saemix.data,saemix.options)

# Comparing the likelihoods obtained by linearisation and importance sampling
# to the likelihood obtained by Gaussian Quadrature
saemix.fit<-llgq.saemix(saemix.fit)
{
   cat("LL by Importance sampling, LL_IS=",saemix.fit["results"]["ll.is"],"\n")
   cat("LL by linearisation, LL_lin=",saemix.fit["results"]["ll.lin"],"\n")
   cat("LL by Gaussian Quadrature, LL_GQ=",saemix.fit["results"]["ll.gq"],"\n")
}

# Testing for an effect of covariate soil.nitrogen on Xmax
saemix.model2<-saemixModel(model=yield.LP,description="Linear plus plateau model",
         psi0=matrix(c(8,100,0.2,0,0,0),ncol=3,byrow=TRUE,dimnames=list(NULL,
          c("Ymax","Xmax","slope"))),covariate.model=matrix(c(0,1,0),ncol=3,byrow=TRUE),
         transform.par=c(0,0,0),covariance.model=matrix(c(1,0,0,0,1,0,0,0,1),ncol=3,
             byrow=TRUE),error.model="constant")

saemix.fit2<-saemix(saemix.model2,saemix.data,saemix.options)
# BIC for the two models
{
  cat("Model without covariate, BIC=",saemix.fit["results"]["bic.is"],"\n")
```

```
  cat("Model with covariate, BIC=",saemix.fit2["results"]["bic.is"],"\n")
  pval<-1-pchisq(-2*saemix.fit["results"]["ll.is"]+2*saemix.fit2["results"]["ll.is"],1)
  cat("        LRT: p=",pval,"\n")
}

#' @keywords datasets
```

`[` *Get/set methods for SaemixData object*

Description

Access slots of an SaemixData object using the object["slot"] format

Usage

```
## S4 method for signature 'SaemixData'
x[i, j, drop]

## S4 method for signature 'SaemixRepData'
x[i, j, drop]

## S4 replacement method for signature 'SaemixRepData'
x[i, j] <- value

## S4 method for signature 'SaemixSimData'
x[i, j, drop]

## S4 replacement method for signature 'SaemixSimData'
x[i, j] <- value

## S4 replacement method for signature 'SaemixRes'
x[i, j] <- value
```

Arguments

x	object
i	element to be replaced
j	element to replace with
drop	whether to drop unused dimensions
value	value to replace with

[,SaemixModel-method *Get/set methods for SaemixModel object*

Description

Access slots of an SaemixModel object using the object["slot"] format

Usage

```
## S4 method for signature 'SaemixModel'
x[i, j, drop]
```

Arguments

x	object
i	element to be replaced
j	element to replace with
drop	whether to drop unused dimensions

[,SaemixObject-method *Get/set methods for SaemixObject object*

Description

Access slots of an SaemixObject object using the object["slot"] format

Usage

```
## S4 method for signature 'SaemixObject'
x[i, j, drop]
```

Arguments

x	object
i	element to be replaced
j	element to replace with
drop	whether to drop unused dimensions

`[,SaemixRes-method` *Get/set methods for SaemixRes object*

Description

Access slots of a SaemixRes object using the object["slot"] format

Usage

```
## S4 method for signature 'SaemixRes'
x[i, j, drop]
```

Arguments

x	object
i	element to be replaced
j	element to replace with
drop	whether to drop unused dimensions

Index