Package 'PFIM'

June 26, 2025

Type Package

Title Population Fisher Information Matrix

Version 7.0

Date 2025

NeedsCompilation no

Description Evaluate or optimize designs for nonlinear mixed effects models using the Fisher Information matrix. Methods used in the package refer to
Mentré F, Mallet A, Baccar D (1997) <doi:10.1093/biomet/84.2.429>,
Retout S, Comets E, Samson A, Mentré F (2007) <doi:10.1002/sim.2910>,
Bazzoli C, Retout S, Mentré F (2009) <doi:10.1002/sim.3573>,
Le Nagard H, Chao L, Tenaillon O (2011) <doi:10.1186/1471-2148-11-326>,
Combes FP, Retout S, Frey N, Mentré F (2013) <doi:10.1007/s11095-013-1079-3> and
Seurat J, Tang Y, Mentré F, Nguyen TT (2021) <doi:10.1016/j.cmpb.2021.106126>.

URL http://www.pfim.biostat.fr/, https://github.com/packagePFIM

BugReports https://github.com/packagePFIM/Issues

Materials NEWS.md

Depends R (>= 4.0.0)

License GPL (>=3)

Encoding UTF-8

LazyData true

Imports utils,
Deriv,
methods,
devtools,
deSolve,
purrr,
stringr,
S7,
Matrix,
ggplot2,
Rcpp,
RcppArmadillo,
pracma,
kableExtra,
tibble,
scales,
knitr

Roxygen list(markdown = TRUE, roclets = c(``rd'', ``namespace''))

RoxygenNote 7.3.2

VignetteBuilder knitr

Collate 'Administration.R'
'AdministrationConstraints.R'
'Model.R'
'Fim.R'
'PFIMProject.R'
'Optimization.R'
'PGBOAlgorithm.R'
'PSOAlgorithm.R'
'SimplexAlgorithm.R'
'FedorovWynnAlgorithm.R'
'MultiplicativeAlgorithm.R'
'Arm.R'
'BayesianFim.R'
'ModelError.R'
'Combined.R'
'Combined1.R'
'Constant.R'
'Design.R'
'Distribution.R'
'Evaluation.R'
'IndividualFim.R'
'LibraryOfModels.R'
'LibraryOfPDModels.R'
'LibraryOfPKModels.R'
'LogNormal.R'
'ModelODE.R'
'ModelAnalytic.R'
'ModelInfusion.R'
'ModelAnalyticInfusion.R'
'ModelAnalyticInfusionSteadyState.R'
'ModelAnalyticSteadyState.R'
'ModelODEBolus.R'
'ModelODEDoseInEquations.R'
'ModelODEDoseNotInEquations.R'
'ModelODEInfusion.R'
'ModelODEInfusionDoseInEquation.R'
'ModelParameter.R'
'Normal.R'
'PFIM-package.R'
'PopulationFim.R'
'Proportional.R'
'SamplingTimeConstraints.R'
'SamplingTimes.R'
'zzz.R'

Suggests rmarkdown,
testthat (>= 3.0.0)

Config/testthat/edition 3

Contents

`PFIM-package` *Fisher Information matrix for design evaluation/optimization for non-linear mixed effects models.*

Description

Evaluate or optimize designs for nonlinear mixed effects models using the Fisher Information matrix. Methods used in the package refer to Mentré F, Mallet A, Baccar D (1997) doi:10.1093/biomet/84.2.429, Retout S, Comets E, Samson A, Mentré F (2007) doi:10.1002/sim.2910, Bazzoli C, Retout S, Mentré F (2009) doi:10.1002/sim.3573, Le Nagard H, Chao L, Tenaillon O (2011) doi:10.1186/1471214811326, Combes FP, Retout S, Frey N, Mentré F (2013) doi:10.1007/s11095-01310793 and Seurat J, Tang Y, Mentré F, Nguyen TT (2021) doi:10.1016/j.cmpb.2021.106126.

Description

Nonlinear mixed effects models (NLMEM) are widely used in model-based drug development and use to analyze longitudinal data. The use of the "population" Fisher Information Matrix (FIM) is a good alternative to clinical trial simulation to optimize the design of these studies. The present version, **PFIM 7.0**, is an R package that uses the S4 object system for evaluating and/or optimizing population designs based on FIM in NLMEMs.

This version of **PFIM** now includes a library of models implemented also using the object oriented system S4 of R. This library contains two libraries of pharmacokinetic (PK) and/or pharmacodynamic (PD) models. The PK library includes model with different administration routes (bolus, infusion, first-order absorption), different number of compartments (from 1 to 3), and different types of eliminations (linear or Michaelis-Menten). The PD model library, contains direct immediate models (e.g. Emax and Imax) with various baseline models, and turnover response models. The PK/PD models are obtained with combination of the models from the PK and PD model libraries. **PFIM** handles both analytical and ODE models and offers the possibility to the user to define his/her own model(s). In **PFIM 7.0**, the FIM is evaluated by first order linearization of the model assuming a block diagonal FIM as in Mentré et al. (1997). The Bayesian FIM is also available to give shrinkage predictions (Combes et al., 2013). **PFIM 7.0** includes several algorithms to conduct design optimization based on the D-criterion, given design constraints: the simplex algorithm (Nelder-Mead) (Nelder & Mead, 1965), the multiplicative algorithm (Seurat et al., 2021), the Fedorov-Wynn algorithm (Fedorov, 1972), PSO (*Particle Swarm Optimization*) and PGBO (*Population Genetics Based Optimizer*) (Le Nagard et al., 2011).

Documentation

Documentation and user guide are available at `http://www.pfim.biostat.fr/`

Validation

PFIM 7.0 also provides quality control with tests and validation using the evaluated FIM to assess the validity of the new version and its new features. Finally, **PFIM 7.0** displays all the results with both clear graphical form and a data summary, while ensuring their easy manipulation in R. The standard data visualization package ggplot2 for R is used to display all the results with clear graphical form (Wickham, 2016). A quality control using the D-criterion is also provided.

Organization of the source files in the `/R` folder

PFIM 7.0 contains a hierarchy of S4 classes with corresponding methods and functions serving as constructors. All of the source code related to the specification of a certain class is contained in a file named `[Name_of_the_class]-Class.R`. These classes include:

1. all roxygen `@include` to insure the correctly generated collate for the DESCRIPTION file,
2. `\setClass` preceded by a roxygen documentation that describes the purpose and slots of the class,
3. specification of an initialize method,
4. all getter and setter, respectively returning attributes of the object and associated objects.

Author(s)

Maintainer: Romain Leroux `<romainlerouxPFIM@gmail.com>`

Authors:

- France Mentré `<france.mentre@inserm.fr>`

Other contributors:

- Jérémy Seurat `<jeremy.seurat@inserm.fr>` [contributor]

References

Dumont C, Lestini G, Le Nagard H, Mentré F, Comets E, Nguyen TT, et al. PFIM 4.0, an extended R program for design evaluation and optimization in nonlinear mixed-effect models. Comput Methods Programs Biomed. 2018;156:217-29.

Chambers JM. Object-Oriented Programming, Functional Programming and R. Stat Sci. 2014;29:167-80.

Mentré F, Mallet A, Baccar D. Optimal Design in Random-Effects Regression Models. Biometrika. 1997;84:429-42.

Combes FP, Retout S, Frey N, Mentré F. Prediction of shrinkage of individual parameters using the Bayesian information matrix in nonlinear mixed effect models with evaluation in pharmacokinetics. Pharm Res. 2013;30:2355-67.

Nelder JA, Mead R. A simplex method for function minimization. Comput J. 1965;7:308-13.

Seurat J, Tang Y, Mentré F, Nguyen, TT. Finding optimal design in nonlinear mixed effect models using multiplicative algorithms. Computer Methods and Programs in Biomedicine, 2021.

Fedorov VV. Theory of Optimal Experiments. Academic Press, New York, 1972.

Eberhart RC, Kennedy J. A new optimizer using particle swarm theory. Proc. of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, 4-6 October 1995, 39-43.

Le Nagard H, Chao L, Tenaillon O. The emergence of complexity and restricted pleiotropy in adapting networks. BMC Evol Biol. 2011;11:326.

Wickham H. ggplot2: Elegant Graphics for Data Analysis, Springer-Verlag New York, 2016.

See Also

Useful links:

- http://www.pfim.biostat.fr/
- https://github.com/packagePFIM
- Report bugs at https://github.com/packagePFIM/Issues

`adjustGradient` *adjustGradient: adjust the gradient for the log normal distribution.*

Description

adjustGradient: adjust the gradient for the log normal distribution.

Arguments

`distribution`	An object `Distribution` giving the distribution.
`gradient`	The gradient of the model responses.

Value

The adjusted gradient of the model responses.

Administration *Administration*

Description

The class Administration represents the administration and stores information concerning the administration for the dosage regimen.

Usage

```
Administration(
  outcome = character(0),
  timeDose = numeric(0),
  dose = numeric(0),
  Tinf = numeric(0),
  tau = 0
)
```

Arguments

outcome	A string giving the outcome for the administration.
timeDose	A vector of double giving the time doses.
dose	A vector of double giving the doses.
Tinf	A vector of double giving the time for infusion Tinf.
tau	An integer giving the tau value for repeated dose or steady state.

Details

Class Administration

AdministrationConstraints *AdministrationConstraints*

Description

The class AdministrationConstraints represents the constraint of an input to the system. The class stores information concerning the constraints for the dosage regimen.

Usage

```
AdministrationConstraints(outcome = character(0), doses = list())
```

Arguments

outcome	A string giving the outcome for the administration.
doses	A vector of double giving the doses.

Details

Class AdministrationConstraints

Arm *Arm*

Description

The class Arm represents an arm and stores information concerning an arm.

Usage

```
Arm(
  name = character(0),
  size = numeric(0),
  administrations = list(),
  initialConditions = list(),
  samplingTimes = list(),
  administrationsConstraints = list(),
  samplingTimesConstraints = list(),
  evaluationModel = list(),
  evaluationGradients = list(),
  evaluationVariance = list(),
  evaluationFim = Fim()
)
```

Arguments

name	A string giving the name of the arm.
size	A integer giving the size of the arm.
administrations	A list giving the objects of class Administration that define the administrations of the arm.
initialConditions	A list giving the initial conditions for the ode model where the names are string that define the variable and their value are giving by double.
samplingTimes	A list giving the objects of class SamplingTime that define the sampling time of the arm.
administrationsConstraints	A list giving the objects of class AdministrationsConstraints that define the administration constraints of the arm.
samplingTimesConstraints	A list giving the objects of class SamplingTimeConstraints that define the sampling time constraints of the arm.
evaluationModel	A list giving the evaluation of the responses of the arm.
evaluationGradients	A list giving the evaluation of the responses gradient of the arm.
evaluationVariance	A list giving the evaluation of the variance.
evaluationFim	A object of class Fim giving the Fisher Information Matrix.

Details

Class Arm

`armAdministration` *getArmAdministration: get the administration parameters of an arm.*

Description

getArmAdministration: get the administration parameters of an arm.

Arguments

`arm`	A object of class `Arm` giving the arm.

Value

A list giving the administration parameters of an arm.

`BayesianFim` *BayesianFim*

Description

The class `BayesianFim` represents and stores information for the Bayesian Fim.

Usage

```
BayesianFim(
  fisherMatrix = numeric(0),
  fixedEffects = numeric(0),
  varianceEffects = numeric(0),
  SEAndRSE = list(),
  condNumberFixedEffects = numeric(0),
  condNumberVarianceEffects = numeric(0),
  shrinkage = numeric(0)
)
```

Arguments

`fisherMatrix`	A matrix giving the numerical values of the Fim.
`fixedEffects`	A matrix giving the numerical values of the fixedEffects of the Fim.
`varianceEffects`	A matrix giving the numerical values of varianceEffects of the Fim.
`SEAndRSE`	A data frame giving the value of the SE and RSE.
`condNumberFixedEffects`	The conditional number of the fixedEffects of the Fim.
`condNumberVarianceEffects`	The conditional number of the varianceEffects of the Fim.
`shrinkage`	A vector giving the shrinkage values.

Details

BayesianFim

`checkSamplingTimeConstraintsForMetaheuristic`
checkSamplingTimeConstraintsForMetaheuristic

Description

checkSamplingTimeConstraintsForMetaheuristic

Arguments

`samplingTimesConstraints`	An object `SamplingTimeConstraints`.
`arm`	An object `Arm`.
`newSamplings`	A vector of numeric for the new samplings.
`outcome`	A string giving the outcome.

Value

A boolean TRUE/FALSE, with a message error if FALSE.

`checkValiditySamplingConstraint`
checkValiditySamplingConstraint: check if the constraints used for the design optimization are valid.

Description

checkValiditySamplingConstraint: check if the constraints used for the design optimization are valid.

Arguments

`design`	An object `Design` giving the design.

Value

A boolean TRUE / FALSE, if FALSE it also gives an error message.

Combined1 *Combined1*

Description

The class Combined1 represents and stores information for the error model Combined1.

The class Combined1 represents and stores information for the error model Combined1.

Usage

```
Combined1(
  output = "output",
  equation = expression(sigmaInter + sigmaSlope * output),
  derivatives = list(),
  sigmaInter = NULL,
  sigmaSlope = NULL,
  sigmaInterFixed = FALSE,
  sigmaSlopeFixed = FALSE,
  cError = NULL
)

Combined1(
  output = "output",
  equation = expression(sigmaInter + sigmaSlope * output),
  derivatives = list(),
  sigmaInter = NULL,
  sigmaSlope = NULL,
  sigmaInterFixed = FALSE,
  sigmaSlopeFixed = FALSE,
  cError = NULL
)
```

Arguments

output	A string giving the model error output.
equation	A expression giving the model error equation.
derivatives	A list giving the derivatives of the model error equation.
sigmaInter	A double giving the sigma inter.
sigmaSlope	A double giving the sigma slope
sigmaInterFixed	A boolean giving if the sigma inter is fixed or not. - not in the v7.0
sigmaSlopeFixed	A boolean giving if the sigma slope is fixed or not. - not in the v7.0
cError	A integer giving the power parameter.

Details

Combined1

Combined1

`computeVMat` *computeVMat*

Description

computeVMat

Usage

```
computeVMat(varParam1, varParam2, invCholV)
```

Arguments

`varParam1`	varParam1
`varParam2`	varParam2
`invCholV`	invCholV

Value

VMat

`Constant` *Constant*

Description

The class `Constant` represents and stores information for the error model Constant.

Usage

```
Constant(
  output = "output",
  equation = expression(sigmaInter),
  derivatives = list(),
  sigmaInter = NULL,
  sigmaSlope = NULL,
  sigmaInterFixed = FALSE,
  sigmaSlopeFixed = FALSE,
  cError = NULL
)
```

Arguments

`output`	A string giving the model error output.
`equation`	A expression giving the model error equation.
`derivatives`	A list giving the derivatives of the model error equation.
`sigmaInter`	A double giving the sigma inter.
`sigmaSlope`	A double giving the sigma slope

`sigmaInterFixed`
: A boolean giving if the sigma inter is fixed or not.

`sigmaSlopeFixed`
: A boolean giving if the sigma slope is fixed or not.

`cError`
: A integer giving the power parameter.

Details

Constant

`constraintsTableForReport`
: *constraintsTableForReport: table of the PGBOAlgorithm constraints for the report.*

Description

constraintsTableForReport: table of the PGBOAlgorithm constraints for the report.

constraintsTableForReport: table of the PSOAlgorithm constraints for the report.

constraintsTableForReport: table of the SimplexAlgorithm constraints for the report.

constraintsTableForReport

constraintsTableForReport: table of the MultiplicativeAlgorithm constraints for the report.

Arguments

`optimizationAlgorithm`
: A object `MultiplicativeAlgorithm`.

`arms`
: List of the arms.

Value

The table for the constraints in the arms.

The table for the constraints in the arms.

The table for the constraints in the arms.

armsConstraintsTable

The table for the constraints in the arms.

convertPKModelAnalyticToPKModelODE

convertPKModelAnalyticToPKModelODE: conversion from analytic to ode

Description

convertPKModelAnalyticToPKModelODE: conversion from analytic to ode

convertPKModelAnalyticToPKModelODE: conversion from analytic to ode

convertPKModelAnalyticToPKModelODE: conversion from analytic infusion to ode

Arguments

pkModel	An object of class ModelAnalyticInfusion that defines the model.

Dcriterion

Dcriterion: get the D-criterion of the Fim.

Description

Dcriterion: get the D-criterion of the Fim.

Arguments

Fim	A object Fim giving the Fim.

Value

A double giving the D-criterion of the Fim.

defineFim

define the type of Fisher information matrix: population, individual or Bayesian

Description

define the type of Fisher information matrix: population, individual or Bayesian

Arguments

pfimproject	An object PFIMProject.

Value

An object Fim.

`defineModelAdministration` *defineModelAdministration: define the administration*

Description

defineModelAdministration: define the administration

defineModelAdministration: define the administration

defineModelAdministration: define the administration

defineModelAdministration: define the administration

defineModelAdministration: define the administration

defineModelAdministration: define the administration

defineModelAdministration: define the administration

Arguments

`model`	An object of class `ModelODEInfusionDoseInEquation` that defines the model.
`arm`	An object of class `Arm` that defines the arm.

Value

The model with samplings, solverInputs

The model with samplings, solverInputs

The model with samplings, solverInputs

The model with updated slots.

The model with samplings, solverInputs

The model with samplings, solverInputs

The model with updated slots.

`defineModelEquationsFromLibraryOfModel` *defineModelEquationsFromLibraryOfModel: define the model equations giving the models in the library of models.*

Description

defineModelEquationsFromLibraryOfModel: define the model equations giving the models in the library of models.

Arguments

`pfimproject`	An object `PFIMProject` giving the evaluation to be run.

Value

A list giving the model equations.

`defineModelType` *defineModelType: define the class of the model to be evaluated.*

Description

defineModelType: define the class of the model to be evaluated.

Arguments

`pfimproject`	An object `PFIMProject` giving the evaluation to be run.

Value

An object `Model` giving the model to be evaluated with its modelParameters, odeSolverParameters, modelError, modelEquations.

`defineModelWrapper` *defineModelWrapper: define the model wrapper for the ode solver*

Description

defineModelWrapper: define the model wrapper for the ode solver

defineModelWrapper: define the model wrapper for the ode solver

defineModelWrapper: define the model wrapper for the ode solver

defineModelWrapper: define the model wrapper for the ode solver

defineModelWrapper: define the model wrapper for the ode solver

defineModelWrapper: define the model wrapper for the ode solver

defineModelWrapper: define the model wrapper for the ode solver

defineModelWrapper: define the model wrapper for the ode solver

Arguments

`model`	An object of class `ModelODEInfusionDoseInEquation` that defines the model.
`evaluation`	An object of class Evaluation that defines the evaluation

Value

The model with wrapperModelAnalytic, functionArgumentsModelAnalytic, functionArgumentsSymbolModelAnalytic, outputNames, outcomesWithAdministration

The model with wrapperModelAnalyticInfusion, functionArgumentsModelAnalyticInfusion, functionArgumentsSymbolModelAnalyticInfusion, outputNames, outcomesWithAdministration

The model with wrapperModelAnalyticInfusion, functionArgumentsModelAnalyticInfusion, functionArgumentsSymbolModelAnalyticInfusion, outputNames, outcomesWithAdministration

The model with wrapperModelAnalytic, functionArgumentsModelAnalytic, functionArgumentsSymbolModelAnalytic, outputNames, outcomesWithAdministration

The model with updated slots.

The model with the updated slots.

The model with the updated slots.

The model with updated slots.

`defineOptimizationAlgorithm` *Define optimization algorithm*

Description

Define optimization algorithm

Arguments

`optimization`	An `Optimization` object.

Value

An optimization algorithm.

`definePKModel` *definePKModel: define a PK model from library of model*

Description

definePKModel: define a PK model from library of model

definePKModel ModelAnalyticInfusion

definePKModel

definePKModel

definePKModel: define PK model ode bolus

definePKModel: define a PK model from library of model

definePKModel: define a PK model from library of model

definePKModel: define PK model ode bolus

Arguments

`pkModel`	An object of class `ModelODEInfusionDoseInEquation` that defines the PK model.
`pfimproject`	An object of class `PFIMProject` that defines the pfimproject.

`definePKPDModel` *definePKPDModel: define a PKPD model from library of model*

Description

definePKPDModel: define a PKPD model from library of model

definePKPDModel: define a PKPD model from library of model

definePKPDModel ModelAnalyticInfusion, ModelAnalytic

definePKPDModel ModelAnalyticInfusion, ModelODE

definePKPDModel

definePKPDModel

definePKPDModel

definePKPDModel: define a PKPD model from library of model

Arguments

`pkModel`	An object of class `ModelODE` that defines the PD model.
`pfimproject`	An object of class `PFIMProject` that defines the pfimproject.

`Design` *Design*

Description

The class `Design` represents and stores information for the Design.

Usage

```
Design(
  name = character(0),
  size = numeric(0),
  arms = list(),
  evaluationArms = list(),
  numberOfArms = numeric(0),
  fim = Fim()
)
```

Arguments

`name`	A string giving the name of the design.
`size`	A integer gving the size of the design.
`arms`	A list giving the arms of the design.
`evaluationArms`	A list giving the valuation of the arms of the design.
`numberOfArms`	A integer giving the number of arms.
`fim`	A object `Fim` giving the Fim of the design.

Details

Design

`Distribution` *Distribution*

Description

The class `Distribution` represents and stores information for the parameter distribution.

Usage

```
Distribution(name = character(0), mu = numeric(0), omega = numeric(0))
```

Arguments

`name`	A string giving the name of the distribution.
`mu`	A double giving the mean mu.
`omega`	A double giving omega.

Details

Distribution

`evaluateArm` *evaluateArm: evaluation of the model with the arm parameters.*

Description

evaluateArm: evaluation of the model with the arm parameters.

Arguments

`arm`	A object of class `Arm` giving the arm.
`model`	A object of class `Model` giving the model.
`fim`	A object of class `Fim` giving the fim.

Value

The object arm with the slots evaluationModel, evaluationGradients, evaluationVariance and evaluationFim.

`evaluateDesign` *evaluateDesign: evaluation of a design.*

Description

evaluateDesign: evaluation of a design.

Arguments

`design`	An object `Design` giving the design.
`model`	An object `Model` giving the model.
`fim`	An object `Fim` giving the Fim.

Value

The object `Design` with its evaluation results.

`evaluateErrorModelDerivatives` *evaluateErrorModelDerivatives; evaluate the derivatives of the model error.*

Description

evaluateErrorModelDerivatives; evaluate the derivatives of the model error.

Arguments

`modelError`	An object `ModelError` that defines the model error.
`evaluationModel`	A dataframe giving the outputs for the model evaluation.

Value

The matrices sigmaDerivatives and errorVariance.

`evaluateFim` *evaluateFim: evaluation of the Fim*

Description

evaluateFim: evaluation of the Fim

evaluateFim: evaluation of the Fim

evaluateFim: evaluation of the Fim

Arguments

`fim`	An object `PopulationFim` giving the Fim.
`model`	An object `Model` giving the model.
`arm`	An object `Arm` giving the arm.

Value

The object `Fim` with the fisherMatrix and the shrinkage.

The object `IndividualFim` with the fisherMatrix and the shrinkage.

The object `IndividualFim` with the fisherMatrix and the shrinkage.

`evaluateInitialConditions`
evaluateInitialConditions: evaluate the initial conditions.

Description

evaluateInitialConditions: evaluate the initial conditions.

evaluateInitialConditions: evaluate the initial conditions.

evaluateInitialConditions: evaluate the initial conditions.

Arguments

`arm`	A object of class `Arm` giving the arm.
`model`	A object of class `ModelODEInfusion` giving the model.
`doseEvent`	A data frame giving the dose event for the ode solver.

Value

A list giving the evaluated initial conditions.

`evaluateModel` *evaluateModel: evaluate the model*

Description

evaluateModel: evaluate the model

evaluateModel: evaluate the ModelAnalyticInfusion

evaluateModel: evaluate the ModelAnalyticInfusion

evaluateModel: evaluate the ModelAnalyticInfusion

evaluateModel

evaluateModel

evaluateModel: evaluate the model

evaluateModel: evaluate the model

evaluateModel

Arguments

`arm`	A object of class `Arm` giving the arm.
`model`	A object of class `ModelODEInfusionDoseInEquation` giving the model.

Value

A list of dataframes that contains the results for the evaluation of the model.

A list of dataframes that contains the results for the evaluation of the model.

A list of dataframes that contains the results for the evaluation of the model.

A list of dataframes that contains the results for the evaluation of the model.

A list of dataframes that contains the evaluation of the model.

A data frame giving the output of the model evaluation.

A list of dataframes that contains the results for the evaluation of the model.

A list of dataframes that contains the results for the evaluation of the model.

A data frame giving the output of the model evaluation.

`evaluateModelGradient` *evaluateModelGradient: evaluate the gradient of the model*

Description

evaluateModelGradient: evaluate the gradient of the model

Arguments

`model`	An object `Model` that defines the model.
`arm`	A object `Arm` giving the arm

Value

A data frame that contains the gradient of the model.

evaluateModelVariance *evaluateModelVariance: evaluate the variance of the model*

Description

evaluateModelVariance: evaluate the variance of the model

Arguments

`model`	A object `Model` giving the model.
`arm`	A object `Arm` giving the arm

Value

A list giving errorVariance and sigmaDerivatives.

evaluateVarianceFIM *evaluateVarianceFIM: evaluate the variance*

Description

evaluateVarianceFIM: evaluate the variance

evaluateVarianceFIM: evaluate the variance

evaluateVarianceFIM: evaluate the variance

Arguments

`arm`	A object of class `Arm` giving the arm.
`model`	A object of class `Model` giving the model.
`fim`	A object of class `PopulationFim` giving the Fim.

Value

The matrices MFbeta and V.

The matrices MFbeta and V.

The matrices MFVar and V.

Evaluation *Evaluation*

Description

The class Evaluation represents and stores information for the evaluation of a design

Usage

```
Evaluation(
  name = character(0),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelParameters = list(),
  modelError = list(),
  optimizer = character(0),
  optimizerParameters = list(),
  outputs = list(),
  designs = list(),
  fimType = character(0),
  fim = Fim(),
  odeSolverParameters = list(),
  evaluationDesign = list()
)
```

Arguments

name	A string giving the name of the design evaluation.
modelEquations	A list giving the model equations.
modelFromLibrary	A list giving the model equations from the library of model.
modelParameters	A list giving the model parameters.
modelError	A list giving the model error.
optimizer	A string giving the name of the optimization algorithm being used.
optimizerParameters	A list giving the parameters of the optimization algorithm.
outputs	A list giving the model outputs.
designs	A list giving the designs to be evaluated.
fimType	A string giving the type of Fim being evaluated.
fim	A object Fim giving the Fim.
odeSolverParameters	A list giving the atol and rtol parameters for the ode solver.
evaluationDesign	A list giving the evaluation of the design.

Details

Evaluation

FedorovWynnAlgorithm *FedorovWynnAlgorithm*

Description

The class FedorovWynnAlgorithm implements the FedorovWynn algorithm.

Usage

```
FedorovWynnAlgorithm(
  name = character(0),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelParameters = list(),
  modelError = list(),
  optimizer = character(0),
  optimizerParameters = list(),
  outputs = list(),
  designs = list(),
  fimType = character(0),
  fim = Fim(),
  odeSolverParameters = list(),
  optimisationDesign = list(),
  optimisationAlgorithmOutputs = list(),
  elementaryProtocols = list(),
  numberOfSubjects = logical(0),
  proportionsOfSubjects = logical(0),
  showProcess = logical(0),
  FedorovWynnAlgorithmOutputs = list()
)
```

Arguments

name	A string giving the name.
modelEquations	A list giving the model equations.
modelFromLibrary	A list giving the model equations from the library of model.
modelParameters	A list giving the model parameters.
modelError	A list giving the model error.
optimizer	A string giving the name of the optimization algorithm being used.
optimizerParameters	A list giving the parameters of the optimization algorithm.
outputs	A list giving the model outputs.
designs	A list giving the designs to be evaluated.
fimType	A string giving the type of Fim being evaluated.
fim	A object Fim giving the Fim.

`odeSolverParameters`
: A list giving the atol and rtol parameters for the ode solver.

`optimisationDesign`
: A list giving the evaluation of initial and optimal design.

`optimisationAlgorithmOutputs`
: A list giving the outputs of the optimization process.

`elementaryProtocols`
: A lit giving the elementaryProtocols.

`numberOfSubjects`
: A numeric giving the numberOfSubjects.

`proportionsOfSubjects`
: A numeric giving the proportionsOfSubjects.

`showProcess`
: A boolean giving showProcess.

`FedorovWynnAlgorithmOutputs`
: A list giving the output of the optimization algorithm.

Details

Class "FedorovWynnAlgorithm"

FedorovWynnAlgorithm_Rcpp *Fedorov-Wynn algorithm in Rcpp.*

Description

Run the FedorovWynnAlgorithm in Rcpp

Usage

```
FedorovWynnAlgorithm_Rcpp(
  protocols_input,
  ndimen_input,
  nbprot_input,
  numprot_input,
  freq_input,
  nbdata_input,
  vectps_input,
  fisher_input,
  nok_input,
  protdep_input,
  freqdep_input
)
```

Arguments

`protocols_input`
: parameter protocols_input

`ndimen_input`
: parameter ndimen_input

nbprot_input	parameter nbprot_input
numprot_input	parameter numprot_input
freq_input	parameter freq_input
nbdata_input	parameter nbdata_input
vectps_input	parameter vectps_input
fisher_input	parameter fisher_input
nok_input	parameter nok_input
protdep_input	parameter protdep_input
freqdep_input	parameter freqdep_input

Value

A list giving the results of the outputs of the FedorovWynn algorithm.

`Fim` *Fim*

Description

The class `Fim` represents and stores information for the Fim.

Usage

```
Fim(
  fisherMatrix = numeric(0),
  fixedEffects = numeric(0),
  varianceEffects = numeric(0),
  SEAndRSE = list(),
  condNumberFixedEffects = numeric(0),
  condNumberVarianceEffects = numeric(0),
  shrinkage = numeric(0)
)
```

Arguments

fisherMatrix	A matrix giving the numerical values of the Fim.
fixedEffects	A matrix giving the numerical values of the fixedEffects of the Fim.
varianceEffects	A matrix giving the numerical values of varianceEffects of the Fim.
SEAndRSE	A data frame giving the value of the SE and RSE.
condNumberFixedEffects	The conditional number of the fixedEffects of the Fim.
condNumberVarianceEffects	The conditional number of the varianceEffects of the Fim.
shrinkage	A vector giving the shrinkage values.

Details

Fim

finiteDifferenceHessian *finiteDifferenceHessian: compute the Hessian*

Description

finiteDifferenceHessian: compute the Hessian

Arguments

model	A object Model giving the model.

Value

The model with the slots parametersForComputingGradient with XcolsInv, shifted, frac.

fisherSimplex *Compute the fisher.simplex*

Description

Compute the fisher.simplex

Arguments

simplex	A list giving the parameters of the simplex.
optimizationObject	An object Optimization.
outcomes	A vector giving the outcomes of the arms.

Value

A list giving the results of the optimization.

fun.amoeba *Compute the fun.amoeba*

Description

Compute the fun.amoeba

Usage

```
fun.amoeba(p, y, ftol, itmax, funk, outcomes, data, showProcess)
```

Arguments

p	parameter p
y	parameter y
ftol	parameter ftol
itmax	parameter itmax
funk	parameter funk
outcomes	The model outcomes.
data	parameter data
showProcess	Boolean.

Value

fun.amoeba

---

generateDosesCombination

generateDosesCombination: generate the combination for the doses.

Description

generateDosesCombination: generate the combination for the doses.

Arguments

design	An object Design giving the design.

Value

dosesForFIMs, numberOfDoses used in the design optimization.

---

generateFimsFromConstraints

Generate FIMs from constraints

Description

Generate FIMs from constraints

Arguments

optimization	An Optimization object.

Value

A list containing FIMs from constraints.

`generateReportEvaluation` *generateReportEvaluation: generate the report for the model evaluation.*

Description

generateReportEvaluation: generate the report for the model evaluation.

generateReportEvaluation: generate the report for the model evaluation.

generateReportEvaluation: generate the report for the model evaluation.

Arguments

`fim`	An object `PopulationFim` giving the Fim.
`tablesForReport`	The output list giving by the method tablesForReport.

Value

The html report for the design evaluation.

The html report for the model evaluation.

The html report for the model evaluation.

`generateReportOptimization` *generateReportOptimization: generate the report for the design optimization.*

Description

generateReportOptimization: generate the report for the design optimization.

generateReportOptimization: generate the report for the design optimization.

generateReportOptimization: generate the report for the design optimization.

generateReportOptimization: generate the report for the design optimization.

generateReportOptimization: generate the report for the design optimization.

generateReportOptimization: generate the report for the design optimization.

generateReportOptimization: generate the report for the design optimization.

generateReportOptimization: generate the report for the design optimization.

generateReportOptimization: generate the report for the design optimization.

generateReportOptimization: generate the report for the design optimization.

generateReportOptimization: generate the report for the design optimization.

generateReportOptimization: generate the report for the design optimization.

Arguments

`fim`	An object `PopulationFim` giving the Fim.
`optimizationAlgorithm`	An object `PGBOAlgorithm` giving the PGBOAlgorithm
`tablesForReport`	The output list giving by the method tablesForReport.

Value

The html report for the design optimization.

The html report for the design optimization.

The html report.

The html report.

The html report.

The html report.

The html report.

The html report.

The html report.

The html report.

The html report.

The html report.

`generateSamplingsFromSamplingConstraints` *generateSamplingsFromSamplingConstraints*

Description

generateSamplingsFromSamplingConstraints

Arguments

`samplingTimeConstraints`	An object `SamplingTimeConstraints`

Value

A list intervalsConstraints.

generateSamplingTimesCombination

generateSamplingTimesCombination: generate the combination for the samplings.

Description

generateSamplingTimesCombination: generate the combination for the samplings.

Arguments

design	An object Design giving the design.

Value

samplingTimesCombinations used in the design optimization.

getArmConstraints

getArmConstraints: get the administration and sampling time constraints for the MultiplicativeAlgorithm.

Description

getArmConstraints: get the administration and sampling time constraints for the MultiplicativeAlgorithm.

getArmConstraints: get the administration and sampling time constraints for the FedorovWynnAlgorithm.

getArmConstraints: get the administration and sampling time constraints for the SimplexAlgorithm.

getArmConstraints: get the administration and sampling time constraints for the PSOAlgorithm.

getArmConstraints: get the administration and sampling time constraints for the PGBOAlgorithm.

Arguments

arm	A object of class Arm giving the arm.
optimizationAlgorithm	A object of class Optimization giving the optimization algorithm.

Value

A list giving the administration and sampling time constraints for the MultiplicativeAlgorithm.

A list giving the administration and sampling time constraints for the FedorovWynnAlgorithm.

A list giving the administration and sampling time constraints for the SimplexAlgorithm.

A list giving the administration and sampling time constraints for the PSOAlgorithm.

A list giving the administration and sampling time constraints for the PGBOAlgorithm.

getArmData *getArmData: extract arm data for The Report*

Description

getArmData: extract arm data for The Report

Arguments

arm	A object of class Arm giving the arm.

Value

A list giving the name, Number of subjects, Outcome, Dose and Sampling times of the arm.

getCorrelationMatrix *getCorrelationMatrix : get the correlation matrix*

Description

getCorrelationMatrix : get the correlation matrix

getCorrelationMatrix : get the correlation matrix

Arguments

pfimproject	A object PFIMProject giving the Evaluation.

Value

The correlation matrix

The Dcriterion

getDcriterion *getDcriterion : get the Dcriterion*

Description

getDcriterion : get the Dcriterion

getDcriterion : get the Dcriterion

Arguments

pfimproject	A object PFIMProject giving the Evaluation.

Value

The Dcriterion of the FIM.

The Dcriterion

`getDeterminant` *getDeterminant: get the determinant*

Description

getDeterminant: get the determinant

getDeterminant: get the determinant

Arguments

`pfimproject`	A object `PFIMProject` giving the Evaluation.

Value

The determinant of the FIM.

The determinant

`getFim` *getFim: get the Fisher matrix.*

Description

getFim: get the Fisher matrix.

Arguments

`evaluation`	An object `Evaluation` giving the evaluation to be run.

Value

The matrices fisherMatrix, fixedEffects, varianceEffects.

`getFisherMatrix` *getFisherMatrix: display the Fisher matrix components*

Description

getFisherMatrix: display the Fisher matrix components

getFisherMatrix: display the Fisher matrix components

Arguments

`evaluation`	An object `Evaluation` giving the evaluation to be run.

Value

The matrices fisherMatrix, fixedEffects, varianceEffects.

The matrices fisherMatrix, fixedEffects, varianceEffects.

`getListLastName` *getListLastName: routine to get the names of last element of a nested list.*

Description

getListLastName: routine to get the names of last element of a nested list.

Usage

```
getListLastName(list)
```

Arguments

`list`	The list to be used.

Value

The names of last element.

`getModelErrorData` *getModelErrorData: get the parameters sigma slope and sigma inter (used for the report).*

Description

getModelErrorData: get the parameters sigma slope and sigma inter (used for the report).

Arguments

`modelError`	An object `ModelError` that defines the model error.

Value

A list of dataframe with outcome, type of model error and sigma slope and inter.

`getModelParametersData` *getModelParametersData: get model parameters data for report.*

Description

getModelParametersData: get model parameters data for report.

Arguments

`modelParameter`	An object if class `Model` giving the model.

Value

A data frame with the data of all the parameters.

getRSE — *getRSE: get the RSE*

Description

getRSE: get the RSE

getRSE: get the RSE

Arguments

pfimproject	A object PFIMProject giving the Evaluation.

Value

The RSE of the parameters.

The RSE

getSamplingData — *getSamplingData: extract sampling times and max sampling time used for plot.*

Description

getSamplingData: extract sampling times and max sampling time used for plot.

Arguments

arm	A object of class Arm giving the arm.

Value

A list giving the samplingTimes object, the vector samplings and the double samplingMax.

getSE — *getSE: get the SE*

Description

getSE: get the SE

getSE: get the SE

Arguments

pfimproject	A object PFIMProject giving the Evaluation.

Value

The SE of the parameters.

The SE.

getShrinkage *getShrinkage: get the shrinkage*

Description

getShrinkage: get the shrinkage

getShrinkage: get the shrinkage

Arguments

pfimproject	A object PFIMProject giving the Evaluation.

Value

The shrinkage of the FIM.

The shrinkage

IndividualFim *IndividualFim*

Description

The class IndividualFim represents and stores information for the IndividualFim.

Usage

```
IndividualFim(
  fisherMatrix = numeric(0),
  fixedEffects = numeric(0),
  varianceEffects = numeric(0),
  SEAndRSE = list(),
  condNumberFixedEffects = numeric(0),
  condNumberVarianceEffects = numeric(0),
  shrinkage = numeric(0)
)
```

Arguments

fisherMatrix	A matrix giving the numerical values of the Fim.
fixedEffects	A matrix giving the numerical values of the fixedEffects of the Fim.
varianceEffects	A matrix giving the numerical values of varianceEffects of the Fim.
SEAndRSE	A data frame giving the value of the SE and RSE.
condNumberFixedEffects	The conditional number of the fixedEffects of the Fim.
condNumberVarianceEffects	The conditional number of the varianceEffects of the Fim.
shrinkage	A vector giving the shrinkage values.

Details

IndividualFim

`LibraryOfModels` *LibraryOfModels*

Description

The class `LibraryOfModels` represents and stores information for the LibraryOfModels.

Usage

```
LibraryOfModels(models = list())
```

Arguments

`models`	A list giving all the PK and PD models.

Details

LibraryOfModels

`LibraryOfPDModels` *LibraryOfPDModels*

Description

The class `LibraryOfPDModels` represents and stores information for the LibraryOfPDModels.

Usage

```
LibraryOfPDModels
```

Format

An object of class `PFIM::LibraryOfPDModels` (inherits from `PFIM::LibraryOfModels`, `S7_object`) of length 1.

Details

LibraryOfPDModels

LibraryOfPKModels *LibraryOfPKModels*

Description

The class `LibraryOfPKModels` represents and stores information for the LibraryOfPKModels.

Usage

```
LibraryOfPKModels
```

Format

An object of class `PFIM::LibraryOfPKModels` (inherits from `PFIM::LibraryOfModels`, `S7_object`) of length 1.

Details

LibraryOfPKModels

Linear2BolusSingleDose_ClQV1V2 *Model Linear2BolusSingleDose_ClQV1V2*

Description

Model Linear2BolusSingleDose_ClQV1V2

Usage

```
Linear2BolusSingleDose_ClQV1V2()
```

Linear2BolusSingleDose_kk12k21V *Model Linear2BolusSingleDose_kk12k21V*

Description

Model Linear2BolusSingleDose_kk12k21V

Usage

```
Linear2BolusSingleDose_kk12k21V()
```

Linear2BolusSteadyState_ClQV1V2tau

Model Linear2BolusSteadyState_ClQV1V2tau

Description

Model Linear2BolusSteadyState_ClQV1V2tau

Usage

```
Linear2BolusSteadyState_ClQV1V2tau()
```

Linear2BolusSteadyState_kk12k21Vtau

Model Linear2BolusSteadyState_kk12k21Vtau

Description

Model Linear2BolusSteadyState_kk12k21Vtau

Usage

```
Linear2BolusSteadyState_kk12k21Vtau()
```

Linear2FirstOrderSingleDose_kaClQV1V2

Model Linear2FirstOrderSingleDose_kaClQV1V2

Description

Model Linear2FirstOrderSingleDose_kaClQV1V2

Usage

```
Linear2FirstOrderSingleDose_kaClQV1V2()
```

Linear2FirstOrderSingleDose_kakk12k21V

Model Linear2FirstOrderSingleDose_kakk12k21V

Description

Model Linear2FirstOrderSingleDose_kakk12k21V

Usage

```
Linear2FirstOrderSingleDose_kakk12k21V()
```

`Linear2FirstOrderSteadyState_kaClQV1V2tau`
Model Linear2FirstOrderSteadyState_kaClQV1V2tau

Description

Model Linear2FirstOrderSteadyState_kaClQV1V2tau

Usage

```
Linear2FirstOrderSteadyState_kaClQV1V2tau()
```

`Linear2FirstOrderSteadyState_kakk12k21Vtau`
Model Linear2FirstOrderSteadyState_kakk12k21Vtau

Description

Model Linear2FirstOrderSteadyState_kakk12k21Vtau

Usage

```
Linear2FirstOrderSteadyState_kakk12k21Vtau()
```

`Linear2InfusionSingleDose_ClQV1V2`
Model Linear2InfusionSingleDose_ClQV1V2

Description

Model Linear2InfusionSingleDose_ClQV1V2

Usage

```
Linear2InfusionSingleDose_ClQV1V2()
```

`Linear2InfusionSingleDose_kk12k21V`
Model Linear2InfusionSingleDose_kk12k21V

Description

Model Linear2InfusionSingleDose_kk12k21V

Usage

```
Linear2InfusionSingleDose_kk12k21V()
```

Linear2InfusionSteadyState_ClQV1V2tau *Model Linear2InfusionSteadyState_ClQV1V2tau*

Description

Model Linear2InfusionSteadyState_ClQV1V2tau

Usage

```
Linear2InfusionSteadyState_ClQV1V2tau()
```

Linear2InfusionSteadyState_kk12k21Vtau *Model Linear2InfusionSteadyState_kk12k21Vtau*

Description

Model Linear2InfusionSteadyState_kk12k21Vtau

Usage

```
Linear2InfusionSteadyState_kk12k21Vtau()
```

LogNormal *LogNormal*

Description

The class `LogNormal` represents and stores information for the LogNormal distribution.

Usage

```
LogNormal(name = character(0), mu = numeric(0), omega = numeric(0))
```

Arguments

`name`	A string giving the name of the distribution.
`mu`	A double giving the mean mu.
`omega`	A double giving omega.

Details

LogNormal

Model *Model*

Description

The class Model represents and stores information for a model.

Usage

```
Model(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list()
)
```

Arguments

name	A string giving the name of the model.
modelParameters	A list giving the model parameters.
samplings	A vector of double giving the samplings of the model.
modelEquations	A list giving the model equations.
wrapper	The method wrapper used to get the ode solver format.
outputFormula	A list giving the formula of the evaluation outputs.
outputNames	A string giving the names of the model outputs.
variableNames	A string giving the names of the model variables.
outcomesWithAdministration	A vector giving the outcomes with administration.
outcomesWithNoAdministration	A vector giving the outcomes with no administration.
modelError	A list of the models error.
odeSolverParameters	A list of the parameters for the ode solver.
parametersForComputingGradient	A list for the parameters used in the computation of the Hessian.

initialConditions
: A list giving the initial conditions of the model.

functionArguments
: A list giving the functionArguments of the wrapper.

functionArgumentsSymbol
: A list giving the functionArgumentsSymbol of the wrapper.

Details

Model

ModelAnalytic *ModelAnalytic*

Description

The class ModelAnalytic is used to defined an analytic model.

Usage

```
ModelAnalytic(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list(),
  modelAnalytic = function() NULL,
  wrapperModelAnalytic = list(),
  functionArgumentsModelAnalytic = list(),
  functionArgumentsSymbolModelAnalytic = list(),
  solverInputs = list()
)
```

Arguments

name
: A string giving the name of the model.

modelParameters
: A list giving the model parameters.

samplings
: A vector of double giving the samplings of the model.

`modelEquations` A list giving the model equations.

`wrapper` The method wrapper used to get the ode solver format.

`outputFormula` A list giving the formula of the evaluation outputs.

`outputNames` A string giving the names of the model outputs.

`variableNames` A string giving the names of the model variables.

`outcomesWithAdministration`
A vector giving the outcomes with administration.

`outcomesWithNoAdministration`
A vector giving the outcomes with no administration.

`modelError` A list of the models error.

`odeSolverParameters`
A list of the parameters for the ode solver.

`parametersForComputingGradient`
A list for the parameters used in the computation of the Hessian.

`initialConditions`
A list giving the initial conditions of the model.

`functionArguments`
A list giving the functionArguments of the wrapper.

`functionArgumentsSymbol`
A list giving the functionArgumentsSymbol of the wrapper.

`modelAnalytic` An object `ModelAnalytic`

`wrapperModelAnalytic`
Wrapper for the ode solver.

`functionArgumentsModelAnalytic`
A list giving the functionArguments of the wrapper for the analytic model.

`functionArgumentsSymbolModelAnalytic`
A list giving the functionArgumentsSymbol of the wrapper for the analytic model.

`solverInputs` A list giving the solver inputs.

Details

ModelAnalytic

`ModelAnalyticInfusion` *ModelAnalyticInfusion*

Description

The class `ModelAnalyticInfusion` is used to defined an analytic model in infusion.

Usage

```
ModelAnalyticInfusion(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list(),
  wrapperModelAnalyticInfusion = list(),
  functionArgumentsModelAnalyticInfusion = list(),
  functionArgumentsSymbolModelAnalyticInfusion = list(),
  solverInputs = list()
)
```

Arguments

`name`	A string giving the name of the model.
`modelParameters`	A list giving the model parameters.
`samplings`	A vector of double giving the samplings of the model.
`modelEquations`	A list giving the model equations.
`wrapper`	The method wrapper used to get the ode solver format.
`outputFormula`	A list giving the formula of the evaluation outputs.
`outputNames`	A string giving the names of the model outputs.
`variableNames`	A string giving the names of the model variables.
`outcomesWithAdministration`	A vector giving the outcomes with administration.
`outcomesWithNoAdministration`	A vector giving the outcomes with no administration.
`modelError`	A list of the models error.
`odeSolverParameters`	A list of the parameters for the ode solver.
`parametersForComputingGradient`	A list for the parameters used in the computation of the Hessian.
`initialConditions`	A list giving the initial conditions of the model.
`functionArguments`	A list giving the functionArguments of the wrapper.

`functionArgumentsSymbol`
: A list giving the functionArgumentsSymbol of the wrapper.

`wrapperModelAnalyticInfusion`
: Wrapper for the ode solver.

`functionArgumentsModelAnalyticInfusion`
: A list giving the functionArguments of the wrapper for the analytic model in infusion.

`functionArgumentsSymbolModelAnalyticInfusion`
: A list giving the functionArgumentsSymbol of the wrapper for the analytic model in infusion.

`solverInputs` A list giving the solver inputs.

Details

ModelAnalyticInfusion

`ModelAnalyticInfusionSteadyState` *ModelAnalyticInfusionSteadyState*

Description

The class `ModelAnalyticInfusionSteadyState` is used to defined an analytic model in infusion steady state.

Usage

```
ModelAnalyticInfusionSteadyState(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list(),
  wrapperModelAnalyticInfusion = list(),
  functionArgumentsModelAnalyticInfusion = list(),
  functionArgumentsSymbolModelAnalyticInfusion = list(),
  solverInputs = list()
)
```

Arguments

name	A string giving the name of the model.
modelParameters	A list giving the model parameters.
samplings	A vector of double giving the samplings of the model.
modelEquations	A list giving the model equations.
wrapper	The method wrapper used to get the ode solver format.
outputFormula	A list giving the formula of the evaluation outputs.
outputNames	A string giving the names of the model outputs.
variableNames	A string giving the names of the model variables.
outcomesWithAdministration	A vector giving the outcomes with administration.
outcomesWithNoAdministration	A vector giving the outcomes with no administration.
modelError	A list of the models error.
odeSolverParameters	A list of the parameters for the ode solver.
parametersForComputingGradient	A list for the parameters used in the computation of the Hessian.
initialConditions	A list giving the initial conditions of the model.
functionArguments	A list giving the functionArguments of the wrapper.
functionArgumentsSymbol	A list giving the functionArgumentsSymbol of the wrapper.
wrapperModelAnalyticInfusion	Wrapper for the ode solver.
functionArgumentsModelAnalyticInfusion	A list giving the functionArguments of the wrapper for the analytic model in infusion.
functionArgumentsSymbolModelAnalyticInfusion	A list giving the functionArgumentsSymbol of the wrapper for the analytic model in infusion.
solverInputs	A list giving the solver inputs.

Details

ModelAnalyticInfusionSteadyState

ModelAnalyticSteadyState
ModelAnalyticSteadyState

Description

The class `ModelAnalyticSteadyState` is used to defined an analytic model in steady state.

Usage

```
ModelAnalyticSteadyState(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list(),
  modelAnalytic = function() NULL,
  wrapperModelAnalytic = list(),
  functionArgumentsModelAnalytic = list(),
  functionArgumentsSymbolModelAnalytic = list(),
  solverInputs = list()
)
```

Arguments

`name`	A string giving the name of the model.
`modelParameters`	A list giving the model parameters.
`samplings`	A vector of double giving the samplings of the model.
`modelEquations`	A list giving the model equations.
`wrapper`	The method wrapper used to get the ode solver format.
`outputFormula`	A list giving the formula of the evaluation outputs.
`outputNames`	A string giving the names of the model outputs.
`variableNames`	A string giving the names of the model variables.
`outcomesWithAdministration`	A vector giving the outcomes with administration.
`outcomesWithNoAdministration`	A vector giving the outcomes with no administration.

`modelError` A list of the models error.

`odeSolverParameters`
A list of the parameters for the ode solver.

`parametersForComputingGradient`
A list for the parameters used in the computation of the Hessian.

`initialConditions`
A list giving the initial conditions of the model.

`functionArguments`
A list giving the functionArguments of the wrapper.

`functionArgumentsSymbol`
A list giving the functionArgumentsSymbol of the wrapper.

`modelAnalytic` A object `ModelAnalytic` giving the analytic model.

`wrapperModelAnalytic`
Wrapper for the ode solver.

`functionArgumentsModelAnalytic`
A list giving the functionArguments of the wrapper for the analytic model in steady state.

`functionArgumentsSymbolModelAnalytic`
A list giving the functionArgumentsSymbol of the wrapper for the analytic model in steady state.

`solverInputs` A list giving the solver inputs.

Details

ModelAnalyticSteadyState

`ModelError` *ModelError*

Description

The class `ModelError` is used to defined a model error.

Usage

```
ModelError(
  output = character(0),
  equation = expression(),
  derivatives = list(),
  sigmaInter = numeric(0),
  sigmaSlope = numeric(0),
  sigmaInterFixed = logical(0),
  sigmaSlopeFixed = logical(0),
  cError = numeric(0)
)
```

Arguments

output	A string giving the model error output.
equation	A expression giving the model error equation.
derivatives	A list giving the derivatives of the model error equation.
sigmaInter	A double giving the sigma inter.
sigmaSlope	A double giving the sigma slope
sigmaInterFixed	A boolean giving if the sigma inter is fixed or not. - not in the v7.0
sigmaSlopeFixed	A boolean giving if the sigma slope is fixed or not. - not in the v7.0
cError	A integer giving the power parameter.

Details

ModelError

`ModelInfusion` *ModelInfusion*

Description

The class `ModelInfusion` is used to defined a model in infusion.

Usage

```
ModelInfusion(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list()
)
```

Arguments

name	A string giving the name of the model.
modelParameters	A list giving the model parameters.
samplings	A vector of double giving the samplings of the model.
modelEquations	A list giving the model equations.
wrapper	The method wrapper used to get the ode solver format.
outputFormula	A list giving the formula of the evaluation outputs.
outputNames	A string giving the names of the model outputs.
variableNames	A string giving the names of the model variables.
outcomesWithAdministration	A vector giving the outcomes with administration.
outcomesWithNoAdministration	A vector giving the outcomes with no administration.
modelError	A list of the models error.
odeSolverParameters	A list of the parameters for the ode solver.
parametersForComputingGradient	A list for the parameters used in the computation of the Hessian.
initialConditions	A list giving the initial conditions of the model.
functionArguments	A list giving the functionArguments of the wrapper.
functionArgumentsSymbol	A list giving the functionArgumentsSymbol of the wrapper.

Details

ModelInfusion

ModelODE *ModelODE*

Description

The class ModelODE is used to defined a ode model.

Usage

```
ModelODE(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
```

```
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list()
)
```

Arguments

`name`	A string giving the name of the model.
`modelParameters`	A list giving the model parameters.
`samplings`	A vector of double giving the samplings of the model.
`modelEquations`	A list giving the model equations.
`wrapper`	The method wrapper used to get the ode solver format.
`outputFormula`	A list giving the formula of the evaluation outputs.
`outputNames`	A string giving the names of the model outputs.
`variableNames`	A string giving the names of the model variables.
`outcomesWithAdministration`	A vector giving the outcomes with administration.
`outcomesWithNoAdministration`	A vector giving the outcomes with no administration.
`modelError`	A list of the models error.
`odeSolverParameters`	A list of the parameters for the ode solver.
`parametersForComputingGradient`	A list for the parameters used in the computation of the Hessian.
`initialConditions`	A list giving the initial conditions of the model.
`functionArguments`	A list giving the functionArguments of the wrapper.
`functionArgumentsSymbol`	A list giving the functionArgumentsSymbol of the wrapper.

Details

ModelODE

ModelODEBolus *ModelODEBolus*

Description

The class ModelODEBolus is used to defined a model ode admin bolus.

Usage

```
ModelODEBolus(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list(),
  modelODE = function() NULL,
  doseEvent = list(),
  solverInputs = list()
)
```

Arguments

name	A string giving the name of the model.
modelParameters	A list giving the model parameters.
samplings	A vector of double giving the samplings of the model.
modelEquations	A list giving the model equations.
wrapper	The method wrapper used to get the ode solver format.
outputFormula	A list giving the formula of the evaluation outputs.
outputNames	A string giving the names of the model outputs.
variableNames	A string giving the names of the model variables.
outcomesWithAdministration	A vector giving the outcomes with administration.
outcomesWithNoAdministration	A vector giving the outcomes with no administration.
modelError	A list of the models error.
odeSolverParameters	A list of the parameters for the ode solver.

`parametersForComputingGradient`
: parametersForComputingGradient.

`initialConditions`
: A list giving the initial conditions of the model.

`functionArguments`
: A list giving the functionArguments of the wrapper.

`functionArgumentsSymbol`
: A list giving the functionArgumentsSymbol of the wrapper.

`modelODE`
: An object `modelODE`.

`doseEvent`
: A dataframge given the doseEvent for the ode solver.

`solverInputs`
: solverInputs

Details

ModelODEBolus

`ModelODEDoseInEquations` *ModelODEDoseNotInEquations*

Description

The class `ModelODEDoseNotInEquations` is used to defined a ModelODEDoseNotInEquations

Usage

```
ModelODEDoseInEquations(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list(),
  modelODEDoseInEquations = function() NULL,
  solverInputs = list()
)
```

Arguments

name	A string giving the name of the model.
modelParameters	A list giving the model parameters.
samplings	A vector of double giving the samplings of the model.
modelEquations	A list giving the model equations.
wrapper	The method wrapper used to get the ode solver format.
outputFormula	A list giving the formula of the evaluation outputs.
outputNames	A string giving the names of the model outputs.
variableNames	A string giving the names of the model variables.
outcomesWithAdministration	A vector giving the outcomes with administration.
outcomesWithNoAdministration	A vector giving the outcomes with no administration.
modelError	A list of the models error.
odeSolverParameters	A list of the parameters for the ode solver.
parametersForComputingGradient	A list for the parameters used in the computation of the Hessian.
initialConditions	A list giving the initial conditions of the model.
functionArguments	A list giving the functionArguments of the wrapper.
functionArgumentsSymbol	A list giving the functionArgumentsSymbol of the wrapper.
modelODEDoseInEquations	An object modelODEDoseInEquations.
solverInputs	A list giving the solver inputs.

Details

ModelODEDoseNotInEquations

ModelODEDoseNotInEquations

ModelODEDoseNotInEquations

Description

The class ModelODEDoseNotInEquations is used to defined a ModelODEDoseNotInEquations

Usage

```
ModelODEDoseNotInEquations(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list(),
  modelODE = function() NULL,
  doseEvent = list(),
  solverInputs = list()
)
```

Arguments

`name`	A string giving the name of the model.
`modelParameters`	A list giving the model parameters.
`samplings`	A vector of double giving the samplings of the model.
`modelEquations`	A list giving the model equations.
`wrapper`	The method wrapper used to get the ode solver format.
`outputFormula`	A list giving the formula of the evaluation outputs.
`outputNames`	A string giving the names of the model outputs.
`variableNames`	A string giving the names of the model variables.
`outcomesWithAdministration`	A vector giving the outcomes with administration.
`outcomesWithNoAdministration`	A vector giving the outcomes with no administration.
`modelError`	A list of the models error.
`odeSolverParameters`	A list of the parameters for the ode solver.
`parametersForComputingGradient`	A list for the parameters used in the computation of the Hessian.
`initialConditions`	A list giving the initial conditions of the model.
`functionArguments`	A list giving the functionArguments of the wrapper.
`functionArgumentsSymbol`	A list giving the functionArgumentsSymbol of the wrapper.

modelODE	An object modelODE.
doseEvent	A dataframge given the doseEvent for the ode solver.
solverInputs	A list giving the solver inputs.

Details

ModelODEDoseNotInEquations

ModelODEInfusion *ModelODEInfusion*

Description

The class ModelODEInfusion is used to defined a model ModelODEInfusion.

Usage

```
ModelODEInfusion(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list()
)
```

Arguments

name	A string giving the name of the model.
modelParameters	A list giving the model parameters.
samplings	A vector of double giving the samplings of the model.
modelEquations	A list giving the model equations.
wrapper	The method wrapper used to get the ode solver format.
outputFormula	A list giving the formula of the evaluation outputs.
outputNames	A string giving the names of the model outputs.
variableNames	A string giving the names of the model variables.
outcomesWithAdministration	A vector giving the outcomes with administration.

`outcomesWithNoAdministration`
: A vector giving the outcomes with no administration.

`modelError`
: A list of the models error.

`odeSolverParameters`
: A list of the parameters for the ode solver.

`parametersForComputingGradient`
: A list for the parameters used in the computation of the Hessian.

`initialConditions`
: A list giving the initial conditions of the model.

`functionArguments`
: A list giving the functionArguments of the wrapper.

`functionArgumentsSymbol`
: A list giving the functionArgumentsSymbol of the wrapper.

Details

ModelODEInfusion

`ModelODEInfusionDoseInEquation`
ModelODEInfusionDoseInEquation

Description

The class `ModelODEInfusionDoseInEquation` is used to defined a ModelODEInfusionDoseInEquation

Usage

```
ModelODEInfusionDoseInEquation(
  name = character(0),
  modelParameters = list(),
  samplings = numeric(0),
  modelEquations = list(),
  wrapper = function() NULL,
  outputFormula = list(),
  outputNames = character(0),
  variableNames = character(0),
  outcomesWithAdministration = character(0),
  outcomesWithNoAdministration = character(0),
  modelError = list(),
  odeSolverParameters = list(),
  parametersForComputingGradient = list(),
  initialConditions = numeric(0),
  functionArguments = character(0),
  functionArgumentsSymbol = list(),
  modelODE = function() NULL,
  wrapperModelInfusion = list(),
  solverInputs = list()
)
```

Arguments

name	A string giving the name of the model.
modelParameters	A list giving the model parameters.
samplings	A vector of double giving the samplings of the model.
modelEquations	A list giving the model equations.
wrapper	Wrapper for solver.
outputFormula	A list giving the formula of the evaluation outputs.
outputNames	A string giving the names of the model outputs.
variableNames	A string giving the names of the model variables.
outcomesWithAdministration	A vector giving the outcomes with administration.
outcomesWithNoAdministration	A vector giving the outcomes with no administration.
modelError	A list of the models error.
odeSolverParameters	A list of the parameters for the ode solver.
parametersForComputingGradient	A list for the parameters used in the computation of the Hessian.
initialConditions	A list giving the initial conditions of the model.
functionArguments	A list giving the functionArguments of the wrapper.
functionArgumentsSymbol	A list giving the functionArgumentsSymbol of the wrapper..
modelODE	An object modelODE.
wrapperModelInfusion	Wrapper for solver.
solverInputs	A list giving the solver inputs.

Details

ModelODEInfusionDoseInEquation

ModelParameter *ModelParameter*

Description

The class ModelParameter is used to defined the model parameters.

Usage

```
ModelParameter(
  name = character(0),
  distribution = Distribution(),
  fixedMu = FALSE,
  fixedOmega = FALSE
)
```

Arguments

name	A string giving the name of the parameter.
distribution	A string giving the distribution of the parameter.
fixedMu	A boolean setting TRUE/FALSE if the mu is estimated or not.
fixedOmega	A boolean setting TRUE/FALSE if the omega is estimated or not.

Details

ModelParameter

MultiplicativeAlgorithm

MultiplicativeAlgorithm

Description

The class MultiplicativeAlgorithm implements the multiplicative algorithm.

Usage

```
MultiplicativeAlgorithm(
  name = character(0),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelParameters = list(),
  modelError = list(),
  optimizer = character(0),
  optimizerParameters = list(),
  outputs = list(),
  designs = list(),
  fimType = character(0),
  fim = Fim(),
  odeSolverParameters = list(),
  optimisationDesign = list(),
  optimisationAlgorithmOutputs = list(),
  lambda = integer(0),
  delta = integer(0),
  numberOfIterations = integer(0),
  weightThreshold = integer(0),
  showProcess = logical(0),
  multiplicativeAlgorithmOutputs = list()
)
```

Arguments

name	A string giving the name of the optimization process.
modelEquations	A list giving the model equations.
modelFromLibrary	A list giving the name of the model for the mode equations.

`modelParameters`
: A list giving the model parameters.

`modelError`
: A list giving the model error.

`optimizer`
: A string giving the name of the optimization algorithm.

`optimizerParameters`
: A list giving the parameters of the optimization algorithm.

`outputs`
: A list giving the ouput of the model.

`designs`
: A list giving the designs.

`fimType`
: A string giving the type of the Fim.

`fim`
: A object `Fim` giving the Fim.

`odeSolverParameters`
: A list giving the ode parameters for solver.

`optimisationDesign`
: A list giving the evaluation of initial and optimal design.

`optimisationAlgorithmOutputs`
: A list giving the outputs of the optimization process.

`lambda`
: A numeric giving the parameter lambda.

`delta`
: A numeric giving the parameter delta

`numberOfIterations`
: A numeric giving the number of iterations.

`weightThreshold`
: A numeric giving the weight threshold.

`showProcess`
: A boolean for displaying the process or not.

`multiplicativeAlgorithmOutputs`
: A list giving the output of the optimization algorithm.

Details

Class "MultiplicativeAlgorithm"

`MultiplicativeAlgorithm_Rcpp`
Function MultiplicativeAlgorithm_Rcpp

Description

Run the MultiplicativeAlgorithm_Rcpp in Rcpp.

Usage

```
MultiplicativeAlgorithm_Rcpp(
  fisherMatrices_input,
  numberOfFisherMatrices_input,
  weights_input,
  numberOfParameters_input,
  dim_input,
  lambda_input,
  delta_input,
  iterationInit_input
)
```

Arguments

`fisherMatrices_input`
: The parameter fotfisherMatrices_input.

`numberOfFisherMatrices_input`
: The parameter numberOfFisherMatrices_input.

`weights_input`
: The parameter weights_input.

`numberOfParameters_input`
: The parameter numberOfParameters_input.

`dim_input`
: The parameter dim_input.

`lambda_input`
: The parameter lambda_input.

`delta_input`
: The parameter delta_input.

`iterationInit_input`
: The parameter iterationInit_input.

Value

The list output with the outputs of the MultiplicativeAlgorithm_Rcpp.

`Normal` *Normal*

Description

The class `Normal` implements the Normal distribution.

Usage

```
Normal(name = character(0), mu = numeric(0), omega = numeric(0))
```

Arguments

`name`
: A string giving the name of the distribution.

`mu`
: A double giving the mean mu.

`omega`
: A double giving omega.

Details

Normal

`Optimization` *Optimization*

Description

The class `Optimization` implements the Optimization.

Usage

```
Optimization(
  name = character(0),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelParameters = list(),
  modelError = list(),
  optimizer = character(0),
  optimizerParameters = list(),
  outputs = list(),
  designs = list(),
  fimType = character(0),
  fim = Fim(),
  odeSolverParameters = list(),
  optimisationDesign = list(),
  optimisationAlgorithmOutputs = list()
)
```

Arguments

`name`	A string giving the name of the design evaluation.
`modelEquations`	A list giving the model equations.
`modelFromLibrary`	A list giving the model equations from the library of model.
`modelParameters`	A list giving the model parameters.
`modelError`	A list giving the model error.
`optimizer`	A string giving the name of the optimization algorithm being used.
`optimizerParameters`	A list giving the parameters of the optimization algorithm.
`outputs`	A list giving the model outputs.
`designs`	A list giving the designs to be evaluated.
`fimType`	A string giving the type of Fim being evaluated.
`fim`	A object `Fim` giving the Fim.
`odeSolverParameters`	A list giving the atol and rtol parameters for the ode solver.
`optimisationDesign`	A list giving the evaluation of initial and optimal design.
`optimisationAlgorithmOutputs`	A list giving the outputs of the optimization process.

Details

Optimization

`optimizeDesign: optimization of a design.`
Optimization PGBOAlgorithm

Description

Optimization PGBOAlgorithm

Optimization PSOAlgorithm

Optimization SimplexAlgorithm

Optimization FedorovWynnAlgorithm

Optimization MultiplicativeAlgorithm

Arguments

`optimizationObject`
: A object `Optimization`.

`optimizationAlgorithm`
: A object `MultiplicativeAlgorithm`.

Value

The object `optimizationObject` with the slots updated.

The object `optimizationObject` with the slots updated.

The object `optimizationObject` with the slots updated.

The object `optimizationObject` with the slots updated.

The object `optimizationObject` with the slots updated.

`PFIMProject` *PFIMProject*

Description

The class `PFIMProject` implements the PFIM project.

Usage

```
PFIMProject(
  name = character(0),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelParameters = list(),
  modelError = list(),
  optimizer = character(0),
  optimizerParameters = list(),
  outputs = list(),
  designs = list(),
  fimType = character(0),
  fim = Fim(),
  odeSolverParameters = list()
)
```

Arguments

`name`	A string giving the name of the design evaluation.
`modelEquations`	A list giving the model equations.
`modelFromLibrary`	A list giving the model equations from the library of model.
`modelParameters`	A list giving the model parameters.
`modelError`	A list giving the model error.
`optimizer`	A string giving the name of the optimization algorithm being used.
`optimizerParameters`	A list giving the parameters of the optimization algorithm.
`outputs`	A list giving the model outputs.
`designs`	A list giving the designs to be evaluated.
`fimType`	A string giving the type of Fim being evaluated.
`fim`	A object `Fim` giving the Fim.
`odeSolverParameters`	A list giving the atol and rtol parameters for the ode solver.

Details

PFIMProject

`PGBOAlgorithm` *PGBOAlgorithm*

Description

The class `PGBOAlgorithm` implements the PGBO algorithm.

Usage

```
PGBOAlgorithm(
  name = character(0),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelParameters = list(),
  modelError = list(),
  optimizer = character(0),
  optimizerParameters = list(),
  outputs = list(),
  designs = list(),
  fimType = character(0),
  fim = Fim(),
  odeSolverParameters = list(),
  optimisationDesign = list(),
  optimisationAlgorithmOutputs = list(),
  N = integer(0),
  muteEffect = integer(0),
  maxIteration = integer(0),
  purgeIteration = integer(0),
  seed = integer(0),
  showProcess = logical(0)
)
```

Arguments

`name`	A string giving the name.
`modelEquations`	A list giving the model equations.
`modelFromLibrary`	A list giving the model equations from the library of model.
`modelParameters`	A list giving the model parameters.
`modelError`	A list giving the model error.
`optimizer`	A string giving the name of the optimization algorithm being used.
`optimizerParameters`	A list giving the parameters of the optimization algorithm.
`outputs`	A list giving the model outputs.
`designs`	A list giving the designs to be evaluated.
`fimType`	A string giving the type of Fim being evaluated.
`fim`	A object `Fim` giving the Fim.
`odeSolverParameters`	A list giving the atol and rtol parameters for the ode solver.
`optimisationDesign`	A list giving the evaluation of initial and optimal design.
`optimisationAlgorithmOutputs`	A list giving the outputs of the optimization process.
`N`	A numeric giving the parameter N.
`muteEffect`	A numeric giving the parameter muteEffect.

maxIteration	A numeric giving the parameter maxIteration.
purgeIteration	A numeric giving the parameter purgeIteration.
seed	A numeric giving the parameter seed.
showProcess	A boolean giving showProcess.

Details

Class "PGBOAlgorithm"

`plotEvaluation` *plotEvaluation: plots for the evaluation of the model responses.*

Description

plotEvaluation: plots for the evaluation of the model responses.

Arguments

pfimproject	A object `PFIMProject`.
plotOptions	A list giving the plot options.

Value

All the plots for the evaluation of the model responses.

`plotEvaluationResults` *plotEvaluationResults: process for the evaluation of the responses.*

Description

plotEvaluationResults: process for the evaluation of the responses.

Arguments

arm	A object of class `Arm` giving the arm.
evaluationModel	A list giving the evaluation of the model.
outputNames	A list of string giving the output of the evaluation of the model.
samplingData	A list giving the sampling data from the method getSamplingData.
unitXAxis	A list giving the unit of the x-axis.
unitYAxis	A list giving the unit of the y-axis.
designName	A string giving the design name.

Value

A list giving the plot of the evaluation of the model responses.

plotEvaluationSI *plotEvaluationSI: process for the evaluation of the gradient of the responses.*

Description

plotEvaluationSI: process for the evaluation of the gradient of the responses.

Arguments

arm	A object of class Arm giving the arm.
evaluationModelGradient	A list giving the evaluation of the gradient of the model responses.
parametersNames	A vector of string giving the parameter names?
outputNames	A list of string giving the name of the outputs.
samplingData	A list giving the sampling data from the method getSamplingData.
unitXAxis	A list giving the unit of the x-axis.
unitYAxis	A list giving the unit of the y-axis.
designName	A string giving the design name.

Value

A list giving the plot of the evaluation of gradient of the model responses.

plotFrequencies *Plot frequencies for the FedorovWynn algorithm*

Description

Plot frequencies for the FedorovWynn algorithm

Arguments

optimization	An Optimization object.

Value

Graph of the optimal frequencies.

plotFrequenciesFedorovWynnAlgorithm *plotFrequenciesFedorovWynnAlgorithm*

Description

plotFrequenciesFedorovWynnAlgorithm

Arguments

optimization	optimization
optimizationAlgorithm	optimizationAlgorithm

Value

plotFrequenciesFedorovWynnAlgorithm

plotRSE *Plot relative standard errors*

Description

Plot relative standard errors

plotRSE: bar plot of the RSE.

Arguments

optimization	An Optimization object.
pfimproject	A object PFIMProject giving the Evaluation.

Value

Graph of relative standard errors

The bar plot of the RSE.

plotRSEFIM *plotRSEFIM: barplot for the RSE*

Description

plotRSEFIM: barplot for the RSE

plotRSEFIM: barplot for the RSE

plotRSEFIM: barplot for the RSE

Arguments

fim	An object PopulationFim giving the Fim.
evaluation	An object Evaluation giving the evaluation of the model.

Value

The bar plot of the RSE.

The bar plot of the RSE.

The bar plot of the RSE.

plotSE *Plot standard errors*

Description

Plot standard errors

plotSE: bar plot of the SE.

Arguments

optimization	An Optimization object.
pfimproject	A object PFIMProject giving the Evaluation.

Value

Graph of standard errors

The bar plot of the SE.

plotSEFIM *plotSEFIM: barplot for the SE*

Description

plotSEFIM: barplot for the SE

plotSEFIM: barplot for the SE

plotSEFIM: barplot for the SE

Arguments

`fim`	An object `PopulationFim` giving the Fim.
`evaluation`	An object `Evaluation` giving the evaluation of the model.

Value

The bar plot of the SE.

The bar plot of the SE.

The bar plot of the SE.

plotSensitivityIndices *Plot sensitivity indices.*

Description

Plot sensitivity indices.

plotSensitivityIndices: plots for the evaluation of the gradient of the model responses.

Arguments

`optimization`	An `Optimization` object.
`pfimproject`	A object `PFIMProject` giving the Evaluation.
`plotOptions`	A list giving the plot options.

Value

Graph of sensitivity indices.

All the plots for the evaluation of the gradient of the model responses.

plotShrinkage *plotShrinkage: plot the shrinkage values.*

Description

plotShrinkage: plot the shrinkage values.

Arguments

fim	An object BayesianFim giving the Fim.
evaluation	An object Evaluation giving the evaluation of the model.

Value

The bar plot of the shrinkage.

plotWeights *Plot weights for the multiplicative algorithm*

Description

Plot weights for the multiplicative algorithm

Arguments

optimization	An Optimization object.

Value

Plot of weights

plotWeightsMultiplicativeAlgorithm
plotWeightsMultiplicativeAlgorithm: plot the optimal weight.

Description

plotWeightsMultiplicativeAlgorithm: plot the optimal weight.

Arguments

optimization	A object Optimization.
optimizationAlgorithm	A object MultiplicativeAlgorithm.

Value

The graph plotWeight.

PopulationFim *PopulationFim*

Description

The class PopulationFim implements the PopulationFim.

Usage

```
PopulationFim(
  fisherMatrix = numeric(0),
  fixedEffects = numeric(0),
  varianceEffects = numeric(0),
  SEAndRSE = list(),
  condNumberFixedEffects = numeric(0),
  condNumberVarianceEffects = numeric(0),
  shrinkage = numeric(0)
)
```

Arguments

fisherMatrix	A matrix of numeric giving the fisherMatrix.
fixedEffects	A matrix of numeric giving the fixedEffects.
varianceEffects	A matrix of numeric giving the varianceEffects.
SEAndRSE	A data frame giving the SEAndRSE.
condNumberFixedEffects	A numeric giving the condNumberFixedEffects.
condNumberVarianceEffects	A numeric giving the condNumberVarianceEffects.
shrinkage	A vector of numeric giving the shrinkage.

Details

PopulationFim

processArmEvaluationResults *processArmEvaluationResults: process for the evaluation of an arm.*

Description

processArmEvaluationResults: process for the evaluation of an arm.

Arguments

arm	A object of class Arm giving the arm.
model	A object of class Model giving the model.
fim	A object of class Fim giving the fim.
designName	A string giving the name of the design.
plotOptions	A list giving the plot options.

Value

A list of ggplot object giving the plot of the responses ans the gradient responses of the the model.

---

processArmEvaluationSI

processArmEvaluationSI: process for the evaluation of the gradient of the responses.

Description

processArmEvaluationSI: process for the evaluation of the gradient of the responses.

Arguments

arm	A object of class Arm giving the arm.
model	A object of class Model giving the model.
fim	A object of class Fim giving the fim.
designName	A string giving the name of the design.

Value

A list giving the ggplot object of the plots of the gradient.

Proportional *Proportional*

Description

The class Proportional is used to defined a model error.

Usage

```
Proportional(
  output = "output",
  equation = expression(sigmaSlope),
  derivatives = list(),
  sigmaInter = NULL,
  sigmaSlope = NULL,
  sigmaInterFixed = FALSE,
  sigmaSlopeFixed = FALSE,
  cError = NULL
)
```

Arguments

output	A string giving the model error output.
equation	A expression giving the model error equation.
derivatives	A list giving the derivatives of the model error equation.
sigmaInter	A double giving the sigma inter.
sigmaSlope	A double giving the sigma slope
sigmaInterFixed	A boolean giving if the sigma inter is fixed or not. - not in the v7.0
sigmaSlopeFixed	A boolean giving if the sigma slope is fixed or not. - not in the v7.0
cError	A integer giving the power parameter.

Details

Proportional

PSOAlgorithm *PSOAlgorithm*

Description

The class PSOAlgorithm implements the PSO algorithm.

Usage

```
PSOAlgorithm(
  name = character(0),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelParameters = list(),
  modelError = list(),
  optimizer = character(0),
  optimizerParameters = list(),
  outputs = list(),
  designs = list(),
  fimType = character(0),
  fim = Fim(),
  odeSolverParameters = list(),
  optimisationDesign = list(),
  optimisationAlgorithmOutputs = list(),
  maxIteration = integer(0),
  populationSize = integer(0),
  seed = integer(0),
  personalLearningCoefficient = integer(0),
  globalLearningCoefficient = integer(0),
  showProcess = logical(0),
  iterationAndCriteria = list()
)
```

Arguments

`name`	A string giving the name.
`modelEquations`	A list giving the model equations.
`modelFromLibrary`	A list giving the model equations from the library of model.
`modelParameters`	A list giving the model parameters.
`modelError`	A list giving the model error.
`optimizer`	A string giving the name of the optimization algorithm being used.
`optimizerParameters`	A list giving the parameters of the optimization algorithm.
`outputs`	A list giving the model outputs.
`designs`	A list giving the designs to be evaluated.
`fimType`	A string giving the type of Fim being evaluated.
`fim`	A object `Fim` giving the Fim.
`odeSolverParameters`	A list giving the atol and rtol parameters for the ode solver.
`optimisationDesign`	A list giving the evaluation of initial and optimal design.
`optimisationAlgorithmOutputs`	A list giving the outputs of the optimization process.
`maxIteration`	A numeric giving the maxIteration.
`populationSize`	A numeric giving the populationSize.
`seed`	A numeric giving the seed.
`personalLearningCoefficient`	A numeric giving the personalLearningCoefficient.
`globalLearningCoefficient`	A numeric giving the globalLearningCoefficient.
`showProcess`	A boolean giving the showProcess.
`iterationAndCriteria`	A numeric giving the iterationAndCriteria.

Details

Class "PSOAlgorithm"

`replaceVariablesLibraryOfModels` *replaceVariablesLibraryOfModels: replace variable in the LibraryOfModels*

Description

replaceVariablesLibraryOfModels: replace variable in the LibraryOfModels

Usage

```
replaceVariablesLibraryOfModels(text, old, new)
```

Arguments

text	the text
old	old string
new	new string

Value

text with new string

Report *Generate optimization report*

Description

Generate optimization report

Report: generate the report.

Arguments

optimization	An Optimization object.
pfimproject	A object PFIMProject giving the Evaluation or Optimization.
outputPath	A string giving the path where the output are saved.
outputFile	A string giving the name of the output file.
plotOptions	A list giving the plot options.

Value

Generated report.

The html report of the design evaluation or optimization.

---

run *Run optimization*

Description

Run optimization

run: run the evaluation of a design.

Arguments

optimization	An Optimization object.
pfimproject	A object PFIMProject giving the Evaluation.

Value

The optimization design results.

The object `Evaluation` giving the design evaluation.

`SamplingTimeConstraints` *SamplingTimeConstraints*

Description

The class "SamplingTimeConstraints" implements the constraints for the sampling times.

Usage

```
SamplingTimeConstraints(
  outcome = character(0),
  initialSamplings = logical(0),
  fixedTimes = logical(0),
  numberOfsamplingsOptimisable = integer(0),
  samplingsWindows = list(),
  numberOfTimesByWindows = logical(0),
  minSampling = integer(0)
)
```

Arguments

`outcome`	A string giving the outcome.
`initialSamplings`	A vector of numeric giving the initialSamplings.
`fixedTimes`	A vector of numeric giving the fixedTimes.
`numberOfsamplingsOptimisable`	A vector of numeric giving the numberOfsamplingsOptimisable.
`samplingsWindows`	A vector of numeric giving the samplingsWindows.
`numberOfTimesByWindows`	A vector of numeric giving the numberOfTimesByWindows.
`minSampling`	A vector of numeric giving the minSampling.

Details

Class "SamplingTimeConstraints"

SamplingTimes *SamplingTimes*

Description

The class SamplingTimes is used to defined SamplingTimes.

Usage

```
SamplingTimes(outcome = character(0), samplings = numeric(0))
```

Arguments

outcome	A string giving the outcome.
samplings	A vector of numeric giving the samplings.

Details

SamplingTimes

setEvaluationFim *setEvaluationFim: set the Fim results.*

Description

setEvaluationFim: set the Fim results.

setEvaluationFim: set the Fim results.

setEvaluationFim: set the Fim results.

Arguments

fim	An object PopulationFim giving the Fim.
evaluation	An object Evaluation giving the evaluation of the model.

Value

The object Fim with its fisherMatrix, fixedEffects, shrinkage, condNumberFixedEffects, SEAndRSE.

The object IndividualFim with its fisherMatrix, fixedEffects, shrinkage, condNumberFixedEffects, SEAndRSE.

The object PopulationFim with its fisherMatrix, fixedEffects, shrinkage, condNumberFixedEffects, SEAndRSE.

`setOptimalArms` *setOptimalArms: set the optimal arms of an optimization algorithm.*

Description

setOptimalArms: set the optimal arms of an optimization algorithm.

setOptimalArms: set the optimal arms of an optimization algorithm.

setOptimalArms: set the optimal arms of an optimization algorithm.

setOptimalArms: set the optimal arms of an optimization algorithm.

setOptimalArms: set the optimal arms of an optimization algorithm.

setOptimalArms: set the optimal arms of an optimization algorithm.

Arguments

`fim`	An object `PopulationFim` giving the Fim.
`optimizationAlgorithm`	An object `FedorovWynnAlgorithm` giving the optimization algorithm.

Value

The optimal arms.

The optimal arms.

The list optimalArms.

The list optimalArms.

The list optimalArms.

The list optimalArms.

`setSamplingConstraintForOptimization` *setSamplingConstraintForOptimization: set the sampling time constraints for an arm for the design optimization.*

Description

setSamplingConstraintForOptimization: set the sampling time constraints for an arm for the design optimization.

Arguments

`design`	An object `Design` giving the design.

Value

The arm with the sampling time constraint for the design optimization.

show *Show optimization results*

Description

Show optimization results

show: show the evaluation in the R console.

Arguments

optimization	An Optimization object.
pfimproject	A object PFIMProject giving the Evaluation.

Value

Prints results to console.

The show of the evaluation of the design.

showFIM *showFIM: show the Fim in the R console.*

Description

showFIM: show the Fim in the R console.

showFIM: show the Fim in the R console.

showFIM: show the Fim in the R console.

Arguments

fim	An object IndividualFim giving the Fim.

Value

The fisherMatrix, fixedEffects, Determinant, condition numbers and D-criterion, Shrinkage and Parameters estimation

The fisherMatrix, fixedEffects, Determinant, condition numbers and D-criterion, Shrinkage and Parameters estimation

The fisherMatrix, fixedEffects, Determinant, condition numbers and D-criterion, Shrinkage and Parameters estimation

SimplexAlgorithm *SimplexAlgorithm*

Description

The class SimplexAlgorithm implements the Simplex algorithm.

Usage

```
SimplexAlgorithm(
  name = character(0),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelParameters = list(),
  modelError = list(),
  optimizer = character(0),
  optimizerParameters = list(),
  outputs = list(),
  designs = list(),
  fimType = character(0),
  fim = Fim(),
  odeSolverParameters = list(),
  optimisationDesign = list(),
  optimisationAlgorithmOutputs = list(),
  pctInitialSimplexBuilding = integer(0),
  maxIteration = integer(0),
  tolerance = integer(0),
  seed = integer(0),
  showProcess = logical(0)
)
```

Arguments

name	A string giving the name.
modelEquations	A list giving the model equations.
modelFromLibrary	A list giving the model equations from the library of model.
modelParameters	A list giving the model parameters.
modelError	A list giving the model error.
optimizer	A string giving the name of the optimization algorithm being used.
optimizerParameters	A list giving the parameters of the optimization algorithm.
outputs	A list giving the model outputs.
designs	A list giving the designs to be evaluated.
fimType	A string giving the type of Fim being evaluated.
fim	A object Fim giving the Fim.

`odeSolverParameters`
: A list giving the atol and rtol parameters for the ode solver.

`optimisationDesign`
: A list giving the evaluation of initial and optimal design.

`optimisationAlgorithmOutputs`
: A list giving the outputs of the optimization process.

`pctInitialSimplexBuilding`
: A numeric giving the pctInitialSimplexBuilding.

`maxIteration` A numeric giving the maxIteration.

`tolerance` A numeric giving the tolerance.

`seed` A numeric giving the seed.

`showProcess` A boolean giving the showProcess.

Details

Class "SimplexAlgorithm"

`tablesForReport` *tablesForReport: generate the table for the report.*

Description

tablesForReport: generate the table for the report.

tablesForReport: generate the table for the report.

tablesForReport: generate the table for the report.

Arguments

`fim` An object `PopulationFim` giving the Fim.

`evaluation` An object `Evaluation` giving the evaluation of the model.

Value

fixedEffectsTable, FIMCriteriaTable, SEAndRSETable.

fixedEffectsTable, FIMCriteriaTable, SEAndRSETable.

fixedEffectsTable, FIMCriteriaTable, SEAndRSETable.

`updateSamplingTimes` *updateSamplingTimes: update sampling times for plotting used for plot*

Description

updateSamplingTimes: update sampling times for plotting used for plot

Arguments

`arm`	A object of class `Arm` giving the arm.
`samplingData`	The list giving as output in the method getSamplingData.

Value

The updated sampling times.

Index