Package 'PFIM'

May 7, 2026

Type Package

Title Population Fisher Information Matrix

Version 7.0.3

Date 2026-04-09

Maintainer Romain Leroux <romainlerouxPFIM@gmail.com>

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/PFIM/issues

Depends R (>= 4.0.0)

License GPL (>= 3)

Encoding UTF-8

VignetteBuilder knitr

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

Collate 'Administration.R' 'AdministrationConstraints.R' 'Fim.R' 'PFIMProject.R' 'Optimization.R' 'PGBOAlgorithm.R' 'PSOAlgorithm.R' 'SimplexAlgorithm.R' 'FedorovWynnAlgorithm.R' 'MultiplicativeAlgorithm.R' 'Model.R' 'Arm.R' 'BayesianFim.R' 'ModelError.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' 'plotMethods.R' 'utils.R' 'zzz.R'

RoxygenNote 7.3.3

Suggests rmarkdown, testthat (>= 3.0.0)

Author Romain Leroux [aut, cre] (ORCID:
<https://orcid.org/0009-0009-5779-5303>),
France Mentré [aut] (ORCID: <https://orcid.org/0000-0002-7045-1275>),
Jérémy Seurat [ctb]

Repository CRAN

Date/Publication 2026-04-09 09:30:22 UTC

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. a description of 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> (ORCID)

Authors:

- France Mentré <france.mentre@inserm.fr> (ORCID)

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/PFIM/issues

`adjustGradient` *Adjust the gradient for the log normal distribution.*

Description

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 Class*

Description

The `Administration` class defines the dosing regimen for a specific model outcome. It stores comprehensive information regarding dose amounts, administration timings, infusion durations, and dosing intervals (tau).

Usage

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

Arguments

outcome	A string identifying the target outcome for the administration.
timeDose	A numeric vector of dosing times.
dose	A numeric vector of dose amounts.
Tinf	A numeric vector specifying infusion durations.
tau	A numeric value representing the dosing interval (for multiple doses).

Value

An object of class Administration.

Slots

outcome character. The name of the model output (e.g., "PK").

timeDose numeric vector. The time points at which doses are administered.

dose numeric vector. The amount of drug administered at each time point.

Tinf numeric vector. The duration of the infusion (defaults to 0 for bolus).

tau numeric. The dosing interval for repeated doses or steady-state calculations.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
# Example 1: Single bolus dose at time 0
administrationRespPK = Administration(
  outcome  = "RespPK",
  timeDose = 0,
  dose     = 0.2
)
print( administrationRespPK )

# Example 2: Multiple doses at various times
administrationRespPK = Administration(
  outcome  = "RespPK",
  timeDose = c(0, 10, 20),
  dose     = c(0.1, 0.2, 0.3)
)
print( administrationRespPK )

# Example 3: Multiple doses with a 2-hour infusion duration
administrationRespPK = Administration(
  outcome  = "RespPK",
```

```
  timeDose = c(0, 10, 20),
  dose     = c( 0.1, 0.2, 0.3),
  Tinf     = 2.0
)
print( administrationRespPK )

# Example 4: Repeated dosing with a 5-hour interval (tau)
administrationRespPK = Administration(
  outcome = "RespPK",
  dose    = c(0.1, 0.2, 0.3),
  tau     = 5
)
print( administrationRespPK )
```

`AdministrationConstraints`
AdministrationConstraints Class

Description

The `AdministrationConstraints` class defines the space of admissible doses for a specific model outcome. It is used by optimization algorithms to restrict dosage inputs to a set of discrete candidate values.

Usage

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

Arguments

`outcome`	A string identifying the target outcome for these constraints.
`doses`	A numeric vector containing the candidate dose values.

Value

An object of class `AdministrationConstraints`.

Slots

`outcome character`. The name of the model output (e.g., "PK").

`doses numeric vector`. A vector of authorized dose levels (discrete candidates).

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
# Define discrete dose candidates for a PK model outcome
administrationConstraintsRespK = AdministrationConstraints(
 outcome = "RespPK",
 doses   = list(0.2, 0.64, 2, 6.24, 11.24, 20) )
print( administrationConstraintsRespK )
```

Arm *Arm Class*

Description

The Arm class represents an experimental group within a study. It integrates all components of the design for that group, including the sample size, dosing regimens, sampling schedules, and initial conditions for ODE models.

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	An integer giving the number of subjects in the arm.
administrations	A list of Administration objects defining the dosing.
initialConditions	A named list of numeric values for ODE initial states.
samplingTimes	A list of SamplingTimes objects defining the observations.

`administrationsConstraints`
: A list of `AdministrationsConstraints` objects.

`samplingTimesConstraints`
: A list of `SamplingTimesConstraints` objects.

`evaluationModel`
: A list containing the evaluation of the responses.

`evaluationGradients`
: A list containing the evaluation of the gradients.

`evaluationVariance`
: A list containing the evaluation of the variance.

`evaluationFim`
: An object of class `Fim` representing the Fisher Information Matrix.

Value

An object of class `Arm`.

Slots

`name character`. The unique identifier for the arm.

`size numeric`. The number of subjects assigned to this arm.

`administrations list`. A list of `Administration` objects.

`initialConditions list`. A named list where keys are variable names (strings) and values are their initial states (numeric).

`samplingTimes list`. A list of `SamplingTimes` objects.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
# Note: The 'initialConditions' slot is strictly used for ODE-based model arms.
# For analytic (closed-form) solutions, this slot is ignored as the
# initial state is implicitly defined by the model equations.

# 1. Define sampling times for PK and PD outcomes
samplingTimesRespPK = SamplingTimes(outcome = "RespPK",
                                    samplings = c(0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4))

samplingTimesRespPD = SamplingTimes(outcome = "RespPD",
                                    samplings = c(0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4))

# 2. Define the administration (Dose of 20 at t=0)
adminRespPK = Administration(outcome = "RespPK", timeDose = 0, dose = 20)
```

```
# 3. Define the study arm "0.2mg"
# Outcomes are linked to state variables: RespPK to Cc, RespPD to E.
arm = Arm(name = "0.2mg",
          size = 6,
          administrations = list(adminRespPK),
          samplingTimes   = list(samplingTimesRespPK, samplingTimesRespPD),
          initialConditions = list("Cc" = 0, "E" = 100))

print(arm)
```

`armAdministration` *Get administration parameters of an arm*

Description

Extracts dosing information (outcome, dose, time dose, tau, Tinf) for each administration in the arm.

Usage

```
armAdministration(arm, ...)
```

Arguments

`arm`	An object of class `Arm`.
`...`	Additional arguments.

Value

A list of named lists, one per administration.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

BayesianFim *BayesianFim Class*

Description

The BayesianFim class stores the Bayesian Fisher Information Matrix (FIM). It extends the standard Fim class by incorporating Bayesian-specific metrics, such as the shrinkage of individual parameters.

Usage

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

Arguments

fisherMatrix	A numerical matrix representing the FIM.
fixedEffects	A matrix representing fixed effects information.
varianceEffects	A matrix representing variance components information.
SEAndRSE	A data frame containing calculated SE and RSE values.
condNumberFixedEffects	A numeric value for the fixed effects condition number.
condNumberVarianceEffects	A numeric value for the variance effects condition number.
shrinkage	A numeric vector representing parameter shrinkage.

Slots

fisherMatrix matrix. The numerical values of the Bayesian FIM.

shrinkage numeric vector. The shrinkage values for each random effect.

fixedEffects matrix. The FIM components related to fixed effects.

varianceEffects matrix. The FIM components related to variance components (random effects).

SEAndRSE data.frame. Standard Errors (SE) and Relative Standard Errors (RSE).

condNumberFixedEffects numeric. The condition number for the fixed effects sub-matrix.

condNumberVarianceEffects numeric. The condition number for the variance effects sub-matrix.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`checkSamplingTimeConstraintsForMetaheuristic`
Validate New Sampling Schedules Against Constraints

Description

The `checkSamplingTimeConstraintsForMetaheuristic` method evaluates whether a proposed set of sampling times is feasible. It checks the timings against defined windows, fixed points, and minimum intervals required for clinical safety or logistical practicality.

Arguments

`samplingTimesConstraints`

	An object of class `SamplingTimeConstraints` defining the allowed design space.
`arm`	An object of class `Arm` representing the experimental group being validated.
`newSamplings`	A `vector` of numeric values representing the candidate sampling times proposed by the algorithm.
`outcome`	A `string` specifying the model output (e.g., "PK", "PD") to which these samples belong.

Value

A `logical` value: `TRUE` if the design is valid, `FALSE` otherwise. If `FALSE`, a descriptive error message is usually printed to the console or stored in the optimization log.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

checkValiditySamplingConstraint *Validate optimization constraints*

Description

Validate optimization constraints

Arguments

design	An object Design.

Value

Returns nothing if valid, or stops with an error message if the sampling window/delta constraints are mathematically impossible.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Combined *Combined1 Class*

Description

The Combined1 class defines a combined residual error model, which incorporates both an additive and a proportional component.

Usage

```
Combined1(
  output = character(0),
  equation = expression(sigmaInter + sigmaSlope * output),
  derivatives = list(),
  sigmaInter = 0,
  sigmaSlope = 0,
  sigmaInterFixed = FALSE,
  sigmaSlopeFixed = FALSE,
  cError = 1
)
```

Arguments

output	A string specifying the model error output name.
equation	An expression representing the model error equation.
derivatives	A list of derivatives for the model error equation.
sigmaInter	A numeric value for the additive component (default 0).
sigmaSlope	A numeric value for the proportional component (default 0).
sigmaInterFixed	Logical; indicates if sigmaInter is fixed (default FALSE).
sigmaSlopeFixed	Logical; indicates if sigmaSlope is fixed (default FALSE).
cError	A numeric power parameter (default 1.0).

Value

An object of class Combined1.

Slots

output character. The name of the model output (e.g., "Cc").

sigmaInter numeric. The additive (intercept) error component.

sigmaSlope numeric. The proportional (slope) error component.

sigmaInterFixed logical. If TRUE, the intercept is fixed.

sigmaSlopeFixed logical. If TRUE, the slope is fixed.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

---

computeVMat *Compute Variance Matrix Component*

Description

The computeVMat method calculates the V matrix for a given set of model parameters. In the context of population modeling, V represents the total variance of the data, integrating both the structural model sensitivity to random effects and the residual error components.

Usage

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

Arguments

varParam1	A numeric vector or matrix representing the first set of variance components (typically related to the linearized structural model).
varParam2	A numeric vector or matrix representing the second set of variance components (typically the residual error terms).
invCholV	A logical or numeric matrix used for the Inverse Cholesky decomposition of V, facilitating faster computation of the FIM and likelihood.

Value

A square, symmetric matrix representing the total variance V for the observations.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Constant *Constant Class*

Description

The Constant class defines an additive residual error model, where the standard deviation (SD) of the error remains constant.

Usage

```
Constant(
  output = character(0),
  equation = expression(sigmaInter),
  derivatives = list(),
  sigmaInter = 0,
  sigmaSlope = 0,
  sigmaInterFixed = FALSE,
  sigmaSlopeFixed = FALSE,
  cError = 1
)
```

Arguments

output	A string specifying the name of the model output.
equation	An expression representing the model error equation.
derivatives	A list of derivatives for the model error equation.
sigmaInter	A numeric value for the constant residual error component.
sigmaSlope	A numeric value for the slope (defaulted to 0.0 for this model).
sigmaInterFixed	Logical; indicates if sigmaInter is fixed (default FALSE).
sigmaSlopeFixed	Logical; indicates if sigmaSlope is fixed (default FALSE).
cError	A numeric power parameter (default 1.0).

Value

An object of class Constant.

Slots

output character. The name of the model output.

sigmaInter numeric. The additive residual error value.

sigmaInterFixed logical. If TRUE, sigmaInter is not estimated.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

---

constraintsTableForReport

Generate Algorithm Constraints Table for Reporting

Description

Extracts and formats the design constraints and algorithm hyperparameters into a structured table suitable for inclusion in PFIM reports. Methods are available for all optimization algorithms: MultiplicativeAlgorithm, FedorovWynnAlgorithm, SimplexAlgorithm, PSOAlgorithm, and PGBOAlgorithm.

Usage

```
constraintsTableForReport(optimizationAlgorithm, ...)
```

Arguments

optimizationAlgorithm	An object of one of the PFIM optimization algorithm classes.
...	Additional arguments passed to methods.

Value

A kable object containing the formatted constraints table, listing arm-level and algorithm-level settings.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

---

convertPKModelAnalyticToPKModelODE

Conversion from analytic PK model to ODE PK model

Description

Conversion from analytic PK model to ODE PK model

conversion from analytic to ode

Arguments

pkModel	An object of class ModelInfusion that defines the model.

Value

A character string containing the ODE equation derived from the analytic expression.

Note

Copyright (c) 2026-present Romain Leroux. All rights reserved.

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Dcriterion *Dcriterion*

Description

Computes the D-criterion of the Fisher Information Matrix. The D-criterion is calculated as the determinant of the FIM raised to the power of 1 over the number of parameters.

Arguments

`fim`	An object of class `Fim`.

Value

A double giving the D-criterion of the Fim.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

defineFim *Define the Fisher Information Matrix object*

Description

This method initializes and configures the specific type of Fisher Information Matrix to be calculated within a `PFIMProject`. It maps the project's statistical assumptions (Population, Individual, or Bayesian) to the underlying FIM computational engine.

- **Population:** For Nonlinear Mixed Effects Models (NLME), accounting for inter-individual variability.
- **Individual:** For standard fixed-effects models where only one subject/profile is considered.
- **Bayesian:** When prior distributions for the parameters are incorporated into the information matrix.

Usage

```
defineFim(pfimproject, ...)
```

Arguments

`pfimproject`	An object of class `PFIMProject` containing the model and design specifications.
`...`	Additional arguments.

Value

An object of class Fim initialized with the settings defined in the project.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`defineModelAdministration`
Define the administration for an analytic model

Description

Define the administration for an analytic model

Arguments

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

Value

The model with samplings, solverInputs

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`defineModelEquationsFromLibraryOfModel` *defineModelEquationsFromLibraryOfModel*

Description

The method extracts the structural mathematical equations from the pre-defined PFIM library. This allows users to leverage standard pharmacokinetic (PK) and pharmacodynamic (PD) models without manually defining differential or algebraic equations.

Usage

```
defineModelEquationsFromLibraryOfModel(pfimproject, ...)
```

Arguments

`pfimproject`	An object of class `PFIMProject` containing the library selection criteria.
`...`	Additional arguments.

Details

This function references the `modelFromLibrary` property of the `PFIMProject`. It maps library identifiers to a specific set of symbolic or numeric equations used by the evaluation engine. Typical library models include:

- **One-compartment:** Bolus, Infusion, or First-order absorption.
- **Multi-compartment:** Distribution models with various elimination routes.
- **Standard PD:** Emax, Sigmoid Emax, or Indirect response models.

Value

A `list` of character strings or expressions representing the structural model equations.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

defineModelType *defineModelType*

Description

The method acts as a constructor for the specific model class required for analysis. It extracts configurations from a `PFIMProject` and instantiates a `Model` object, integrating equations, parameter structures, error models, and solver settings.

Usage

```
defineModelType(pfimproject, ...)
```

Arguments

`pfimproject`	An object of class `PFIMProject` containing the project specifications.
`...`	Additional arguments.

Details

This method determines whether the model should be treated as a:

- **Library Model:** Pre-defined structural models (e.g., 1-compartment PK).
- **User-Defined Model:** Custom equations provided via `modelEquations`.
- **ODE Model:** Models requiring numerical integration using specified `odeSolverParameters`.

Value

An object of class `Model` (or a subclass thereof) initialized with `modelParameters`, `odeSolverParameters`, `modelError`, and `modelEquations`.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

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

Description

define the model wrapper for the ode solver

Arguments

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

Value

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

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`defineOptimizationAlgorithm`
Define Optimization Algorithm

Description

This method initializes the selected optimization algorithm using the specific hyperparameters (such as lambda, delta, iterations) extracted from the project options. It prepares the algorithmic object for subsequent use by the `optimizeDesign` function.

Usage

```
defineOptimizationAlgorithm(optimization, ...)
```

Arguments

`optimization`	An object of class `Optimization` containing the optimization settings and design parameters.
`...`	Additional arguments.

Value

An object of class `OptimizationAlgorithm`

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`definePKModel` *Define a PK model from library of model*

Description

Define a PK model from library of model

Arguments

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

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`definePKPDModel` *Define a PKPD model from library of model*

Description

Define a PKPD model from library of model

Arguments

`pkModel`	An object of class `ModelAnalytic` that defines the PK model.
`pdModel`	An object of class `ModelAnalytic` that defines the PD model.
`pfimproject`	An object of class `PFIMProject` that defines the pfimproject.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`Design` *Design Class*

Description

The `Design` class represents a full clinical trial design. It acts as a container for multiple `Arm` objects and stores the population-level Fisher Information Matrix (FIM) and evaluation results for the entire study.

Usage

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

Arguments

`name`	A string giving the name of the design.
`size`	A numeric value representing the total number of subjects.
`arms`	A list of `Arm` objects defining the different groups.
`evaluationArms`	A list containing the evaluation results for each arm.
`numberOfArms`	An integer giving the number of arms.
`fim`	An object of class `Fim` giving the global FIM of the design.

Value

An object of class `Design`.

Slots

`name character`. The name of the design.

`size numeric`. Total number of subjects across all arms.

`arms list`. A list containing the `Arm` objects.

`numberOfArms numeric`. The count of arms in the design.

`fim Fim`. The global Fisher Information Matrix for the design.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
# 1. Define sampling times for PK and PD outcomes
samplingTimesRespPK = SamplingTimes(outcome   = "RespPK",
                                    samplings = c(0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4))

samplingTimesRespPD = SamplingTimes(outcome   = "RespPD",
                                    samplings = c(0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4))

# 2. Define the administration (Dose of 20 at t=0)
adminRespPK = Administration(outcome = "RespPK", timeDose = 0, dose = 20)

# 3. Define the study arm "0.2mg"
# Outcomes are linked to state variables: RespPK to Cc, RespPD to E.
arm02mg = Arm(name = "0.2mg",
              size = 6,
              administrations   = list(adminRespPK),
              samplingTimes     = list(samplingTimesRespPK, samplingTimesRespPD),
              initialConditions = list("Cc" = 0, "E" = 100))

# 4. Create the Design object
# The arm defined above is included in the 'arms' list.
design1 = Design(name = "Design1",
                 arms = list(arm02mg))

# Display the design summary
print(design1)
```

`Distribution` *Distribution Class*

Description

The `Distribution` class is an abstract base class used to represent statistical distributions for model parameters.

Usage

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

Arguments

name	A string specifying the distribution type.
mu	A double representing the fixed effect value.
omega	A double representing the random effect intensity.

Value

An object of class Distribution.

Slots

name character. The name of the distribution (e.g., "Normal", "LogNormal").

mu numeric. The mean value or fixed effect of the parameter.

omega numeric. The standard deviation or variance of the random effect.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

---

evaluateArm *Evaluate an arm*

Description

Evaluates the model, gradients, variance, and FIM for an arm.

Usage

```
evaluateArm(arm, model, fim, ...)
```

Arguments

arm	An object of class Arm.
model	An object of class Model.
fim	An object of class Fim.
...	Additional arguments

Value

The Arm object with updated evaluationModel, evaluationGradients, evaluationVariance, and evaluationFim.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

---

evaluateDesign *Evaluation of a clinical design*

Description

Evaluation of a clinical design

Arguments

design	An object Design to evaluate.
model	An object Model used for evaluation.
fim	An object Fim to store results.

Value

The Design object with evaluated arms and aggregated global FIM.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

---

evaluateErrorModelDerivatives *evaluate the derivatives of the model error.*

Description

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.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`evaluateFim` *Evaluate the Fim.*

Description

Computes the Fisher Information Matrix for an individual design. This method calculates the structural information (fixed effects) and the variance information (residual error components) to assemble the final FIM.

Arguments

`fim`	An object of class `IndividualFim`.
`model`	An object of class `Model` containing the structural and error parameters.
`arm`	An object of class `Arm` representing the individual's design (doses and samplings).

Value

The `IndividualFim` object populated with the calculated `fisherMatrix`.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`evaluateInitialConditions` *evaluate the initial conditions.*

Description

evaluate the initial conditions.

Arguments

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

Note

Author(s)

Romain Leroux `<romainlerouxPFIM@gmail.com>`

`evaluateModel` *Evaluate the analytic model*

Description

Evaluate the analytic model

Arguments

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

Value

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

Note

Author(s)

Romain Leroux `<romainlerouxPFIM@gmail.com>`

evaluateModelGradient *evaluate the gradient of the model*

Description

Computes the numerical gradient of the model response with respect to the structural parameters using the finite difference method.

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.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

evaluateModelVariance *Evaluate Model Variance and Sigma Derivatives*

Description

Evaluates the residual error variance of the model and computes the partial derivatives with respect to the variance parameters (σ).

Arguments

model	An object of class Model defining the structural and residual error models.
arm	An object of class Arm defining the design (sampling times and doses) for a specific group.

Value

A list containing:

- errorVariance: A numeric vector or matrix representing the evaluated residual variance.
- sigmaDerivatives: The derivatives of the variance with respect to the σ parameters.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`evaluateVarianceFIM` *Evaluate the Variance Component of the Fisher Information Matrix*

Description

The `evaluateVarianceFIM` method calculates the portion of the Fisher Information Matrix that corresponds to the variance parameters of the Nonlinear Mixed Effects Model. This includes the inter-individual variability (random effects) and the residual error components.

Arguments

`arm`	An object of class `Arm` defining the experimental design (sampling times, doses) for a group of subjects.
`model`	An object of class `Model` containing the structural equations and the statistical model for random effects.
`fim`	An object of class `PopulationFim` used as the container for the resulting matrices.

Value

A `list` containing:

- `MFVar`: A matrix representing the Fisher Information for the variance parameters.
- `V`: The computed variance-covariance matrix of the observations.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`Evaluation` *Evaluation Class*

Description

The 'Evaluation' class represents and stores all information required to evaluate a clinical trial design. It serves as the main interface for calculating the Fisher Information Matrix (FIM), Standard Errors (SE), and other design criteria.

Usage

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

Arguments

`evaluationDesign`	A list containing the evaluation results of the design.
`name`	A string representing the name of the project or evaluation study.
`modelParameters`	A list defining the fixed effects and random effects (variances) of the model.
`modelEquations`	A list containing the mathematical equations of the model.
`modelFromLibrary`	A list specifying the pre-defined model selected from the PFIM library.
`modelError`	A list specifying the residual error model (e.g., constant, proportional, or combined).
`designs`	A list of 'Design' objects representing the experimental protocols to be evaluated.
`outputs`	A list defining the observation variables or responses of the model.
`fimType`	A string specifying the FIM calculation method. Must be one of: `"population"`, `"individual"`, or `"Bayesian"`.
`odeSolverParameters`	A list containing technical settings for the ODE solver, such as `atol` and `rtol`.

Value

An object of class `Evaluation`.

Slots

`evaluationDesign list`. Stores the results of the design evaluation.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:

# Example: Evaluation of the Population Fisher Information Matrix (FIM)
# extracted from Vignette n°1.

evaluationPop = Evaluation(
  name                = "evaluation_example",
  modelEquations      = modelEquations,
  modelParameters     = modelParameters,
  modelError          = modelError,
  outputs             = list("RespPK" = "Cc", "RespPD" = "E"),
  designs             = list(design1),
  fimType             = "population",
  odeSolverParameters = list(atol = 1e-8, rtol = 1e-8)
)

# Display the results (Standard Errors, RSE, etc.)
show(evaluationPop)


## End(Not run)
```

FedorovWynnAlgorithm *FedorovWynnAlgorithm Class*

Description

The class `FedorovWynnAlgorithm` implements the FedorovWynn algorithm. The class `FedorovWynnAlgorithm` implements the Fedorov-Wynn exchange algorithm. This algorithm is used for discrete design optimization, iteratively adding or exchanging elementary protocols to maximize the determinant of the Fisher Information Matrix (D-optimality).

Usage

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

Arguments

`elementaryProtocols`
: A list of elementary protocols available for selection.

`numberOfSubjects`
: A numeric vector specifying the number of subjects per arm.

`proportionsOfSubjects`
: A numeric vector of subject proportions for each protocol.

`showProcess`
: A logical indicating whether to display optimization progress.

`FedorovWynnAlgorithmOutputs`
: A list storing the results of the optimization.

`optimisationDesign`
: A list storing the evaluations (FIM, criteria, SE) of both the initial and the optimal design.

`optimisationAlgorithmOutputs`
: A list containing the logs and algorithm-specific outputs produced during the optimization process.

`name`
: A string representing the name of the project or evaluation study.

`modelParameters`
: A list defining the fixed effects and random effects (variances) of the model.

`modelEquations`
: A list containing the mathematical equations of the model.

`modelFromLibrary`
: A list specifying the pre-defined model selected from the PFIM library.

`modelError`
: A list specifying the residual error model (e.g., constant, proportional, or combined).

designs A list of 'Design' objects representing the experimental protocols to be evaluated.

outputs A list defining the observation variables or responses of the model.

fimType A string specifying the FIM calculation method. Must be one of: "population", "individual", or "Bayesian".

odeSolverParameters
A list containing technical settings for the ODE solver, such as atol and rtol.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:

# Example from Vignette 1: Population FIM optimization using Fedorov-Wynn

# 1. Initialize the Optimization object
optiFW <- Optimization(
  name                = "FedorovWynn_Optimization",
  modelEquations      = modelEquations,
  modelParameters     = modelParameters,
  modelError          = modelError,
  optimizer           = "FedorovWynnAlgorithm",
  optimizerParameters = list(
    elementaryProtocols   = initialElementaryProtocols,
    numberOfSubjects      = numberOfSubjects,
    proportionsOfSubjects = proportionsOfSubjects,
    showProcess           = TRUE
  ),
  designs             = list(designConstraint),
  fimType             = "population",
  outputs             = list("RespPK" = "Cc", "RespPD" = "E"),
  odeSolverParameters = list(atol = 1e-8, rtol = 1e-8)
)

# 2. Run the optimization algorithm
optimizationResults = run(optiFW)

# 3. Display the optimized design and Fisher Information Matrix
show(optimizationResults)


## End(Not run)
```

FedorovWynnAlgorithm_Rcpp

FedorovWynnAlgorithm with Rcpp

Description

Implementation of the Fedorov-Wynn algorithm in C++ via Rcpp. This function handles the heavy matrix computations and exchange logic required for D-optimal design.

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	List of protocol definitions.
ndimen_input	Dimensions of the problem.
nbprot_input	Number of protocols.
numprot_input	Protocol indices.
freq_input	Frequencies of protocols.
nbdata_input	Data point counts.
vectps_input	Sampling times vector.
fisher_input	Fisher matrix inputs.
nok_input	Error code/status.
protdep_input	Initial protocol indices.
freqdep_input	Initial frequencies.

Value

A list containing optimal frequencies, sampling times, and the resulting FIM.

Fim *Fisher Information Matrix (FIM) Class*

Description

The Fim class represents the Fisher Information Matrix in the context of population pharmacokinetics and pharmacodynamics. It acts as a container for the numerical matrix and derived statistical metrics used to evaluate design performance, such as parameter precision and shrinkage.

Usage

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

Arguments

fisherMatrix	A matrix giving the numerical values of the Fim.
fixedEffects	A matrix giving the numerical values of the fixed effects of the Fim.
varianceEffects	A matrix giving the numerical values of variance effects of the Fim.
SEAndRSE	A data frame giving the calculated values of SE and RSE for parameters.
condNumberFixedEffects	The condition number of the fixed effects portion of the Fim.
condNumberVarianceEffects	The condition number of the variance effects portion of the Fim.
shrinkage	A vector giving the shrinkage values for the random effects.

Value

An object of class Fim.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`finiteDifferenceHessian` *Compute the Hessian*

Description

Prepares the necessary parameters and data structures for the computation of gradients and the Hessian matrix via the finite difference method. This includes calculating inverse column scales (`XcolsInv`), shifted parameter values, and step size fractions.

Arguments

`model`	An object of class `Model` containing the structural and error model definitions.

Value

Returns the `Model` object with the updated slot `parametersForComputingGradient`, now containing:

- `XcolsInv`: The inverse of the column scaling factors.
- `shifted`: The perturbed parameter values for finite differences.
- `frac`: The fractional step size used for the perturbations.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`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.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`fun.amoeba` *Compute the Amoeba (Nelder-Mead) Simplex Search*

Description

`fun.amoeba` is an internal numerical routine that performs the Nelder-Mead simplex search. It iteratively updates the simplex vertices to find the optimal experimental design parameters.

Usage

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

Arguments

`p`	A matrix where each row represents a vertex of the simplex.
`y`	A vector containing the function values (FIM criteria) at each vertex.
`ftol`	A numeric value specifying the fractional convergence tolerance.
`itmax`	An integer specifying the maximum number of iterations.
`funk`	The objective function to be minimized (e.g., the D-optimality criterion).
`outcomes`	The model outcomes used for FIM evaluation.
`data`	Additional data or design constraints.
`showProcess`	A logical value; if `TRUE`, logs the progress of the simplex.

Value

A list containing the optimized parameters, the function value, and the number of iterations performed.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`generateDosesCombination` *Generate dose combinations for optimization*

Description

Generate dose combinations for optimization

Arguments

`design`	An object `Design`.

Value

A list containing the combinations of doses and the total count.

Note

Author(s)

Romain Leroux `<romainlerouxPFIM@gmail.com>`

`generateFimsFromConstraints` *Generate Fisher Information Matrices (FIM) from Design Constraints*

Description

This method explores the design space defined by the project constraints and computes the Fisher Information Matrix (FIM) for every candidate elementary design arm. These matrices are then stored as a library to be used by the optimization algorithms.

Usage

```
generateFimsFromConstraints(optimization, ...)
```

Arguments

`optimization`	An object of class `Optimization` containing the model definitions, parameter values, and design constraints.
`...`	Additional arguments.

Value

The updated `optimization` object, where the slot `fisherMatrices` contains the list of matrices calculated for each candidate arm.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`generateReportEvaluation`

Generate a Comprehensive HTML Evaluation Report

Description

The `generateReportEvaluation` method compiles all computed Fisher Information Matrix (FIM) results into a standalone HTML document. This report serves as the final deliverable for a design evaluation, summarizing parameter precision, design efficiency, and numerical stability.

Arguments

`fim`	An object of class `PopulationFim` containing the finalized FIM data.
`tablesForReport`	A `list` of data frames (as returned by `tablesForReport`) containing the formatted statistical summaries.

Value

An HTML file (or a path to the generated file) containing the complete model evaluation report.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

---

generateReportOptimization

Generate the HTML Report for Design Optimization

Description

The generateReportOptimization method compiles the results of a design optimization into a professional HTML document. It specifically handles the output of the MultiplicativeAlgorithm, documenting the transition from the initial design to the optimal sampling schedule.

Arguments

fim	An object of class PopulationFim containing the Fisher Information Matrix of the final optimized design.
optimizationAlgorithm	An object of class MultiplicativeAlgorithm,FedorovWynnAlgorithm SimplexAlgorithm,PSOAlgor
tablesForReport	A list of data frames (as returned by tablesForReport) containing the final optimized statistical summaries.

Value

An HTML report file (or a path to the file) containing the detailed optimization results and diagnostic plots.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

---

generateSamplingsFromSamplingConstraints

Generate Numerical Intervals from Sampling Constraints

Description

The generateSamplingsFromSamplingConstraints method transforms a SamplingTimeConstraints object into a structured list of mathematical intervals. These intervals define the feasible search space for each optimizable sampling point.

Arguments

`samplingTimeConstraints`
: An object of class `SamplingTimeConstraints` containing the user-defined constraints and windows.

Value

A `list` named `intervalsConstraints`. Each element of the list is a numeric vector of length 2 (lower and upper bound) representing the search space for one optimizable sample.

Note

Author(s)

Romain Leroux `<romainlerouxPFIM@gmail.com>`

`generateSamplingTimesCombination` *Generate sampling time combinations*

Description

Generate sampling time combinations

Arguments

`design`
: An object `Design`.

Value

A list of possible sampling time combinations for each arm.

Note

Author(s)

Romain Leroux `<romainlerouxPFIM@gmail.com>`

getArmConstraints *Get arm constraints for optimization algorithms*

Description

Extracts administration and sampling time constraints formatted for the `MultiplicativeAlgorithm`, `FedorovWynnAlgorithm`, `SimplexAlgorithm`, `PSOAlgorithm`, or `PGBOAlgorithm`.

Usage

```
getArmConstraints(arm, optimizationAlgorithm, ...)
```

Arguments

`arm`	An object of class `Arm`.
`optimizationAlgorithm`	An object of class `MultiplicativeAlgorithm`, `FedorovWynnAlgorithm`, `SimplexAlgorithm`, `PSOAlgorithm`, or `PGBOAlgorithm`.
`...`	Additional arguments

Value

A list of constraint entries, one per sampling outcome.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

getArmData *Extract arm data for reporting*

Description

Extracts arm-level design information (name, size, outcomes, doses, sampling times) formatted for inclusion in HTML reports.

Usage

```
getArmData(arm, ...)
```

Arguments

arm	An object of class Arm.
...	Additional arguments.

Value

A list of named lists, one per sampling outcome.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

---

getCorrelationMatrix *getCorrelationMatrix*

Description

Returns the correlation matrix of parameter estimates derived from the asymptotic variance-covariance matrix $C = M^{-1}$, where M is the FIM. Formally: $R_{ij} = C_{ij} / \sqrt{C_{ii} C_{jj}}$.

Usage

```
getCorrelationMatrix(pfimproject, ...)
```

Arguments

pfimproject	An object of class PFIMProject.
...	Additional arguments.

Value

A symmetric correlation matrix with values in $[-1, 1]$ and ones on the diagonal.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:

# Extract and print the correlation matrix from the FIM evaluation results
correlationMatrix = getCorrelationMatrix(evaluationPopulationFIMResults)

# Display the matrix
print(correlationMatrix)


## End(Not run)
```

`getDcriterion` *getDcriterion: Extract the D-optimality criterion*

Description

Returns the D-criterion derived from the determinant of the FIM, normalized by the number of parameters for cross-design comparisons.

Usage

```
getDcriterion(pfimproject, ...)
```

Arguments

`pfimproject`	An object of class `PFIMProject`.
`...`	Additional arguments.

Value

A numeric value representing the D-criterion.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:

# Examples from Vignette 1 and 2

# Extract the D-criterion from the FIM evaluation results
dCriterion = getDcriterion(evaluationPopulationFIMResults)

# Display the D-criterion value
print(dCriterion)


## End(Not run)
```

`getDeterminant` *getDeterminant*

Description

Returns the determinant of the Fisher Information Matrix (FIM), a global measure of design information used for D-optimality.

Usage

```
getDeterminant(pfimproject, ...)
```

Arguments

`pfimproject`	An object of class `PFIMProject`.
`...`	Additional arguments.

Value

A numeric value representing the determinant of the FIM.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:

# Extract the determinant of the Fisher Information Matrix (FIM)
determinant = getDeterminant(evaluationPopulationFIMResults)

# Display the determinant value
print(determinant)


## End(Not run)
```

`getFisherMatrix` *getFisherMatrix*

Description

Extracts partitioned components of the FIM for an `Evaluation` object.

Usage

```
getFisherMatrix(pfimproject, ...)
```

Arguments

`pfimproject`	An object of class `Evaluation`.
`...`	Additional arguments.

Value

A `list` with `fisherMatrix`, `fixedEffects`, and `varianceEffects`.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:
fisherMatrixComponents = getFisherMatrix(evaluationPopulationFIMResults)
# Access specific matrices
FIM             = fisherMatrixComponents$fisherMatrix
fixedEffects    = fisherMatrixComponents$fixedEffects
varianceEffects = fisherMatrixComponents$varianceEffects

## End(Not run)
```

`getListLastName` *getListLastName: Get names of the deepest elements in a nested list*

Description

Recursively traverses a nested list to extract the names of the elements at the lowest level of the hierarchy.

Usage

```
getListLastName(list)
```

Arguments

`list`	A `list` (potentially nested).

Value

A `character` vector containing the names of the last elements.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

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

Description

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.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`getModelParametersData`

Extract Model Parameter Data for Reporting

Description

The `getModelParametersData` function retrieves and summarizes the properties of all parameters within a `Model` object. It compiles their statistical characteristics—including distribution types, population means, and variability—into a structured `data.frame` suitable for display or export.

Arguments

`model`	A `Model` object containing a collection of `ModelParameter` instances.

Value

A `data.frame` with the following columns:

- `Parameter`: The unique identifier of the parameter (e.g., "Cl", "V").
- `Distribution`: The statistical law applied (e.g., "Normal", "Log-Normal").
- `Mu`: The population mean value.
- `Fixed_Mu`: Logical; `TRUE` if the mean is fixed (not estimated).
- `Omega`: The inter-individual variability (IIV) value.
- `Fixed_Omega`: Logical; `TRUE` if the variance is fixed.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

getRSE *getRSE*

Description

Retrieves the Relative Standard Errors (RSE, %) from the FIM.

Usage

```
getRSE(pfimproject, ...)
```

Arguments

pfimproject	An object of class PFIMProject.
...	Additional arguments.

Value

A numeric vector of RSE (%) values for each model parameter.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:
# Extract RSE (%) for all model parameters for evaluationPopulationFIMResults
rse = getRSE(evaluationPopulationFIMResults)
# Display the RSE values
print(rse)

## End(Not run)
```

getSamplingData *Extract sampling times and maximum sampling time*

Description

Returns structured sampling information from an arm, used internally by plotting methods.

Usage

```
getSamplingData(arm, ...)
```

Arguments

arm	An object of class Arm.
...	Additional arguments.

Value

A list with samplingTimes, samplings (named list of numeric vectors), and samplingMax (numeric scalar).

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

getSE *getSE*

Description

Retrieves the Standard Errors (SE) from the Fisher Information Matrix.

Usage

```
getSE(pfimproject, ...)
```

Arguments

pfimproject	An object of class PFIMProject.
...	Additional arguments.

Value

A numeric vector of SE values for each model parameter.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:
# Extract Standard Errors from evaluationPopulationFIMResults
se = getSE(evaluationPopulationFIMResults)
print(se)

## End(Not run)
```

---

getShrinkage *getShrinkage*

Description

Retrieves shrinkage values for the random effects (omega), measuring how individual estimates are shrunk toward the population mean.

Usage

```
getShrinkage(pfimproject, ...)
```

Arguments

`pfimproject`	An object of class `PFIMProject`.
`...`	Additional arguments.

Value

A numeric vector of shrinkage values for each random effect.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:

# Extract the shrinkage values from the Bayesian FIM evaluation results
shrinkage = getShrinkage(evaluationBayesianFIMResults)
print(shrinkage)


## End(Not run)
```

`IndividualFim` *Individual Fisher Information Matrix (IndividualFim) Class*

Description

The `IndividualFim` class represents and stores the Fisher Information Matrix (FIM) calculated for a single individual's design. In contrast to a population FIM, it focuses solely on the precision of the fixed parameters (or individual parameters) without considering inter-individual variability.

Usage

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

Arguments

`fisherMatrix`	A matrix giving the numerical values of the Fim.
`fixedEffects`	A matrix giving the numerical values of the fixed effects of the Fim.
`varianceEffects`	A matrix giving the numerical values of variance effects of the Fim.
`SEAndRSE`	A data frame giving the calculated values of SE and RSE for parameters.
`condNumberFixedEffects`	The condition number of the fixed effects portion of the Fim.
`condNumberVarianceEffects`	The condition number of the variance effects portion of the Fim.
`shrinkage`	A vector giving the shrinkage values for the random effects.

Methods

`calculateDcriterion` Computes the D-optimality criterion.

`calculateEfficiency` Compares the efficiency of two individual designs.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`LibraryOfModels` *LibraryOfModels Class*

Description

The `LibraryOfModels` class is a centralized container designed to store and manage Pharmacokinetic (PK) and Pharmacodynamic (PD) model definitions.

Usage

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

Arguments

`models`	A named list containing the PK and PD model strings or objects.

Details

This class acts as a bridge between structural model definitions and the `Evaluation` engine, ensuring that PK/PD associations are correctly mapped.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

LibraryOfPDModels *LibraryOfPDModels Class*

Description

The LibraryOfPDModels class is a specialized container for managing and storing Pharmacodynamic (PD) model definitions.

Usage

```
LibraryOfPDModels
```

Format

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

Details

This class inherits from LibraryOfModels and provides a dedicated structure for pharmacodynamic responses. It is designed to handle various PD mechanisms, including direct effect models (Emax, Sigmoid Emax), indirect response models (Turnover), and kinetic-pharmacodynamic (K-PD) structures.

Slots

models A named list of PD model structures (e.g., Emax, Indirect Response).

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

LibraryOfPKModels *LibraryOfPKModels Class*

Description

The LibraryOfPKModels class is a specialized container for managing and storing Pharmacokinetic (PK) model definitions.

Usage

```
LibraryOfPKModels
```

Format

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

Details

This class inherits from `LibraryOfModels`. It is specifically optimized to handle PK-specific attributes such as absorption types (e.g., Bolus, Infusion, Zero-Order), clearance structures, and compartmental volumes.

Slots

`models` A named list of PK model structures (e.g., 1-compartment, 2-compartment).

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`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 Class*

Description

The class `LogNormal` implements the LogNormal distribution.

Usage

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

Arguments

`name`	A string specifying the distribution type.
`mu`	A double representing the fixed effect value.
`omega`	A double representing the random effect intensity.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
# Set a Log-Normal distribution for a population parameter
distribution = LogNormal(mu = 0.74, omega = 0.316)
print(distribution)
```

`Model` *Model Class*

Description

The `Model` class represents and stores all information required to define a structural model (PK, PD, or PKPD). This includes model parameters, differential equations (ODEs), and the residual error 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 `character` vector specifying the name of the model.
`modelParameters`	A `list` of objects defining the model parameters.
`samplings`	A `numeric` vector specifying the planned sampling times.
`modelEquations`	A `list` containing the system of equations (analytical or ODEs).
`wrapper`	A `function` wrapper used to interface the model (defaults to `function() NULL`).
`outputFormula`	A `list` of mathematical formulas for the model outputs.
`outputNames`	A `character` vector defining the names of the output variables.
`variableNames`	A `character` vector defining the names of the state variables.
`outcomesWithAdministration`	A `character` vector specifying outcomes associated with drug administration.
`outcomesWithNoAdministration`	A `character` vector specifying outcomes without drug administration.

`modelError` A `list` defining the residual error model structure.

`odeSolverParameters`
A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).

`parametersForComputingGradient`
A `list` of parameters required for numerical gradient computation.

`initialConditions`
A `numeric` vector specifying the initial state of the system.

`functionArguments`
A `character` vector of arguments required by the model function.

`functionArgumentsSymbol`
A `list` of symbols representing the function arguments.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`ModelAnalytic` *ModelAnalytic Class*

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(),
```

```
  wrapperModelAnalytic = list(),
  functionArgumentsModelAnalytic = list(),
  functionArgumentsSymbolModelAnalytic = list(),
  solverInputs = list()
)
```

Arguments

`name`	A `character` vector specifying the name of the model.
`modelParameters`	A `list` of objects defining the model parameters.
`samplings`	A `numeric` vector specifying the planned sampling times.
`modelEquations`	A `list` containing the system of equations (analytical or ODEs).
`wrapper`	A `function` wrapper used to interface the model (defaults to `function() NULL`).
`outputFormula`	A `list` of mathematical formulas for the model outputs.
`outputNames`	A `character` vector defining the names of the output variables.
`variableNames`	A `character` vector defining the names of the state variables.
`outcomesWithAdministration`	A `character` vector specifying outcomes associated with drug administration.
`outcomesWithNoAdministration`	A `character` vector specifying outcomes without drug administration.
`modelError`	A `list` defining the residual error model structure.
`odeSolverParameters`	A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).
`parametersForComputingGradient`	A `list` of parameters required for numerical gradient computation.
`initialConditions`	A `numeric` vector specifying the initial state of the system.
`functionArguments`	A `character` vector of arguments required by the model function.
`functionArgumentsSymbol`	A `list` of symbols representing the function arguments.
`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.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`ModelAnalyticInfusion` *ModelAnalyticInfusion Class*

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 `character` vector specifying the name of the model.
`modelParameters`	A `list` of objects defining the model parameters.
`samplings`	A `numeric` vector specifying the planned sampling times.
`modelEquations`	A `list` containing the system of equations (analytical or ODEs).
`wrapper`	A `function` wrapper used to interface the model (defaults to `function() NULL`).
`outputFormula`	A `list` of mathematical formulas for the model outputs.
`outputNames`	A `character` vector defining the names of the output variables.

`variableNames` A `character` vector defining the names of the state variables.

`outcomesWithAdministration`
A `character` vector specifying outcomes associated with drug administration.

`outcomesWithNoAdministration`
A `character` vector specifying outcomes without drug administration.

`modelError` A `list` defining the residual error model structure.

`odeSolverParameters`
A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).

`parametersForComputingGradient`
A `list` of parameters required for numerical gradient computation.

`initialConditions`
A `numeric` vector specifying the initial state of the system.

`functionArguments`
A `character` vector of arguments required by the model function.

`functionArgumentsSymbol`
A `list` of symbols representing the function arguments.

`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.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`ModelAnalyticInfusionSteadyState`
ModelAnalyticInfusionSteadyState Class

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 `character` vector specifying the name of the model.
`modelParameters`	A `list` of objects defining the model parameters.
`samplings`	A `numeric` vector specifying the planned sampling times.
`modelEquations`	A `list` containing the system of equations (analytical or ODEs).
`wrapper`	A `function` wrapper used to interface the model (defaults to `function() NULL`).
`outputFormula`	A `list` of mathematical formulas for the model outputs.
`outputNames`	A `character` vector defining the names of the output variables.
`variableNames`	A `character` vector defining the names of the state variables.
`outcomesWithAdministration`	A `character` vector specifying outcomes associated with drug administration.
`outcomesWithNoAdministration`	A `character` vector specifying outcomes without drug administration.
`modelError`	A `list` defining the residual error model structure.
`odeSolverParameters`	A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).
`parametersForComputingGradient`	A `list` of parameters required for numerical gradient computation.

`initialConditions`
: A `numeric` vector specifying the initial state of the system.

`functionArguments`
: A `character` vector of arguments required by the model function.

`functionArgumentsSymbol`
: A `list` of symbols representing the function arguments.

`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.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`ModelAnalyticSteadyState`
ModelAnalyticSteadyState Class

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(),
  wrapperModelAnalytic = list(),
  functionArgumentsModelAnalytic = list(),
  functionArgumentsSymbolModelAnalytic = list(),
  solverInputs = list()
)
```

Arguments

`name`	A `character` vector specifying the name of the model.
`modelParameters`	A `list` of objects defining the model parameters.
`samplings`	A `numeric` vector specifying the planned sampling times.
`modelEquations`	A `list` containing the system of equations (analytical or ODEs).
`wrapper`	A `function` wrapper used to interface the model (defaults to `function()` `NULL`).
`outputFormula`	A `list` of mathematical formulas for the model outputs.
`outputNames`	A `character` vector defining the names of the output variables.
`variableNames`	A `character` vector defining the names of the state variables.
`outcomesWithAdministration`	A `character` vector specifying outcomes associated with drug administration.
`outcomesWithNoAdministration`	A `character` vector specifying outcomes without drug administration.
`modelError`	A `list` defining the residual error model structure.
`odeSolverParameters`	A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).
`parametersForComputingGradient`	A `list` of parameters required for numerical gradient computation.
`initialConditions`	A `numeric` vector specifying the initial state of the system.
`functionArguments`	A `character` vector of arguments required by the model function.
`functionArgumentsSymbol`	A `list` of symbols representing the function arguments.
`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.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`ModelError` *ModelError Class*

Description

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

Usage

```
ModelError(
  output = "output",
  equation = expression(),
  derivatives = list(),
  sigmaInter = 0.1,
  sigmaSlope = 0,
  sigmaInterFixed = FALSE,
  sigmaSlopeFixed = FALSE,
  cError = 1
)
```

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.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
# 1. Define an additive error model
# sigmaInter: intercept (additive), sigmaSlope: slope (proportional)
additiveError = ModelError(
  output     = "RespPK",
  sigmaInter = 0.1,
  sigmaSlope = 0.0
)
print(additiveError)

# 2. Define a combined error model (Additive + Proportional)
combinedError = ModelError(
  output     = "RespPK",
  sigmaInter = 0.05,
  sigmaSlope = 0.15
)
print(combinedError)
```

ModelInfusion *ModelInfusion Class*

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 `character` vector specifying the name of the model.
`modelParameters`	A `list` of objects defining the model parameters.
`samplings`	A `numeric` vector specifying the planned sampling times.
`modelEquations`	A `list` containing the system of equations (analytical or ODEs).
`wrapper`	A `function` wrapper used to interface the model (defaults to `function() NULL`).
`outputFormula`	A `list` of mathematical formulas for the model outputs.
`outputNames`	A `character` vector defining the names of the output variables.
`variableNames`	A `character` vector defining the names of the state variables.
`outcomesWithAdministration`	A `character` vector specifying outcomes associated with drug administration.
`outcomesWithNoAdministration`	A `character` vector specifying outcomes without drug administration.
`modelError`	A `list` defining the residual error model structure.
`odeSolverParameters`	A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).
`parametersForComputingGradient`	A `list` of parameters required for numerical gradient computation.
`initialConditions`	A `numeric` vector specifying the initial state of the system.
`functionArguments`	A `character` vector of arguments required by the model function.
`functionArgumentsSymbol`	A `list` of symbols representing the function arguments.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`ModelODE` *ModelODE Class*

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 `character` vector specifying the name of the model.
`modelParameters`	A `list` of objects defining the model parameters.
`samplings`	A `numeric` vector specifying the planned sampling times.
`modelEquations`	A `list` containing the system of equations (analytical or ODEs).
`wrapper`	A `function` wrapper used to interface the model (defaults to `function() NULL`).
`outputFormula`	A `list` of mathematical formulas for the model outputs.
`outputNames`	A `character` vector defining the names of the output variables.
`variableNames`	A `character` vector defining the names of the state variables.
`outcomesWithAdministration`	A `character` vector specifying outcomes associated with drug administration.
`outcomesWithNoAdministration`	A `character` vector specifying outcomes without drug administration.
`modelError`	A `list` defining the residual error model structure.
`odeSolverParameters`	A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).
`parametersForComputingGradient`	A `list` of parameters required for numerical gradient computation.
`initialConditions`	A `numeric` vector specifying the initial state of the system.
`functionArguments`	A `character` vector of arguments required by the model function.
`functionArgumentsSymbol`	A `list` of symbols representing the function arguments.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`ModelODEBolus` *ModelODEBolus Class*

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 `character` vector specifying the name of the model.
`modelParameters`	A `list` of objects defining the model parameters.
`samplings`	A `numeric` vector specifying the planned sampling times.
`modelEquations`	A `list` containing the system of equations (analytical or ODEs).
`wrapper`	A `function` wrapper used to interface the model (defaults to `function() NULL`).

`outputFormula` A `list` of mathematical formulas for the model outputs.

`outputNames` A `character` vector defining the names of the output variables.

`variableNames` A `character` vector defining the names of the state variables.

`outcomesWithAdministration`
A `character` vector specifying outcomes associated with drug administration.

`outcomesWithNoAdministration`
A `character` vector specifying outcomes without drug administration.

`modelError` A `list` defining the residual error model structure.

`odeSolverParameters`
A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).

`parametersForComputingGradient`
A `list` of parameters required for numerical gradient computation.

`initialConditions`
A `numeric` vector specifying the initial state of the system.

`functionArguments`
A `character` vector of arguments required by the model function.

`functionArgumentsSymbol`
A `list` of symbols representing the function arguments.

`modelODE` An object `modelODE`.

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

`solverInputs` A list giving the solver inputs.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`ModelODEDoseInEquations`
ModelODEDoseInEquations Class

Description

The `ModelODEDoseInEquations` class is designed to define Ordinary Differential Equation (ODE) models where the dose and the time elapsed since administration are explicitly included within the system of equations (e.g., infusions, zero-order inputs, or custom input functions).

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 `character` vector specifying the name of the model.
`modelParameters`	A `list` of objects defining the model parameters.
`samplings`	A `numeric` vector specifying the planned sampling times.
`modelEquations`	A `list` containing the system of equations (analytical or ODEs).
`wrapper`	A `function` wrapper used to interface the model (defaults to `function() NULL`).
`outputFormula`	A `list` of mathematical formulas for the model outputs.
`outputNames`	A `character` vector defining the names of the output variables.
`variableNames`	A `character` vector defining the names of the state variables.
`outcomesWithAdministration`	A `character` vector specifying outcomes associated with drug administration.
`outcomesWithNoAdministration`	A `character` vector specifying outcomes without drug administration.
`modelError`	A `list` defining the residual error model structure.
`odeSolverParameters`	A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).
`parametersForComputingGradient`	A `list` of parameters required for numerical gradient computation.
`initialConditions`	A `numeric` vector specifying the initial state of the system.

`functionArguments`
: A `character` vector of arguments required by the model function.

`functionArgumentsSymbol`
: A `list` of symbols representing the function arguments.

`modelODEDoseInEquations`
: A `function` representing the ODE system, incorporating dose-related variables.

`solverInputs`
: A `list` containing solver-specific inputs, such as dose intervals, rates, and administration schedules.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`ModelODEDoseNotInEquations`
ModelODEDoseNotInEquations Class

Description

The `ModelODEDoseNotInEquations` class defines an ODE-based model where doses are handled as discrete events rather than continuous functions within the equations. This is typically used for bolus administrations where the dose results in an instantaneous change in state variables.

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 `character` vector specifying the name of the model.
`modelParameters`	A `list` of objects defining the model parameters.
`samplings`	A `numeric` vector specifying the planned sampling times.
`modelEquations`	A `list` containing the system of equations (analytical or ODEs).
`wrapper`	A `function` wrapper used to interface the model (defaults to `function() NULL`).
`outputFormula`	A `list` of mathematical formulas for the model outputs.
`outputNames`	A `character` vector defining the names of the output variables.
`variableNames`	A `character` vector defining the names of the state variables.
`outcomesWithAdministration`	A `character` vector specifying outcomes associated with drug administration.
`outcomesWithNoAdministration`	A `character` vector specifying outcomes without drug administration.
`modelError`	A `list` defining the residual error model structure.
`odeSolverParameters`	A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).
`parametersForComputingGradient`	A `list` of parameters required for numerical gradient computation.
`initialConditions`	A `numeric` vector specifying the initial state of the system.
`functionArguments`	A `character` vector of arguments required by the model function.
`functionArgumentsSymbol`	A `list` of symbols representing the function arguments.
`modelODE`	An object of class `modelODE` defining the structural differential equations.
`doseEvent`	A `data.frame` containing the event schedule (time, dose amount, compartment index) required by the ODE solver.
`solverInputs`	A `list` providing specific configurations for the numerical integrator, such as event-handling logic.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

ModelODEInfusion *ModelODEInfusion Class*

Description

The ModelODEInfusion class is specifically designed to define ODE-based models for infusion-type administrations. It extends the ModelInfusion properties to handle continuous drug delivery through differential equations.

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 character vector specifying the name of the model.
modelParameters	A list of objects defining the model parameters.
samplings	A numeric vector specifying the planned sampling times.
modelEquations	A list containing the system of equations (analytical or ODEs).
wrapper	A function wrapper used to interface the model (defaults to function() NULL).
outputFormula	A list of mathematical formulas for the model outputs.
outputNames	A character vector defining the names of the output variables.
variableNames	A character vector defining the names of the state variables.
outcomesWithAdministration	A character vector specifying outcomes associated with drug administration.
outcomesWithNoAdministration	A character vector specifying outcomes without drug administration.

`modelError` A `list` defining the residual error model structure.

`odeSolverParameters`
A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).

`parametersForComputingGradient`
A `list` of parameters required for numerical gradient computation.

`initialConditions`
A `numeric` vector specifying the initial state of the system.

`functionArguments`
A `character` vector of arguments required by the model function.

`functionArgumentsSymbol`
A `list` of symbols representing the function arguments.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`ModelODEInfusionDoseInEquation`
ModelODEInfusionDoseInEquation Class

Description

The `ModelODEInfusionDoseInEquation` class is designed for infusion models where the differential equations transition between different states depending on whether the infusion is currently active or has ended.

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 `character` vector specifying the name of the model.
`modelParameters`	A `list` of objects defining the model parameters.
`samplings`	A `numeric` vector specifying the planned sampling times.
`modelEquations`	A `list` containing the system of equations (analytical or ODEs).
`wrapper`	A `function` wrapper used to interface the model (defaults to `function() NULL`).
`outputFormula`	A `list` of mathematical formulas for the model outputs.
`outputNames`	A `character` vector defining the names of the output variables.
`variableNames`	A `character` vector defining the names of the state variables.
`outcomesWithAdministration`	A `character` vector specifying outcomes associated with drug administration.
`outcomesWithNoAdministration`	A `character` vector specifying outcomes without drug administration.
`modelError`	A `list` defining the residual error model structure.
`odeSolverParameters`	A `list` of parameters for the ODE solver (e.g., `atol`, `rtol`).
`parametersForComputingGradient`	A `list` of parameters required for numerical gradient computation.
`initialConditions`	A `numeric` vector specifying the initial state of the system.
`functionArguments`	A `character` vector of arguments required by the model function.
`functionArgumentsSymbol`	A `list` of symbols representing the function arguments.
`modelODE`	A `function` representing the complete ODE system.
`wrapperModelInfusion`	A `list` containing two distinct sets of equations: `duringInfusion` (for active drug delivery) and `afterInfusion` (for the post-infusion phase).
`solverInputs`	A `list` of pre-calculated inputs for the numerical solver, including infusion start times, doses, and durations.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`ModelParameter` *ModelParameter Class*

Description

The `ModelParameter` class defines the characteristics of a model parameter, including its identifier (name), statistical distribution (mean and variance), and the estimation status of its components.

Usage

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

Arguments

`name`	The parameter name (string).
`distribution`	A `Distribution` object.
`fixedMu`	Logical; indicates if the mean is fixed. Defaults to `FALSE`.
`fixedOmega`	Logical; indicates if the variance is fixed. Defaults to `FALSE`.

Value

An object of class `ModelParameter`.

Slots

`name character`. A unique string identifying the parameter.

`distribution Distribution`. An object of class `Distribution` defining the statistical law (e.g., Log-Normal, Normal).

`fixedMu logical`. If `TRUE`, the population mean is fixed and will not be estimated.

`fixedOmega logical`. If `TRUE`, the inter-individual variability (omega) is fixed and will not be estimated.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
# 1. Clearance with estimated mean and estimated variance (mu and omega)
clEstimated = ModelParameter(
  name         = "Cl",
  distribution = LogNormal(mu = 0.28, omega = 0.456),
  fixedMu      = FALSE,
  fixedOmega   = FALSE
)
print(clEstimated)

# 2. Clearance with fixed mean and fixed variance
# Useful for parameters known from literature (e.g., mu = log(20) approx 2.99)
clFixed = ModelParameter(
  name         = "Cl",
  distribution = LogNormal(mu = 2.99, omega = 0.1),
  fixedMu      = TRUE,
  fixedOmega   = TRUE
)
print(clFixed)
```

MultiplicativeAlgorithm

MultiplicativeAlgorithm Class

Description

The MultiplicativeAlgorithm class implements the multiplicative algorithm for the continuous optimization of study design weights. This approach iteratively updates weights to maximize a specific optimality criterion (e.g., D-optimality).

Usage

```
MultiplicativeAlgorithm(
  optimisationDesign = list(),
  optimisationAlgorithmOutputs = list(),
  name = character(0),
  modelParameters = list(),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelError = list(),
  designs = list(),
  outputs = list(),
  fimType = character(0),
  odeSolverParameters = list(),
```

```
  optimizer = character(0),
  optimizerParameters = list(),
  lambda = 0,
  delta = 0,
  numberOfIterations = 0,
  weightThreshold = 0,
  showProcess = FALSE,
  multiplicativeAlgorithmOutputs = list()
)
```

Arguments

`optimisationDesign`	A list storing the evaluations (FIM, criteria, SE) of both the initial and the optimal design.
`optimisationAlgorithmOutputs`	A list containing the logs and algorithm-specific outputs produced during the optimization process.
`name`	A string representing the name of the project or evaluation study.
`modelParameters`	A list defining the fixed effects and random effects (variances) of the model.
`modelEquations`	A list containing the mathematical equations of the model.
`modelFromLibrary`	A list specifying the pre-defined model selected from the PFIM library.
`modelError`	A list specifying the residual error model (e.g., constant, proportional, or combined).
`designs`	A list of 'Design' objects representing the experimental protocols to be evaluated.
`outputs`	A list defining the observation variables or responses of the model.
`fimType`	A string specifying the FIM calculation method. Must be one of: `"population"`, `"individual"`, or `"Bayesian"`.
`odeSolverParameters`	A list containing technical settings for the ODE solver, such as `atol` and `rtol`.
`optimizer`	A character string naming the optimization algorithm.
`optimizerParameters`	A named list of algorithm-specific hyperparameters.
`lambda`	A `numeric` value for the relaxation parameter `lambda` (typically between 0 and 1).
`delta`	A `numeric` convergence criterion `delta`.
`numberOfIterations`	Maximum `integer` number of iterations to perform.
`weightThreshold`	A `numeric` threshold; weights below this value are considered zero and effectively removed from the design.

showProcess logical; if TRUE, displays the optimization progress and convergence status in the console.

multiplicativeAlgorithmOutputs
A list storing optimization results, including weight history and optimality criterion values.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:
# Example from Vignette 1: Initializing the Multiplicative algorithm for population FIM optimization

optimizationMultPopFIM = Optimization(
  name                = "PKPD_ODE_multi_doses_populationFIM",
  modelEquations      = modelEquations,
  modelParameters     = modelParameters,
  modelError          = modelError,
  optimizer           = "MultiplicativeAlgorithm",
  optimizerParameters = list(
    lambda             = 0.99,    # near-unity: slow but stable weight updates
    numberOfIterations = 1000,    # maximum multiplicative iterations
    weightThreshold    = 0.01,    # discard protocols with weight < 1%
    delta              = 1e-04,   # stop when D-criterion improvement < 0.01%
    showProcess        = TRUE
  ),
  designs             = list(designConstraint),
  fimType             = "population",
  outputs             = list("RespPK" = "Cc", "RespPD" = "E"),
  odeSolverParameters = list(atol = 1e-8, rtol = 1e-8)
)

# Run the optimization and display results
optimizationResults = run(optimizationMultPopFIM)
show(optimizationResults)

## End(Not run)
```

MultiplicativeAlgorithm_Rcpp
Rcpp Multiplicative Algorithm for Optimal Design

Description

Executes the high-performance C++ implementation of the multiplicative algorithm. This function serves as the computational engine for the `MultiplicativeAlgorithm` class, processing Fisher Information Matrices (FIM) from multiple arms to determine the optimal weight distribution.

Usage

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

Arguments

`fisherMatrices_input`	A `list` or `vector` of flattened Fisher Information Matrices for each candidate elementary design arm.
`numberOfFisherMatrices_input`	An `integer` specifying the total number of candidate arms.
`weights_input`	A `numeric` vector of initial weights (must sum to 1).
`numberOfParameters_input`	The number of fixed parameters in the model.
`dim_input`	The dimension of the matrices (typically equal to `numberOfParameters_input`).
`lambda_input`	Relaxation parameter for weight updates (step size).
`delta_input`	Convergence threshold for the optimality criterion.
`iterationInit_input`	Maximum number of iterations allowed for the C++ solver.

Value

A `list` containing:

- `weights`: The vector of optimized design weights.
- `criterion`: The final value of the optimality criterion.
- `iterations`: The number of iterations performed.
- `convergence`: A `logical` indicating if the convergence criterion was met.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`Normal` *Normal Class*

Description

The class `Normal` implements the Normal distribution.

Usage

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

Arguments

`name`	A string specifying the distribution type.
`mu`	A double representing the fixed effect value.
`omega`	A double representing the random effect intensity.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
# Set the Normal distribution for a parameter
normalDistribution = Normal(
  mu    = 0.74,
  omega = 0.316
)
# Display distribution summary
print(normalDistribution)
```

`Optimization` *Optimization Class*

Description

The 'Optimization' class is the base class for managing design optimization processes in PFIM. It allows comparison of the performance of the initial design against the optimal design generated by the optimization algorithm.

Usage

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

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

Arguments

`optimisationDesign`
: A list storing the evaluations (FIM, criteria, SE) of both the initial and the optimal design.

`optimisationAlgorithmOutputs`
: A list containing the logs and algorithm-specific outputs produced during the optimization process.

`name`
: A string representing the name of the project or evaluation study.

`modelParameters`
: A list defining the fixed effects and random effects (variances) of the model.

`modelEquations`
: A list containing the mathematical equations of the model.

`modelFromLibrary`
: A list specifying the pre-defined model selected from the PFIM library.

`modelError`
: A list specifying the residual error model (e.g., constant, proportional, or combined).

`designs`
: A list of 'Design' objects representing the experimental protocols to be evaluated.

`outputs`
: A list defining the observation variables or responses of the model.

`fimType`
: A string specifying the FIM calculation method. Must be one of: `"population"`, `"individual"`, or `"Bayesian"`.

`odeSolverParameters`
: A list containing technical settings for the ODE solver, such as `atol` and `rtol`.

`optimizer`
: A character string naming the optimization algorithm.

`optimizerParameters`
: A named list of algorithm-specific hyperparameters.

Value

An object of class `Optimization`, or of the corresponding algorithm sub-class when `optimizer` is supplied.

Slots

`optimisationDesign list`. Stores the evaluations (FIM, criteria, SE) of both the initial and the optimal design.

`optimisationAlgorithmOutputs list`. Contains the logs and algorithm-specific outputs produced during the optimization process.

`optimizer character`. Name of the optimization algorithm to use. Must be one of `"FedorovWynnAlgorithm"`, `"MultiplicativeAlgorithm"`, `"SimplexAlgorithm"`, `"PSOAlgorithm"`, or `"PGBOAlgorithm"`. When provided, the constructor acts as a factory and returns an instance of the corresponding sub-class directly.

`optimizerParameters list`. Named list of algorithm-specific hyperparameters passed to the sub-class constructor when `optimizer` is set.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:

Examples from Vignette 1 and 2

# 1. Multiplicative algorithm for population FIM optimization
optimizationMultPopFIM = Optimization(
  name                = "PKPD_ODE_multi_doses_populationFIM",
  modelEquations      = modelEquations,
  modelParameters     = modelParameters,
  modelError          = modelError,
  optimizer           = "MultiplicativeAlgorithm",
  optimizerParameters = list(
    lambda             = 0.99,    # near-unity: slow but stable weight updates
    numberOfIterations = 1000,    # maximum multiplicative iterations
    weightThreshold    = 0.01,    # discard protocols with weight < 1%
    delta              = 1e-04,   # stop when D-criterion improvement < 0.01%
    showProcess        = TRUE
  ),
  designs             = list(designConstraint),
  fimType             = "population",
  outputs             = list("RespPK" = "Cc", "RespPD" = "E"),
  odeSolverParameters = list(atol = 1e-8, rtol = 1e-8)
)

# 2. Fedorov-Wynn algorithm for population FIM optimization
optimizationFWPopFIM = Optimization(
  name                = "PKPD_ODE_multi_doses_populationFIM",
  modelEquations      = modelEquations,
  modelParameters     = modelParameters,
  modelError          = modelError,
  optimizer           = "FedorovWynnAlgorithm",
  optimizerParameters = list(
    elementaryProtocols   = initialElementaryProtocols,
    numberOfSubjects      = numberOfSubjects,
    proportionsOfSubjects = proportionsOfSubjects,
    showProcess           = TRUE
  ),
  designs             = list(designConstraint),
  fimType             = "population",
  outputs             = list("RespPK" = "Cc", "RespPD" = "E"),
  odeSolverParameters = list(atol = 1e-8, rtol = 1e-8)
)

# 3. Particle Swarm Optimization (PSO) algorithm
optimizationPSOPopFIM = Optimization(
  name                = "optimizationExamplePSO",
  modelFromLibrary    = modelFromLibrary,
  modelParameters     = modelParameters,
  modelError          = modelError,
  optimizer           = "PSOAlgorithm",
  optimizerParameters = list(
```

```
    maxIteration               = 100,   # number of swarm update cycles
    populationSize             = 50,    # number of particles
    personalLearningCoefficient = 2.05,  # c1: attraction toward personal best
    globalLearningCoefficient   = 2.05,  # c2: attraction toward global best
    seed                       = 42,    # reproducibility
    showProcess                = FALSE  # suppress iteration-level output
  ),
  designs             = list(design2),
  fimType             = "population",
  outputs             = list("RespPK")
)

# 4. Population-Based Gradient Optimization (PGBO) algorithm
optimizationPGBOPopFIM = Optimization(
  name                = "optimizationExamplePGBO",
  modelFromLibrary    = modelFromLibrary,
  modelParameters     = modelParameters,
  modelError          = modelError,
  optimizer           = "PGBOAlgorithm",
  optimizerParameters = list(
    N              = 30,    # population of 30 candidate designs
    muteEffect     = 0.65,  # mutation amplitude (65% of window width)
    maxIteration   = 1000,  # total evolutionary steps
    purgeIteration = 200,  # reinitialize worst solutions every 200 steps
    seed           = 42,
    showProcess    = FALSE
  ),
  designs             = list(design2),
  fimType             = "population",
  outputs             = list("RespPK")
)

# 5. Nelder-Mead Simplex algorithm
optimizationSimplexPopFIM = Optimization(
  name                = "optimizationExampleSimplex",
  modelFromLibrary    = modelFromLibrary,
  modelParameters     = modelParameters,
  modelError          = modelError,
  optimizer           = "SimplexAlgorithm",
  optimizerParameters = list(
    pctInitialSimplexBuilding = 10,    # initial spread: 10% of window widths
    maxIteration              = 1000,  # max Nelder-Mead iterations
    tolerance                 = 1e-10, # convergence on relative D-criterion change
    showProcess               = FALSE
  ),
  designs             = list(design2),
  fimType             = "population",
  outputs             = list("RespPK")
)


## End(Not run)
```

optimizeDesign *Optimize Study Design*

Description

Dispatches the optimization of a study design to the appropriate algorithm. Methods are available for all PFIM optimization algorithms: MultiplicativeAlgorithm, FedorovWynnAlgorithm, SimplexAlgorithm, PSOAlgorithm, and PGBOAlgorithm.

Usage

```
optimizeDesign(optimizationObject, optimizationAlgorithm, ...)
```

Arguments

optimizationObject	An object of class Optimization containing the design space, model, and objective function settings.
optimizationAlgorithm	An object of the algorithm class to use (MultiplicativeAlgorithm, FedorovWynnAlgorithm, etc.).
...	Additional arguments passed to methods

Value

The updated optimizationObject with optimized designs, final FIM value, and algorithm-specific outputs stored in the relevant slots.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

PFIMProject *PFIMProject Class*

Description

The 'PFIMProject' class is the central orchestrator for a Population Fisher Information Matrix (PFIM) analysis. It encapsulates all necessary components for design evaluation or optimization, including structural models, statistical parameters, experimental designs, and optimization settings.

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 representing the name of the project or evaluation study.
`modelEquations`	A list containing the mathematical equations of the model.
`modelFromLibrary`	A list specifying the pre-defined model selected from the PFIM library.
`modelParameters`	A list defining the fixed effects and random effects (variances) of the model.
`modelError`	A list specifying the residual error model (e.g., constant, proportional, or combined).
`optimizer`	A string identifying the optimization algorithm to be used. Must be one of: `"MultiplicativeAlgorithm"`, `"FedorovWynnAlgorithm"`, `"SimplexAlgorithm"`, `"PSOAlgorithm"`, or `"PGBOAlgorithm"`.
`optimizerParameters`	A list of settings for the chosen optimizer (e.g., iterations, tolerance).
`outputs`	A list defining the observation variables or responses of the model.
`designs`	A list of 'Design' objects representing the experimental protocols to be evaluated.
`fimType`	A string specifying the FIM calculation method. Must be one of: `"population"`, `"individual"`, or `"Bayesian"`.
`fim`	An object of class `Fim` representing the computed Fisher Information Matrix.
`odeSolverParameters`	A list containing technical settings for the ODE solver, such as `atol` and `rtol`.

Slots

`name character`

`modelEquations list`

`modelFromLibrary list`

modelParameters list

modelError list

optimizer character

optimizerParameters list

outputs list

designs list

fimType character

fim Fim

odeSolverParameters list

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

PGBOAlgorithm *PGBOAlgorithm Class*

Description

The PGBOAlgorithm class implements a stochastic optimization routine based on population genetics principles. It is designed to navigate complex design spaces by simulating mutation, selection, and purging processes to maximize the Fisher Information Matrix (FIM) criteria.

Usage

```
PGBOAlgorithm(
  N = 30,
  muteEffect = 0.65,
  maxIteration = 1000,
  purgeIteration = 200,
  seed = 42,
  showProcess = FALSE,
  optimisationDesign = list(),
  optimisationAlgorithmOutputs = list(),
  name = character(0),
  modelParameters = list(),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelError = list(),
  designs = list(),
  outputs = list(),
```

```
  fimType = character(0),
  odeSolverParameters = list()
)
```

Arguments

N	A numeric value specifying the population size (number of individuals) per generation.
muteEffect	A numeric value (0-1) representing the mutation rate or the intensity of the genetic mutation effect.
maxIteration	An integer specifying the maximum number of generations before the algorithm terminates.
purgeIteration	An integer defining the frequency (in iterations) at which "weak" individuals are removed from the population to maintain genetic fitness.
seed	A numeric value for the random number generator to ensure reproducibility of the optimization results.
showProcess	A logical value; if TRUE, the algorithm prints progress updates and fitness scores to the console.
optimisationDesign	A list storing the evaluations (FIM, criteria, SE) of both the initial and the optimal design.
optimisationAlgorithmOutputs	A list containing the logs and algorithm-specific outputs produced during the optimization process.
name	A string representing the name of the project or evaluation study.
modelParameters	A list defining the fixed effects and random effects (variances) of the model.
modelEquations	A list containing the mathematical equations of the model.
modelFromLibrary	A list specifying the pre-defined model selected from the PFIM library.
modelError	A list specifying the residual error model (e.g., constant, proportional, or combined).
designs	A list of 'Design' objects representing the experimental protocols to be evaluated.
outputs	A list defining the observation variables or responses of the model.
fimType	A string specifying the FIM calculation method. Must be one of: "population", "individual", or "Bayesian".
odeSolverParameters	A list containing technical settings for the ODE solver, such as atol and rtol.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:

# Examples from Vignette 2

# Initializing the PGBO algorithm for population FIM optimization
optimizationPGBOPopFIM = Optimization(
  name                = "optimizationExamplePGBO",
  modelFromLibrary    = modelFromLibrary,
  modelParameters     = modelParameters,
  modelError          = modelError,
  optimizer           = "PGBOAlgorithm",
  optimizerParameters = list(
    N              = 30,    # population of 30 candidate designs
    muteEffect     = 0.65,  # mutation amplitude (65% of window width)
    maxIteration   = 1000,  # total evolutionary steps
    purgeIteration = 200,  # reinitialize worst solutions every 200 steps
    seed           = 42,
    showProcess    = FALSE
  ),
  designs             = list(design2),
  fimType             = "population",
  outputs             = list("RespPK")
)

# Run the PGBO optimization and display the results
resultsPGBOPopFIM = run(optimizationPGBOPopFIM)
show(resultsPGBOPopFIM)


## End(Not run)
```

`plot` *plot*

Description

Generates plots for `Evaluation` or `Optimization` objects and returns them as a named list. Dispatches automatically on the object class – the user never needs to know which specific plot function to call. Follows the same OO convention as `plot.lm` in base R. For any other object type, falls through to `graphics::plot`.

Usage

```
plot(pfimobject, plotOptions = list(), which = NULL,
     thresholdWeights = 0, thresholdFrequencies = 0, ...)
```

Arguments

pfimobject	An object of class Evaluation or Optimization.
plotOptions	A list of graphical options forwarded to plotEvaluation and plotSensitivityIndices. Defaults to list().
which	character or NULL. Subset of plots to compute. Partial matching supported (like plot.lm).
thresholdWeights	A numeric value for thresholding the weigth for the multiplicative algorithm.
thresholdFrequencies	A numeric value for thresholding the weigth for the FedorovWynn algorithm.

- Evaluation: any subset of c("evaluation", "sensitivityIndices", "SE", "RSE").
- Optimization: any subset of c("evaluation", "sensitivityIndices", "SE", "RSE", "weights", "frequencies").

NULL (default): all plots for the given class.

...	Forwarded to graphics::plot for non-PFIM objects.

Format

An object of class character of length 4.

Value

For PFIM objects: a named list of ggplot2 objects. For other objects: the return value of graphics::plot.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

See Also

plotEvaluation, plotSensitivityIndices, plotSE, plotRSE, plotWeights, plotFrequencies

Examples

```
## Not run:
results = run(evaluationPop)
p = plot(results,
         plotOptions = plotOptions,
         which       = c("evaluation", "sensitivityIndices", "SE", "RSE"))
p$SE

opt = run(optimizationMult)
```

```
p = plot(opt,
         plotOptions = plotOptions,
         which       = c("evaluation", "SE", "RSE", "weights"))
p$weights

## End(Not run)
```

---

plotEvaluation *plotEvaluation*

Description

Generates graphical representations of model responses based on the design and parameters defined in the PFIM project.

Usage

```
plotEvaluation(pfimproject, ...)
```

Arguments

pfimproject	An object of class PFIMProject.
...	Additional arguments.

Value

A named list of plots per design.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:
# Plot the model responses from evaluationPopulationFIMResults
plotEvaluation(evaluationPopulationFIMResults, plotOptions = plotOptions)

## End(Not run)
```

plotEvaluationResults *Plot model responses for an arm*

Description

Generates ggplot2 line plots of model responses, overlaying design sampling points in red on the secondary x-axis.

Usage

```
plotEvaluationResults(
  arm,
  evaluationModel,
  outputNames,
  samplingData,
  designName,
  plotOptions
)
```

Arguments

arm	An object of class Arm.
evaluationModel	A list of data frames from evaluateModel.
outputNames	A list of strings giving the output names.
samplingData	A list from getSamplingData.
designName	A string giving the design name.
plotOptions	A list with unitTime and unitOutcomes.

Value

A named list of ggplot objects.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

plotEvaluationSI *Plot sensitivity indices for an arm*

Description

Generates ggplot2 line plots of the partial derivatives of model responses with respect to population parameters (sensitivity indices).

Usage

```
plotEvaluationSI(
  arm,
  evaluationModelGradient,
  parametersNames,
  outputNames,
  samplingData,
  designName,
  plotOptions
)
```

Arguments

arm	An object of class Arm.
evaluationModelGradient	A list of data frames from evaluateModelGradient.
parametersNames	A character vector of parameter names.
outputNames	A list of strings giving the output names.
samplingData	A list from getSamplingData.
designName	A string giving the design name.
plotOptions	A list with unitTime and unitOutcomes.

Value

A named list of ggplot objects per output and parameter.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

plotFrequencies *plotFrequencies: visualize optimal frequencies for the Fedorov-Wynn algorithm*

Description

Generates a bar plot showing the frequencies (normalized counts) of the design arms selected by the Fedorov-Wynn algorithm. This plot identifies the support points of the optimal discrete design.

Usage

```
plotFrequencies(optimization, ...)
```

Arguments

optimization	An object of class Optimization containing Fedorov-Wynn algorithm results.
...	Additional arguments.

Value

A ggplot2 bar plot of the optimal frequencies.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:
# plot frequencie from optimizationFedorovWynnPopulationFIMObject
frequencies = plotFrequencies( optimizationFedorovWynnPopulationFIMObject )
print(frequencies)

## End(Not run)
```

plotFrequenciesFedorovWynnAlgorithm *plotFrequenciesFedorovWynnAlgorithm for the FedorovWynnAlgorithm*

Description

Generates a horizontal bar chart representing the optimal frequencies of the arms selected by the Fedorov-Wynn algorithm.

Arguments

optimization	An object of class Optimization.
optimizationAlgorithm	An object of class FedorovWynnAlgorithm.

Value

A ggplot object showing the frequency distribution.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

plotRSE *plotRSE*

Description

Bar plot of Relative Standard Errors (RSE, %) for the model parameters.

Usage

```
plotRSE(pfimproject, ...)
```

Arguments

pfimproject	An object of class PFIMProject.
...	Additional arguments.

Value

A bar plot of RSE (%) values.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:
# Extract Relative Standard Errors from evaluationPopulationFIMResults
se = getSE(evaluationPopulationFIMResults)
print(se)

## End(Not run)
```

---

plotRSEFIM *Plot Relative Standard Errors (RSE%) for Population FIM*

Description

Plot Relative Standard Errors (RSE%) for Population FIM

Arguments

`fim`	An object of class `PopulationFim` containing the calculated RSE values.
`evaluation`	An object of class `Evaluation` providing the context and parameter names for the plot.

Value

A plot object (typically `ggplot2` or `lattice`) displaying the RSE% for structural and variance parameters.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

plotSE *plotSE*

Description

Bar plot of Standard Errors (SE) for fixed effects and variance components.

Usage

```
plotSE(pfimproject, ...)
```

Arguments

pfimproject	An object of class PFIMProject.
...	Additional arguments.

Value

A bar plot of SE values.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:
# plotSE from evaluationPopulationFIMResults
plotSE( evaluationPopulationFIMResults )

## End(Not run)
```

plotSEFIM *Plot Standard Errors from the Population FIM*

Description

The plotSEFIM method generates a diagnostic bar plot representing the Standard Errors (SE) or Relative Standard Errors (RSE%) for all estimated parameters. This visualization is crucial for comparing the precision between structural parameters (fixed effects) and variance components.

Arguments

`fim`	An object of class `PopulationFim` containing the calculated SE and RSE values.
`evaluation`	An object of class `Evaluation` providing the context of the model being plotted.

Value

A `ggplot2` or base R plot object showing the bar plot of the parameter uncertainties.

Note

Author(s)

Romain Leroux `<romainlerouxPFIM@gmail.com>`

`plotSensitivityIndices` *plotSensitivityIndices*

Description

Generates sensitivity index plots (partial derivatives of model responses with respect to population parameters).

Usage

```
plotSensitivityIndices(pfimproject, ...)
```

Arguments

`pfimproject`	An object of class `PFIMProject`.
`...`	Additional arguments.

Value

A named list of sensitivity index plots per design.

Note

Author(s)

Romain Leroux `<romainlerouxPFIM@gmail.com>`

Examples

```
## Not run:
plot the sensitivity indices from evaluationPopulationFIMResults
plotSensitivityIndices( evaluationPopulationFIMResults, plotOptions = plotOptions )

## End(Not run)
```

`plotShrinkage` *Bar plot of the Bayesian shrinkage*

Description

Bar plot of the Bayesian shrinkage

Arguments

`fim`	An object of class `BayesianFim` giving the Fim.
`evaluation`	An object of class `PFIMProject` giving the evaluation.

Value

A `ggplot2` object representing the shrinkage bar plot.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`plotWeights` *Visualize the distribution of optimal design weights*

Description

Generates a bar plot showing the weights assigned to each candidate arm after the optimization process. This represents the proportion of the total population that should be allocated to each specific design.

Usage

```
plotWeights(optimization, ...)
```

Arguments

optimization	An object of class Optimization that has been optimized.
...	Additional arguments.

Value

A ggplot2 bar plot of the optimized weights.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:
# plot the weights from the optimizationMultiplicativePopulationFIMObject
weights = plotWeights( optimizationMultiplicativePopulationFIMObject )
print(weights)

## End(Not run)
```

---

plotWeightsMultiplicativeAlgorithm

Visualize Optimal Weight Distribution from Multiplicative Algorithm

---

Description

Generates a bar plot representing the optimal weights allocated to each study arm after the execution of the multiplicative algorithm. This visualization quickly highlights which arms have been retained or prioritized by the optimization process.

Arguments

optimization	An object of class Optimization containing the optimized weights.
optimizationAlgorithm	An object of class MultiplicativeAlgorithm
thresholdWeights	An numeric threshold for the weights used for the optimization.

Value

A ggplot2 graphical object representing the weights per arm.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`PopulationFim` *PopulationFim Class*

Description

The `PopulationFim` class is a child of the `Fim` class. It is specifically designed to store and manage the Fisher Information Matrix calculated for population-level analyses. Unlike individual FIMs, it incorporates the variance-covariance components of the random effects, providing a measure of the information content regarding both structural and statistical parameters.

- Determining the Standard Errors (SE) of population parameters.
- Calculating the Relative Standard Errors (RSE%).
- Evaluating and optimizing designs for clinical trials.

Usage

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

Arguments

`fisherMatrix`	A matrix giving the numerical values of the Fim.
`fixedEffects`	A matrix giving the numerical values of the fixed effects of the Fim.
`varianceEffects`	A matrix giving the numerical values of variance effects of the Fim.
`SEAndRSE`	A data frame giving the calculated values of SE and RSE for parameters.
`condNumberFixedEffects`	The condition number of the fixed effects portion of the Fim.
`condNumberVarianceEffects`	The condition number of the variance effects portion of the Fim.
`shrinkage`	A vector giving the shrinkage values for the random effects.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`processArmEvaluationResults` *Process arm evaluation for response plots*

Description

Evaluates the model on a dense time grid and generates response plots for a given arm, overlaying the design sampling points.

Usage

```
processArmEvaluationResults(arm, model, fim, designName, plotOptions, ...)
```

Arguments

`arm`	An object of class `Arm`.
`model`	An object of class `Model`.
`fim`	An object of class `Fim`.
`designName`	A string giving the name of the design.
`plotOptions`	A list with `unitTime` and `unitOutcomes`.
`...`	Additional arguments

Value

A named list of `ggplot` objects per design and arm.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

processArmEvaluationSI *Process arm evaluation for sensitivity index plots*

Description

Evaluates model gradients on a dense time grid and generates sensitivity index plots (partial derivatives of responses with respect to parameters).

Usage

```
processArmEvaluationSI(arm, model, fim, designName, plotOptions, ...)
```

Arguments

arm	An object of class Arm.
model	An object of class Model.
fim	An object of class Fim.
designName	A string giving the name of the design.
plotOptions	A list with unitTime and unitOutcomes.
...	Additional arguments

Value

A named list of ggplot objects per design, arm, output, and parameter.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`Proportional` *Proportional Class*

Description

The `Proportional` class defines a proportional residual error model, where the standard deviation of the error is proportional to the predicted value.

Usage

```
Proportional(
  output = character(0),
  equation = expression(sigmaSlope),
  derivatives = list(),
  sigmaInter = 0,
  sigmaSlope = 0,
  sigmaInterFixed = FALSE,
  sigmaSlopeFixed = FALSE,
  cError = 1
)
```

Arguments

`output`	A string specifying the name of the model output (e.g., "RespPK").
`equation`	An `expression` defining the error model relationship.
`derivatives`	A `list` containing the analytic derivatives of the error equation.
`sigmaInter`	A `numeric` specifying the additive error component (intercept).
`sigmaSlope`	A `numeric` specifying the proportional error component (slope).
`sigmaInterFixed`	A `logical` indicating if the intercept parameter is fixed.
`sigmaSlopeFixed`	A `logical` indicating if the slope parameter is fixed.
`cError`	A `numeric` representing the power parameter (typically 1.0).

Value

An object of class `Proportional`.

Slots

`output character`. The name of the model output (e.g., "RespPK").

`sigmaSlope numeric`. The proportional error component (slope).

`sigmaSlopeFixed logical`. If `TRUE`, the slope is fixed.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
# Define a proportional error model for a PK output "RespPK"
errorModelRespPK = Proportional(
  output     = "RespPK",
  sigmaSlope = 0.10
)

# Display the proportional error model summary
print(errorModelRespPK)
```

PSOAlgorithm *PSOAlgorithm Class*

Description

The PSOAlgorithm class is a subclass of Optimization that implements the PSO metaheuristic. It optimizes experimental designs by moving a "swarm" of candidate solutions (particles) through the search space.

Usage

```
PSOAlgorithm(
  maxIteration = 100,
  populationSize = 50,
  seed = 42,
  personalLearningCoefficient = 2.05,
  globalLearningCoefficient = 2.05,
  showProcess = FALSE,
  optimisationDesign = list(),
  optimisationAlgorithmOutputs = list(),
  name = character(0),
  modelParameters = list(),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelError = list(),
  designs = list(),
  outputs = list(),
  fimType = character(0),
  odeSolverParameters = list()
)
```

Arguments

`maxIteration`	An integer specifying the maximum number of iterations before the algorithm stops.
`populationSize`	An integer specifying the number of particles in the swarm. Larger populations explore the space better but increase computation time.
`seed`	A numeric value for the random number generator to ensure reproducibility of the optimization results.
`personalLearningCoefficient`	A numeric value (often denoted as c_1) that controls the "cognitive" component—how much the particle trusts its own best experience.
`globalLearningCoefficient`	A numeric value (often denoted as c_2) that controls the "social" component—how much the particle follows the swarm's best experience.
`showProcess`	A logical. If `TRUE`, the algorithm prints the current best fitness and iteration progress to the R console.
`optimisationDesign`	A list storing the evaluations (FIM, criteria, SE) of both the initial and the optimal design.
`optimisationAlgorithmOutputs`	A list containing the logs and algorithm-specific outputs produced during the optimization process.
`name`	A string representing the name of the project or evaluation study.
`modelParameters`	A list defining the fixed effects and random effects (variances) of the model.
`modelEquations`	A list containing the mathematical equations of the model.
`modelFromLibrary`	A list specifying the pre-defined model selected from the PFIM library.
`modelError`	A list specifying the residual error model (e.g., constant, proportional, or combined).
`designs`	A list of 'Design' objects representing the experimental protocols to be evaluated.
`outputs`	A list defining the observation variables or responses of the model.
`fimType`	A string specifying the FIM calculation method. Must be one of: `"population"`, `"individual"`, or `"Bayesian"`.
`odeSolverParameters`	A list containing technical settings for the ODE solver, such as `atol` and `rtol`.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:

# Examples from Vignette 2

# Initializing the PSO algorithm for population FIM optimization
optimizationPSOPopFIM = Optimization(
  name                = "optimizationExamplePSO",
  modelFromLibrary    = modelFromLibrary,
  modelParameters     = modelParameters,
  modelError          = modelError,
  optimizer           = "PSOAlgorithm",
  optimizerParameters = list(
    maxIteration               = 100,   # number of swarm update cycles
    populationSize             = 50,    # number of particles
    personalLearningCoefficient = 2.05,  # c1: attraction toward personal best
    globalLearningCoefficient   = 2.05,  # c2: attraction toward global best
    seed                       = 42,    # reproducibility
    showProcess                = FALSE  # suppress iteration-level output
  ),
  designs             = list(design2),
  fimType             = "population",
  outputs             = list("RespPK")
)

# Run the PSO optimization and display the results
resultsPSOPopFIM = run(optimizationPSOPopFIM)
show(resultsPSOPopFIM)


## End(Not run)
```

`replaceVariablesLibraryOfModels` *: replace variable in the LibraryOfModels*

Description

A utility function designed to rename or replace variable names within model strings (e.g., library equations). It ensures safe replacement by protecting reserved mathematical terms and keywords.

Usage

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

Arguments

`text`	the text
`old`	old string
`new`	new string

Value

text with new string

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`Report` *Report*

Description

Creates a detailed HTML report from the design evaluation results, including tables, matrices, and plots.

Usage

```
Report(pfimproject, ...)
```

Arguments

`pfimproject`	An object of class `PFIMProject`.
`...`	Additional arguments: outputPath, outputFile, plotOptions

Value

Generates and saves an HTML report to `outputPath/outputFile`.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:

# Examples from Vignette 1 and 2

# Generate a comprehensive HTML report for the design evaluation
Report(
  pfimproject = evaluationPopulationFIMResults,
  outputPath  = "C:/MyResults",
  outputFile  = "Design_Evaluation_Report.html",
  plotOptions = plotOptions
)


## End(Not run)
```

`run` *run*

Description

This function performs the evaluation or the optimization of a experimental design based on the settings provided in a `PFIMProject` object.

Usage

```
run(pfimproject, ...)
```

Arguments

`pfimproject`	An object of class `PFIMProject` containing the project settings, model definitions, and design parameters.
`...`	Additional arguments.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:
# Execute the evaluation process
results = run(evaluationPopObject)

## End(Not run)
```

SamplingTimeConstraints
SamplingTimeConstraints Class

Description

The SamplingTimeConstraints class defines the boundaries and rules for longitudinal sampling within a specific outcome of an experimental arm. It manages fixed sampling points, flexible windows, and minimum intervals.

Usage

```
SamplingTimeConstraints(
  outcome = character(0),
  initialSamplings = 0,
  fixedTimes = 0,
  numberOfSamplingsOptimisable = 0,
  samplingsWindows = list(),
  numberOfTimesByWindows = 0,
  minSampling = 0
)
```

Arguments

outcome	A string specifying which model output (e.g., "PK", "PD") these constraints apply to.
initialSamplings	A vector of numeric values representing the starting sampling schedule before optimization.
fixedTimes	A vector of numeric values specifying time points that cannot be moved or removed by the optimizer.
numberOfSamplingsOptimisable	A double representing the number of sampling points allowed to be modified.
samplingsWindows	A list of intervals (e.g., list(c(0,2), c(4,8))) defining the search boundaries for the optimizer.
numberOfTimesByWindows	A vector indicating how many samples are allowed within each specific window.
minSampling	A vector (or numeric) defining the minimum allowable time between two consecutive samples.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:
# --- Examples from Vignette 1 ---

# The following code illustrates how to configure sampling constraints for both
# discrete search grids (e.g., Fedorov-Wynn) and continuous optimization
# windows (e.g., PGBO or Simplex):

# 1. Discrete Grid Constraints (for Multiplicative and Fedorov-Wynn algorithms)
samplingConstraintsRespPK = SamplingTimeConstraints(
  outcome                     = "RespPK",
  initialSamplings            = c(0.25, 0.75, 1, 1.5, 2, 4, 6),
  fixedTimes                  = c(0.25, 4),
  numberOfSamplingsOptimisable = 4
)

# 2. Continuous Window Constraints (for PSO, PGBO, or Simplex algorithms)
samplingConstraintsRespPK = SamplingTimeConstraints(
  outcome               = "RespPK",
  initialSamplings      = c(1, 48, 72, 120),
  numberOfTimesByWindows = c(2, 2),
  samplingsWindows      = list(c(1, 48),
                               c(72, 120)),
  minSampling           = 5
)

## End(Not run)
```

`SamplingTimes` *SamplingTimes Class*

Description

The `SamplingTimes` class defines the specific time points at which observations are collected for a given model outcome. In multi-response models, this class allows each outcome (e.g., PK and PD) to have its own independent sampling schedule.

Usage

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

Arguments

`outcome`	A `string` specifying the name of the model output (e.g., "RespPK", "Metabolite").
`samplings`	A `numeric vector` representing the sampling schedule.

Value

An object of class SamplingTimes.

Slots

outcome character. The name of the model output (e.g., "RespPK").

samplings numeric vector. The sequence of observation time points.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
# Define a PK sampling schedule
samplingTimesRespPK = SamplingTimes(
  outcome   = "RespPK",
  samplings = c(0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4)
)

# Display the sampling schedule summary
print(samplingTimesRespPK)
```

---

setEvaluationFim *Finalize and Set Population FIM Results*

Description

The setEvaluationFim method processes the raw results from a model evaluation to populate the detailed statistical slots of a PopulationFim object. It transforms the Fisher Information Matrix into actionable metrics like Standard Errors (SE) and Relative Standard Errors (RSE).

Arguments

fim	An object of class PopulationFim to be updated.
evaluation	An object of class Evaluation containing the outputs from the structural model and error engine.

Value

The updated PopulationFim object, with the following slots populated: fisherMatrix, fixedEffects, shrinkage, condNumberFixedEffects, and SEAndRSE.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`setOptimalArms` *Set Optimal Arms for the MultiplicativeAlgorithm and FedorovWynnAlgorithm*

Description

The `setOptimalArms` method identifies and extracts the best performing experimental arms from an optimization routine. It converts the output of the `MultiplicativeAlgorithm` into a structured list of optimized experimental designs.

Arguments

`fim`	An object of class `PopulationFim` containing the Fisher Information Matrix evaluated at the optimal design points.
`optimizationAlgorithm`	An object of class `MultiplicativeAlgorithmFedorovWynnAlgorithm` representing the solver that has completed its execution. sampling schedule and dosing protocols that maximize the optimization criterion.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`setSamplingConstraintForOptimization` *Initialize sampling constraints*

Description

Initialize sampling constraints

Arguments

`design`	An object `Design`.

Value

The `Design` object with updated `samplingTimesConstraints`.

Note

Author(s)

Romain Leroux `<romainlerouxPFIM@gmail.com>`

`show` *show*

Description

Displays a formatted summary of a `PFIMProject` object including the FIM, standard errors, and design metrics.

Usage

```
show(pfimproject, ...)
```

Arguments

`pfimproject`	The object to be displayed.
`...`	Additional arguments.

Value

Invisibly prints the FIM summary to the console.

Note

Author(s)

Romain Leroux `<romainlerouxPFIM@gmail.com>`

Examples

```
## Not run:
# Show the results of the evaluationPopulationFIMResults
show(evaluationPopulationFIMResults)

## End(Not run)
```

showFIM *Display FIM Results in the R Console*

Description

The showFIM method provides a comprehensive summary of the Fisher Information Matrix (FIM) results. It prints structural and statistical metrics to the console, allowing the user to evaluate parameter precision, numerical stability, and optimization criteria for a specific design.

Arguments

fim	An object of class IndividualFim (or PopulationFim) containing the computed results.

Value

This function returns a formatted summary to the console. It invisibly returns a list containing the fisherMatrix, fixedEffects, Determinant, conditionNumbers, D-criterion, and Shrinkage.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

SimplexAlgorithm *SimplexAlgorithm Class*

Description

The SimplexAlgorithm class implements the Nelder-Mead downhill simplex method for derivative-free optimization. It is particularly robust for non-smooth objective functions in population FIM optimization.

Usage

```
SimplexAlgorithm(
  pctInitialSimplexBuilding = 10,
  maxIteration = 1000,
  tolerance = 1e-10,
  seed = 42,
  showProcess = FALSE,
  optimisationDesign = list(),
  optimisationAlgorithmOutputs = list(),
```

```
  name = character(0),
  modelParameters = list(),
  modelEquations = list(),
  modelFromLibrary = list(),
  modelError = list(),
  designs = list(),
  outputs = list(),
  fimType = character(0),
  odeSolverParameters = list()
)
```

Arguments

pctInitialSimplexBuilding	A numeric value giving the percent variation used to build the initial simplex around the starting point.
maxIteration	An integer specifying the maximum number of iterations allowed.
tolerance	A numeric value for the convergence tolerance on the FIM criterion.
seed	A numeric value for the random number generator seed (if applicable).
showProcess	A logical value; if TRUE, prints optimization progress to the console at each iteration.
optimisationDesign	A list storing the evaluations (FIM, criteria, SE) of both the initial and the optimal design.
optimisationAlgorithmOutputs	A list containing the logs and algorithm-specific outputs produced during the optimization process.
name	A string representing the name of the project or evaluation study.
modelParameters	A list defining the fixed effects and random effects (variances) of the model.
modelEquations	A list containing the mathematical equations of the model.
modelFromLibrary	A list specifying the pre-defined model selected from the PFIM library.
modelError	A list specifying the residual error model (e.g., constant, proportional, or combined).
designs	A list of 'Design' objects representing the experimental protocols to be evaluated.
outputs	A list defining the observation variables or responses of the model.
fimType	A string specifying the FIM calculation method. Must be one of: "population", "individual", or "Bayesian".
odeSolverParameters	A list containing technical settings for the ODE solver, such as atol and rtol.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Examples

```
## Not run:

# Examples from Vignette 2

# Initializing the Simplex algorithm for population FIM optimization
optimizationSimplexPopFIM = Optimization(
  name                = "optimizationExampleSimplex",
  modelFromLibrary    = modelFromLibrary,
  modelParameters     = modelParameters,
  modelError          = modelError,
  optimizer           = "SimplexAlgorithm",
  optimizerParameters = list(
    pctInitialSimplexBuilding = 10,    # initial spread: 10% of window widths
    maxIteration              = 1000,  # max Nelder-Mead iterations
    tolerance                 = 1e-10, # convergence on relative D-criterion change
    showProcess               = FALSE
  ),
  designs             = list(design2),
  fimType             = "population",
  outputs             = list("RespPK")
)

# Run the Simplex optimization and display the results
resultsSimplexPopFIM = run(optimizationSimplexPopFIM)
show(resultsSimplexPopFIM)


## End(Not run)
```

---

tablesForReport *Generate Statistical Tables for Evaluation Reports*

Description

The `tablesForReport` method aggregates the results of a PFIM analysis into three standardized tables. It provides a structured view of parameter estimates, global design criteria, and the precision of the estimation (Standard Errors and Relative Standard Errors).

Arguments

`fim`	An object of class `PopulationFim` containing the calculated Fisher Information Matrix and derived statistics.
`evaluation`	An object of class `Evaluation` providing the structural model context and output definitions.

Value

A `list` containing three data frames:

`fixedEffectsTable` Table of parameter names and values.

`FIMCriteriaTable` Summary of FIM-based optimality criteria.

`SEAndRSETable` Table of precision metrics (SE and RSE%).

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

`updateSamplingTimes` *Update sampling times for plotting*

Description

Replaces the arm's sampling times with a dense regular grid combined with the original sampling points, enabling smooth curve rendering.

Usage

```
updateSamplingTimes(arm, samplingData, ...)
```

Arguments

`arm`	An object of class `Arm`.
`samplingData`	The list output from `getSamplingData`.
`...`	Additional arguments

Value

The updated `Arm` object.

Note

Author(s)

Romain Leroux <romainlerouxPFIM@gmail.com>

Index