Package: Certara.RDarwin (via r-universe)

February 26, 2025

Title Interface for 'pyDarwin' Machine Learning Pharmacometric Model Development

Version 1.1.1

Description Utilities that support the usage of 'pyDarwin' (<https://certara.github.io/pyDarwin/>) for ease of setup and execution of a machine learning based pharmacometric model search with Certara's Non-Linear Mixed Effects (NLME) modeling engine.

Depends R (>= 3.6)

License LGPL-3

URL https://certara.github.io/R-Darwin/

Encoding UTF-8

RoxygenNote 7.3.2

Imports methods, magrittr, jsonlite

Suggests testthat (>= 3.0.0), dplyr, tidyr, tibble, knitr, rmarkdown

Config/testthat/edition 3

NeedsCompilation no

Author Michael Tomashevskiy [aut], James Craig [aut, cre], Certara USA, Inc [cph, fnd]

Maintainer James Craig <james.craig@certara.com>

Date/Publication 2025-02-25 17:50:11 UTC

Repository https://certara-jcraig.r-universe.dev

RemoteUrl https://github.com/cran/Certara.RDarwin

RemoteRef HEAD

RemoteSha 7f36cc912105841cdecd0219a08f95898384711b

Contents

`add_Covariate` *Add Covariate into PML models*

Description

Add Covariate into PML models

Usage

```
add_Covariate(
  PMLParametersSets,
  Name,
  Type = "Continuous",
  StParmNames = NULL,
  State = "Present",
  Direction = "Forward",
  Center = "None",
  Categories = c(),
  PMLStructures = NULL
)
```

Arguments

`PMLParametersSets`	A list of PML parameters sets (`PMLModels` class instance).
`Name`	A character string representing the name of the covariate to be added.
`Type`	A character specifying the type of the covariate. Possible values are: • `Continuous` A covariate can take values on a continuous scale. • `Categorical` A covariate can only take a finite number of values. • `Occasion` The associated PK parameter may vary within an individual from one event to the next, called interoccasion variability.
`StParmNames`	Character or character vector specifying names of structural parameters to which covariates should be added. Can be set to `NULL` or not specified, for such case, covariate will be added to all structural parameters.
`State`	A character string representing the presence of the covariate on the structural parameters. Possible values are: • `None` The covariate does not have an effect on any structural parameter. • `Present` The covariate has an effect on the structural parameters (the default). • `Searched` The effect of the covariate on structural parameters is searched.
`Direction`	A character string representing the direction of the Covariate. Options are `Forward`, `Backward`, `Interpolate`. Default is `Forward`. `Interpolate` is only applicable to `Type == "Continuous"`.

Center	A character string (None, Mean or Median) or numeric value representing the center of the Covariate. Default is None. Valid only if Type == "Continuous".
Categories	A numeric vector representing the categories (at least two) of the covariate. Applicable only if Type is either Occasion or Categorical. The first category is set to the reference category for categorical covariate.
PMLStructures	Character or character vector specifying names of PML structures to which the covariate will be added. For the naming covention of PMLStructures, see Details section of create_ModelPK() for PK models and create_ModelPD() for PD models.

Details

- If Covariate already exists, it will be substituted with a new instance with given properties. New covariate will have default bound omegas/thetas. The user can change thetas with modify_Theta() and omegas with modify_Omega().
- The current functionality does not support adding or modifying custom covariates that are defined within the PML code of custom model spaces.

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

list_Covariates() modify_Theta() modify_Omega()

Functions used for Covariate specification: Covariate(), create_ModelPD(), create_ModelPK(), remove_Covariate()

Examples

```
PMLParametersSets <- create_ModelPK()

PMLParametersSetsWT <-
 add_Covariate(PMLParametersSets,
               Name = "WT",
               Type = "Continuous",
               State = "Present",
               Direction = "Forward",
               Center = 70)

PMLParametersSetsWTCL <-
 add_Covariate(PMLParametersSets = PMLParametersSetsWT,
               Name = "Race",
               Type = "Categorical",
               State = "Searched",
               Direction = "Backward",
               Categories = c(1,2,3),
               StParmNames = "Cl",
               PMLStructure = "PK1IVC")
```

add_CustomSpace *Add a Custom Space*

Description

This function adds a custom space to a list of spaces.

Usage

```
add_CustomSpace(Spaces, CustomCode)
```

Arguments

Spaces	A list of existing spaces.
CustomCode	A character string containing the custom code for the new space.

Value

A list of spaces with the new custom space added.

add_StParm *Add Structural parameter into PML models Dosepoints*

Description

Add Structural parameter into PML models Dosepoints

Usage

```
add_StParm(
  PMLParametersSets,
  StParmName,
  Type = "LogNormal",
  State = "Present",
  ThetaStParm = list(),
  OmegaStParm = list(),
  Covariates = list(),
  PMLStructures = NULL,
  DosepointArgName = character()
)
```

Arguments

PMLParametersSets
: A list of PML parameters sets (PMLModels class instance).

StParmName
: Character specifying the name of the structural parameter to be added.

Type
: Character specifying the type of the structural parameter. Options are

- LogNormal The PML statement of the structural parameter will look like the following:

```
stparm(V = tvV * wt^dVdwt * exp(nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))
```

- LogNormal1 The PML statement of the structural parameter will look like the following:

```
stparm(V = (tvV + wt*dVdwt) * exp(nV + nVx0*( Occasion==0) + nVx1*( Occasion==1)))
```

- LogNormal2 The PML statement of the structural parameter will look like the following:

```
stparm(V = exp(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occason==1)))
```

- LogitNormal The PML statement of the structural parameter will look like the following:

```
stparm(V = ilogit(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))
```

- Normal The PML statement of the structural parameter will look like the following:

```
stparm(V = tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1))
```

State
: character string that indicates the presence of the structural parameter. Options are:

- None The structural parameter does not exist in the specified PMLStructures.
- Present The structural parameter exists in the specified PMLStructures (the default).
- Searched The presence of the structural parameter is searched.

ThetaStParm
: A Theta class instance inside the structural parameter. If not given, the associated Theta will be automatically created with its name set to "tv" + StParmName.

OmegaStParm
: An Omega class instance inside the structural parameter. If not given, the associated Omega will be automatically created with its name set to "n" + StParmName

Covariates
: A list of covariates (Covariate instances) that should be included in the structural parameter statement.

PMLStructures
: Character or character vector specifying names of PML structures to whichthe structural parameter will be added. For the naming convention of PMLStructures, see Details section of get_PMLParametersSets().

DosepointArgName
: Character specifying the name of the argument in the Dosepoint() instance to add/update the associated structural parameter. Options are bioavail, rate, duration, tlag. Not applicable for custom models

Details

- only special `Dosepoint()` related structural parameters could be added to built-in models (i.e. created using either `create_ModelPD()` or `create_modelPK()`. Due to ambiguity of situation when a structural parameter is added with `State == 'None'`, a warning is given for such cases.
- A structural parameter could be added to the custom model if it not presented in the model yet (as a custom or built-in structural parameter).

Value

An updated list of PML models (`PMLModels` class instance) matching the specified options.

See Also

`Dosepoint()` `list_StParms()`

Functions used for StParm specification: `StParm()`, `create_ModelPD()`, `create_ModelPK()`, `modify_StParm()`, `modify_StParmCustom()`, `remove_StParm()`

Examples

```
PMLParametersSets <-
  get_PMLParametersSets(CompartmentsNumber = c(1, 2, 3))

# add Rate structural parameter for all PMLModels
PMLParametersSetsVModDuration <-
 add_StParm(PMLParametersSets,
            StParmName = "Duration",
            ThetaStParm = Theta("tvD",
                          InitialEstimates = 2),
            OmegaStParm = Omega(Name = "nD",
                                State = "Searched"),
            DosepointArgName = "duration")
```

Covariate *Create a new Covariate object and validate it*

Description

Create a new Covariate object and validate it

Usage

```
Covariate(
  Name = character(),
  Type = "Continuous",
  StParmName = character(),
  State = "Present",
```

```
  Direction = "Forward",
  Center = "None",
  Categories = c(),
  Thetas = c(),
  Omegas = c(),
  PMLStructure = character()
)
```

Arguments

`Name`	Character specifying the name of the covariate.
`Type`	A character specifying the type of the covariate. Possible values are: • `Continuous` A covariate can take values on a continuous scale. • `Categorical` A covariate can only take a finite number of values. • `Occasion` The associated PK parameter may vary within an individual from one event to the next, called interoccasion variability.
`StParmName`	A character specifying the corresponding structural parameter name.
`State`	A character string representing the presence of the covariate on the structural parameters. Possible values are: • `None` The covariate does not have an effect on any structural parameter. • `Present` The covariate has an effect on the structural parameters (the default). • `Searched` The effect of the covariate on structural parameters is searched.
`Direction`	A character string representing the direction of the Covariate. Options are `Forward`, `Backward`, `Interpolate`. Default is `Forward`. `Interpolate` is only applicable to `Type == "Continuous"`.
`Center`	A character string (`None`, `Mean` or `Median`) or numeric value representing the center of the Covariate. Default is `None`. Valid only if `Type == "Continuous"`.
`Categories`	A numeric vector representing the categories (at least two) of the covariate. Applicable only if `Type` is either `Occasion` or `Categorical`. The first category is set to the reference category for categorical covariate.
`Thetas`	A list of Theta objects representing Thetas covariate effects. Only applicable if `Type` is either `Categorical` or `Continuous`. If `Type == "Continuous"`, one Theta corresponding to current Covariate should be presented. If `Type == "Categorical"`, thetas corresponding to each category (except the reference category) can be specified. If not given, theta(s) will be automatically generated with initial estimate set to 0.0.
`Omegas`	A list of Omega objects representing the Omegas of the inter-occasion random effects. Applicable only if `Type == "Occasion"`. The number of Omegas should be equal to the number of categories provided. If not given, Omegas will be created automatically with initial estimate set to 1.0.
`PMLStructure`	PML structure current Covariate instance belongs to.

Value

A Covariate object

See Also

Functions used for Covariate specification: add_Covariate(), create_ModelPD(), create_ModelPK(), remove_Covariate()

Examples

```
WT_Covariate <-
  Covariate(Name = "WT",
            Type = "Continuous",
            StParmName = "V",
            State = "Present",
            Direction = "Forward",
            Center = 70,
            Thetas = Theta("dVdWT", 1))

Race_Covariate <-
  Covariate(
    Name = "Race",
    Type = "Categorical",
    StParmName = "V2",
    State = "Searched",
    Direction = "Backward",
    Center = "None",
    Categories = c(1,2,3))
```

---

create_CustomSpace *Create a Custom Space*

Description

This function creates a custom space object based on the provided custom code.

Usage

```
create_CustomSpace(CustomCode = character())
```

Arguments

CustomCode	A character string containing the custom code.

Details

This function parses the provided CustomCode and extracts information related to:

- Responses/observations (observe, multi, ordinal, count, event, and LL)
- Structural parameters (stparm)
- Covariates (covariate, fcovariate and interpolate)

- Dosepoints (dosepoint and dosepoint2)
- Random effects (ranef)
- Fixed effects (fixef)
- Derivatives (deriv)
- Urine compartments (urinecpt)
- Closed Form statements (cfMicro, cfMacro and cfMacro1)
- Distributed delay statements (transit and delayInfCpt)

The extracted information is then used to create a CustomSpace object, which contains the parsed and structured representation of the custom code. An identifier is generated and used as the name of the Space.

Value

A list with one element of the class PMLModels.

`create_ModelEmax` *Get the list of objects describing the PML models by set of Emax parameters*

Description

This function provides the PML (Pharmacometric Modelling Language) Emax parameter sets based on the specified options. They are available as a list of specific S3 classes.

Usage

```
create_ModelEmax(
  Baseline = FALSE,
  Fractional = FALSE,
  Inhibitory = FALSE,
  Sigmoid = FALSE,
  ByVector = FALSE,
  ...
)
```

Arguments

Baseline	Logical indicating whether the Emax model contains a baseline response. If it is set to TRUE, the new parameter, E0, for baseline response is added to the model. Default is FALSE.
Fractional	Logical indicating whether the Emax model with baseline response is fractional. Applicable only for the Emax models with baseline response, otherwise a warning is given and current parameter is ignored. Default is FALSE.

Inhibitory	Logical indicating whether the model is inhibitory. If it is set to TRUE, the structural parameters 'EC50' and 'Emax' change to 'IC50' (concentration producing 50% of maximal inhibition) and 'Imax'. Default is FALSE.
Sigmoid	Logical indicating whether the model is sigmoidal. If it is set to TRUE, the Hill coefficient, 'Gam', is added to the model. Default is FALSE.
ByVector	Logical indicating whether each element in vectorized argument should be treated as a separate PML structure (i.e. treated as data.frame vectors), TRUE, or as parameters to obtain a pool (i.e. expanded) of PML structures, FALSE. Default is FALSE (one value for a function call).
...	Additional named arguments, including Structural parameters (StParm), Covariates, Dosepoints (for PK models), Thetas and Omegas. See 'Additional arguments' section.

Value

A list of PML models (PMLModels class instance) matching the specified options.

Examples

```
# Get Emax model set with default options
PDParametersSets <- create_ModelEmax()

# Create Emax model set with all possible combinations
# will give a warning since When 'Baseline == FALSE',
# there could be no model with 'Fractional == TRUE'
PDParametersSets <-
  create_ModelEmax(Baseline = TRUE,
                   Fractional = c(FALSE, TRUE),
                   Inhibitory = c(FALSE, TRUE),
                   Sigmoid = c(FALSE, TRUE),
                   ByVector = FALSE)
```

---

create_ModelPD *Get the list of objects describing the PML models by set of PD parameters*

Description

This function provides the PML (Pharmacometric Modelling Language) PD parameter sets based on the specified options. They are available as a list of specific S3 classes.

Usage

```
create_ModelPD(
  Type = "Emax",
  Baseline = FALSE,
  Fractional = FALSE,
```

```
  Inhibitory = FALSE,
  Sigmoid = FALSE,
  ByVector = FALSE,
  ...
)
```

Arguments

Type	Pharmacodynamic model type. Currently, only Emax is supported.
Baseline	Logical indicating whether the Emax model contains a baseline response. If it is set to TRUE, the new parameter, E0, for baseline response is added to the model. Default is FALSE.
Fractional	Logical indicating whether the Emax model with baseline response is fractional. Applicable only for the Emax models with baseline response, otherwise a warning is given and current parameter is ignored. Default is FALSE.
Inhibitory	Logical indicating whether the model is inhibitory. If it is set to TRUE, the structural parameters 'EC50' and 'Emax' change to 'IC50' (concentration producing 50% of maximal inhibition) and 'Imax'. Default is FALSE.
Sigmoid	Logical indicating whether the model is sigmoidal. If it is set to TRUE, the Hill coefficient, 'Gam', is added to the model. Default is FALSE.
ByVector	Logical indicating whether each element in vectorized argument should be treated as a separate PML structure (i.e. treated as data.frame vectors), TRUE, or as parameters to obtain a pool (i.e. expanded) of PML structures, FALSE. Default is FALSE (one value for a function call).
...	Additional named arguments, including Structural parameters (StParm), Covariates, Dosepoints (for PK models), Thetas and Omegas. See 'Additional arguments' section.

Details

The names of PMLStructure are constructed by the following parts:

- Baseline if presented (abbreviated as 'E0'),
- Fractional if presented (abbreviated as '1+'),
- Inhibitory (abbreviated as 'Imax' if the model is inhibitory and 'Emax' otherwise),
- Sigmoid if presented (abbreviated as 'Gam').

Value

A list of PML models (PMLModels class instance) matching the specified options.

Additional arguments

Additional arguments (ellipsis) will be applied sequentially. They can be used to add or modify Structural parameters (StParm), Covariates, Observations, Dosepoints (for PK models); by the way it is advised to use specific functions for it (see 'See Also' section for the references). Also it is possible to modify Omegas and Thetas, but it is impossible to add them (they are parts of other

structures). If PMLStructure argument is not specified, class instances will be modified or added in all PML structures. If PMLStructure argument is specified, class instances in the specified PML structure will be modified/added. Note that only one PML structure could be added to the class instance. If multiple structures should be modified, suggest to use specific functions.

See Also

Functions used for StParm specification: StParm(), add_StParm(), create_ModelPK(), modify_StParm(), modify_StParmCustom(), remove_StParm()

Functions used for Observation specification: Observation(), ObservationCustom(), Sigmas(), create_ModelPK(), modify_Observation(), remove_Observation()

Functions used for Omega specification: Omega(), create_ModelPK(), modify_Omega()

Functions used for Theta specification: InitialEstimate(), Theta(), create_ModelPK(), modify_Theta()

Functions used for Covariate specification: Covariate(), add_Covariate(), create_ModelPK(), remove_Covariate()

Examples

```
# Get PD model set with default options
PDParametersSets <- create_ModelPD(Type = "Emax")

# Create PD model set with all possible combinations
# will give a warning since When 'Baseline == FALSE',
# there could be no model with 'Fractional == TRUE'
PDParametersSets <-
  create_ModelPD(Type = "Emax",
                 Baseline = FALSE,
                 Inhibitory = c(FALSE, TRUE),
                 Sigmoid = c(FALSE, TRUE),
                 ByVector = FALSE)
```

---

create_ModelPK *Get the list of objects describing the PML models by set of PK parameters*

Description

This function provides the PML (Pharmacometric Modelling Language) PK parameter sets based on the specified options. They are available as a list of specific S3 classes.

Usage

```
create_ModelPK(
  CompartmentsNumber = 1,
  Absorption = "Intravenous",
  Parameterization = "Clearance",
```

```
  Saturation = FALSE,
  EliminationCpt = FALSE,
  FractionExcreted = FALSE,
  ByVector = FALSE,
  ClosedForm = TRUE,
  ...
)

get_PMLParametersSets(
  CompartmentsNumber = 1,
  Absorption = "Intravenous",
  Parameterization = "Clearance",
  Saturation = FALSE,
  EliminationCpt = FALSE,
  FractionExcreted = FALSE,
  ByVector = FALSE,
  ClosedForm = TRUE,
  ...
)
```

Arguments

`CompartmentsNumber`
: The number of compartments in the model. Supported embedded models are 1-, 2-, 3-compartments. Default is 1.

`Absorption`
: The absorption type of the model. Supported types are:

  - `Intravenous` (Default) - Dose is given in the main compartment (A1) directly.
  - `First-Order` - Dose is absorbed to the main compartment (A1) from the absorption compartment (Aa) by first-order kinetic.
  - `Gamma` - Dose is absorbed to A1 by Gamma Distributed delay kinetic.
  - `Inverse Gaussian` - Dose is absorbed to A1 by Inverse Gaussian Distributed delay kinetic.
  - `Weibull` - Dose is absorbed to A1 by Weibull Distributed delay kinetic.

`Parameterization`
: The parameterization type. Possible options are `Clearance` - Clearance parameters: Cl, Cl2 to be used and `Micro` - Micro parameters: Ke, K12, K21 to be used. Default is `Clearance`.

`Saturation`
: Logical indicating whether saturation should be considered. Default is `FALSE`.

`EliminationCpt`
: Logical indicating whether elimination compartment should be included. Default is `FALSE`.

`FractionExcreted`
: Logical indicating whether fraction excreted structural parameter should be included in urinecpt statement: `urinecpt(A0 = Cl * C, fe=Fe)`. Valid only if `EliminationCpt == TRUE`. Default is `FALSE`.

ByVector	Logical indicating whether each element in vectorized argument should be treated as a separate PML structure (i.e. treated as data.frame vectors), TRUE, or as parameters to obtain a pool (i.e. expanded) of PML structures, FALSE. Default is FALSE (one value for a function call).
ClosedForm	Logical indicating whether closed forms (cfMicro) should be used when possible. Note that closed forms are not available for the models with elimination compartment, models with saturation or absorption types other than Intravenous or First-Order. The models with interpolated covariates must use ClosedForm == FALSE. Default is TRUE (one value for a function call).
...	Additional named arguments, including Structural parameters (StParm), Covariates, Dosepoints (for PK models), Thetas and Omegas. See 'Additional arguments' section.

Details

The names of PMLStructure are constructed by the following parts:

- Model type ('PK'),
- Compartments number
- Abbreviated absorption type:
  - 'IV' for Intravenous,
  - 'FO' for First-Order,
  - 'G' for Gamma,
  - 'W' for Weibull,
  - 'IG' for Inverse Gaussian,
- Abbreviated parameterization ('C' for Clearance and 'M' for Micro),
- Abbreviated saturation if presented ('S'),
- Abbreviated elimination if presented ('E'),
- Abbreviated fraction excreted if presented ('F').

Value

A list of PML models (PMLModels class instance) matching the specified options.

Additional arguments

Additional arguments (ellipsis) will be applied sequentially. They can be used to add or modify Structural parameters (StParm), Covariates, Observations, Dosepoints (for PK models); by the way it is advised to use specific functions for it (see 'See Also' section for the references). Also it is possible to modify Omegas and Thetas, but it is impossible to add them (they are parts of other structures). If PMLStructure argument is not specified, class instances will be modified or added in all PML structures. If PMLStructure argument is specified, class instances in the specified PML structure will be modified/added. Note that only one PML structure could be added to the class instance. If multiple structures should be modified, suggest to use specific functions.

See Also

Functions used for Dosepoint specification: Dosepoint(), modify_Dosepoint()

Functions used for StParm specification: StParm(), add_StParm(), create_ModelPD(), modify_StParm(), modify_StParmCustom(), remove_StParm()

Functions used for Observation specification: Observation(), ObservationCustom(), Sigmas(), create_ModelPD(), modify_Observation(), remove_Observation()

Functions used for Omega specification: Omega(), create_ModelPD(), modify_Omega()

Functions used for Theta specification: InitialEstimate(), Theta(), create_ModelPD(), modify_Theta()

Functions used for Covariate specification: Covariate(), add_Covariate(), create_ModelPD(), remove_Covariate()

Examples

```
# Get PK model set with default options
PMLParametersSets <- create_ModelPK()

#' # Get PK Model search with custom options:
# will create 2 PML Parameters Sets with 2 and 3 compartments,
# with Absorption First-Order and Gamma accordingly:
ModelPKSearch <-
  create_ModelPK(CompartmentsNumber = c(2, 3),
                 Parameterization = "Micro",
                 Absorption = c("First-Order", "Gamma"),
                 ByVector = TRUE,
                 ClosedForm = TRUE)


# Next example will create a set of 4 PMLParametersSets:
# a combination of models with 2 and 3 compartments and First-Order and Gamma Absorption
PMLParametersSets <-
  create_ModelPK(CompartmentsNumber = c(2, 3),
                 Absorption = c("First-Order", "Gamma"),
                 ByVector = FALSE,
                 ClosedForm = FALSE)

# Create 2 PML Parameters Sets with elimination compartment and fraction excreted
# and add zero order absorption to the main dosepoint of the PML Structure
# with infusion
PMLParametersSets <-
  create_ModelPK(CompartmentsNumber = 1,
                 Absorption = c("Intravenous", "Gamma"),
                 EliminationCpt = TRUE,
                 FractionExcreted = TRUE,
                 duration = StParm(StParmName = "Duration",
                                   OmegaStParm = Omega(State = "None")),
                                   PMLStructure = "PK1IVCEF")

# Create 4 PML Parameters Sets, then modify `Cl` structural parameter for all sets,
# with 2 initial estimates sets to be searched,
# add `tlag` as a structural parameter `Tlag` to 1 compartment First-Order PML parameters set,
```

```
# change `tvKa` Theta initial estimate,
# change `nV` Omega initial estimate,
# change `CObs` Observation sigmas,
# add structural parameter `Rate` for 1 compartment Weibull Parameters set,
# add `Weight` covariate for all structural parameters to be searched.

PMLParametersSets <-
   create_ModelPK(
     CompartmentsNumber = 1,
     Absorption = c("First-Order", "Weibull"),
     ByVector = FALSE,
     Cl = StParm(
       StParmName = "Cl",
       Type = "LogNormal2",
       ThetaStParm =
         Theta(Name = "tvCl",
               InitialEstimates =
                 InitialEstimate(c(-Inf, 0.2, Inf),
                                 c(0, 3, 10)))
     ),
     tlag = StParm(
       StParmName = "Tlag",
       State = "Searched",
       PMLStructure = "PK1FOC",
       Covariates = list(
         Age = Covariate(
           Name = "Age",
           Type = "Categorical",
           State = "Searched",
           Direction = "Backward",
           Center = "None",
           Categories = c(1, 2, 3)
         )
       )
     ),
     tvKa = Theta(Name = "tvKa", InitialEstimates = 10),
     nV = Omega(Name = "nV", InitialOmega = 0.1),
     CObs = Observation(
       ObservationName = "CObs",
       SigmasChosen = list(
         AdditiveMultiplicative = c(PropPart = 0.1, AddPart = 2),
         Proportional = 1
       )
     ),
     A1 = Dosepoint(
       DosepointName = "A1",
       rate = StParm(StParmName = "Rate"),
       PMLStructure = "PK1WC"
     ),
     Weight = Covariate(
       Name = "Weight",
       State = "Searched",
       Center = "Median"
```

```
    )
  )
```

`create_pyDarwinOptions` *Create pyDarwin Options*

Description

Generates a list of parameters to be used in a pyDarwin run.

Usage

```
create_pyDarwinOptions(
  author = "",
  project_name = NULL,
  algorithm = c("GA", "EX", "GP", "RF", "GBRT", "PSO"),
  GA = pyDarwinOptionsGA(),
  PSO = pyDarwinOptionsPSO(),
  random_seed = 11,
  num_parallel = 4,
  num_generations = 6,
  population_size = 4,
  num_opt_chains = 4,
  exhaustive_batch_size = 100,
  crash_value = 99999999,
  penalty = pyDarwinOptionsPenalty(),
  downhill_period = 2,
  num_niches = 2,
  niche_radius = 2,
  local_2_bit_search = TRUE,
  final_downhill_search = TRUE,
  search_omega_blocks = FALSE,
  search_omega_bands = FALSE,
  individual_omega_search = TRUE,
  search_omega_sub_matrix = FALSE,
  max_omega_sub_matrix = 4,
  model_run_timeout = 1200,
  model_run_priority_class = c("below_normal", "normal"),
  postprocess = pyDarwinOptionsPostprocess(),
  keep_key_models = TRUE,
  use_saved_models = FALSE,
  saved_models_file = "{working_dir}/models0.json",
  saved_models_readonly = FALSE,
  remove_run_dir = FALSE,
  remove_temp_dir = TRUE,
```

```
  use_system_options = TRUE,
  model_cache = "darwin.MemoryModelCache",
  model_run_man = c("darwin.LocalRunManager", "darwin.GridRunManager"),
  engine_adapter = c("nlme", "nonmem"),
  working_dir = NULL,
  data_dir = NULL,
  output_dir = "{working_dir}/output",
  temp_dir = NULL,
  nlme_dir = "C:/Program Files/Certara/NLME_Engine",
  gcc_dir = "C:/Program Files/Certara/mingw64",
  nmfe_path = NULL,
  rscript_path = file.path(R.home("bin"), "Rscript"),
  nlme_license = NULL,
  generic_grid_adapter = pyDarwinOptionsGridAdapter(),
  ...
)
```

Arguments

author	Character string: The name of the author.
project_name	Character string (optional): The name of the project. If not specified, pyDarwin will set its value to the name of the parent folder of the options file.
algorithm	Character string: One of EX, GA, GP, RF, GBRT, PSO. See section Details below for more information.
GA	List: Options specific to the Genetic Algorithm (GA). See pyDarwinOptionsGA().
PSO	List: Options specific to the Particle Swarm Optimization (PSO). See pyDarwinOptionsPSO().
random_seed	Positive integer: Seed for random number generation.
num_parallel	Positive integer: Number of models to execute in parallel, i.e., how many threads to create to handle model runs.
num_generations	Positive integer: Number of iterations or generations of the search algorithm to run. Not used/required for EX.
population_size	Positive integer: Number of models to create in every generation. Not used/required for EX.
num_opt_chains	Positive integer: Number of parallel processes to perform the "ask" step (to increase performance). Required only for GP, RF, and GBRT.
exhaustive_batch_size	Positive integer: Batch size for the EX (Exhaustive Search) algorithm.
crash_value	Positive real: Value of fitness or reward assigned when model output is not generated. Should be set larger than any anticipated completed model fitness.
penalty	List: Options specific to the penalty calculation. See pyDarwinOptionsPenalty().
downhill_period	Integer: How often to run the downhill step. If < 1, no periodic downhill search will be performed.

num_niches
: Integer: Used for GA and downhill. A penalty is assigned for each model based on the number of similar models within a niche radius. This penalty is applied only to the selection process (not to the fitness of the model). The purpose is to ensure maintaining a degree of diversity in the population (integer). num_niches is also used to select the number of models that are entered into the downhill step for all algorithms, except EX.

niche_radius
: Positive real: The radius of the niches. The niche radius is used to define how similar pairs of models are. This is used to select models for the Local search, as requested, and to calculate the sharing penalty for Genetic Algorithm.

local_2_bit_search
: Logical: Whether to perform the two-bit local search. The two-bit local search substantially increases the robustness of the search. All downhill local searches are done starting from num_niches models.

final_downhill_search
: Logical: Whether to perform a local search (1-bit and 2-bit) at the end of the global search.

search_omega_blocks
: Logical: whether to perform search for block omegas. Used only when engine_adapter == 'nlme'.

search_omega_bands
: Logical: whether to perform search for band omegas. Used only when engine_adapter == 'nonmem'.

individual_omega_search
: Logical: If set, every search block will be handled individually: each block will have a separate gene and max omega search length (either calculated or set explicitly in the options). If set to FALSE, all search blocks will have the same pattern of block omegas. Default is TRUE.

search_omega_sub_matrix
: Logical: set to true to search omega submatrix. Default is FALSE.

max_omega_sub_matrix
: Integer: Maximum size of sub matrix to use in search. Default is 4.

model_run_timeout
: Positive real: Time (seconds) after which the execution will be terminated, and the crash value assigned.

model_run_priority_class
: Character string (Windows only): Priority class for child processes that build and run models, as well as run the R postprocess script. Options are below_normal and normal. below_normal is recommended to maintain user interface responsiveness.

postprocess
: List: Options specific to postprocessing. See pyDarwinOptionsPostprocess()

keep_key_models
: Logical: Key model is the best model in population (generation). Such models may be a subject of interest when the search is analyzed, so they should be saved separately with all their output. Default is TRUE

use_saved_models
: Logical: Whether to restore saved Model Cache from file. Default is FALSE.

saved_models_file
: Character string: The file from which to restore Model Cache. Will only have an effect if use_saved_models is set to true. By default, the cache is saved in {working_dir}/models.json and cleared every time the search is started. To use saved runs, rename models.json or copy it to a different location.

saved_models_readonly
: Logical: Do not overwrite the saved_models_file content. Default is FALSE.

remove_run_dir
: Logical: If TRUE, will delete the entire model run directory, otherwise - only unnecessary files inside it. Default is FALSE.

remove_temp_dir
: Logical: Whether to delete the entire temp_dir after the search is finished or stopped. Doesn't have any effect when the search is run on a grid. Default is TRUE.

use_system_options
: Logical: Whether to override options with environment-specific values. Default is TRUE.

model_cache
: Character string: ModelCache subclass to be used. Currently, there are only darwin.MemoryModelCache and darwin.AsyncMemoryModelCache. You can create your own and use it (e.g., a cache that stores model runs in a database). The name is quite arbitrary and doesn't have any convention/constraints.

model_run_man
: Character string: ModelRunManager subclass to be used. Currently, there are only darwin.LocalRunManager and darwin.GridRunManager.

engine_adapter
: Character string: ModelEngineAdapter subclass to be used. Currently only nlme (default) and nonmem are available.

working_dir
: Character string: The project's working directory, where all the necessary files and folders are created. By default, it is set to <pyDarwin home>/{project_stem}, where {project_stem} is a file system friendly representation of the project name in a way that it will be easy to manage as a folder name where all non-letters and non-digits are replaced with underscores.

data_dir
: Character string: Directory where datasets are located. Must be available for individual model runs. Default in pyDarwin if not given: {project_dir}.

output_dir
: Character string: Directory where pyDarwin output will be placed. Default is {working_dir}/output.

temp_dir
: Character string: Parent directory for all model runs' run directories, i.e., where all folders for every iteration are located. Default in pyDarwin if not given: {working_dir}/temp.

nlme_dir
: Character string: Directory where the NLME Engine is installed/unzipped. Default: C:/Program Files/Certara/NLME_Engine. Used only when engine_adapter == 'nlme'.

gcc_dir
: Character string: Directory where the Mingw-w64 compiler (gcc) is installed. Default: C:/Program Files/Certara/mingw64 for Windows and gcc version found by which gcc on Linux. Used only when engine_adapter == 'nlme'.

nmfe_path
: Character string: Directory where NONMEM is installed. Used only when engine_adapter == 'nonmem'.

`rscript_path` Character string: Path to the Rscript executable. By default, it is obtained with R.home("bin").

`nlme_license` Character string (optional): Path to the license file. If not provided, pyDarwin will set its value to PhoenixLicenseFile (only for current Python session).

`generic_grid_adapter` List: Options specific to the grids. See `pyDarwinOptionsGridAdapter()`

`...` Additional parameters: Other arguments not explicitly defined in the function's signature are allowed and will be stored in the options list. See pyDarwin documentation.

Details

The algorithm parameter specifies the type of search algorithm to be used in the pyDarwin optimization process. It determines the strategy and approach used to explore the search space and find the optimal solution. The following are the available options for the algorithm parameter.

"EX" (Exhaustive Search Algorithm): The exhaustive search algorithm is a simple and straightforward method to explore the entire search space systematically. The search space is initially represented as a string of integers, one for each dimension. The algorithm exhaustively evaluates all candidate models within the search space, making it best suited for small search spaces with a limited number of dimensions. Due to its exhaustive nature, it is not practical for large search spaces with millions of possible models.

"GA" (Genetic Algorithm): The genetic algorithm is an evolutionary optimization technique inspired by natural selection and genetics. It employs techniques such as selection, crossover, and mutation to evolve a population of candidate models over multiple generations. By applying natural selection principles, the genetic algorithm aims to converge towards better-performing models. It is suitable for moderate to large search spaces and can handle a diverse range of problem types.

"GP" (Gaussian Process Algorithm): The Gaussian Process is one of the two options used in Bayesian Optimization. It specifies the form of the prior and posterior distribution for model evaluations. Initially, the distribution is random, similar to other global search algorithms. As models are executed and their results are obtained, the distribution is updated using the "ask" and "tell" steps. The Gaussian Process aims to use probabilistic models to guide the search towards promising regions of the search space efficiently. It is particularly useful for expensive-to-evaluate functions and can handle both continuous and discrete variables.

"RF" (Random Forest Algorithm): The Random Forest algorithm is an ensemble learning method that constructs multiple decision trees during the optimization process. It leverages bagging and random feature selection to increase the precision of tree building. By combining multiple trees, the Random Forest aims to achieve higher accuracy and robustness in the optimization process. It is effective for a wide range of problem types and can handle both regression and classification tasks.

"GBRT" (Gradient Boosted Random Tree Algorithm): The Gradient Boosted Random Tree algorithm is a variation of the Random Forest approach. It builds trees progressively by calculating the gradient of the reward or fitness with respect to each decision. This allows the algorithm to focus on challenging regions of the search space, which can lead to increased precision and improved optimization results. Similar to Random Forest, it is suitable for regression and classification problems.

"PSO" (Particle Swarm Optimization Algorithm): The Particle Swarm Optimization algorithm is a population-based optimization technique inspired by the social behavior of bird flocks or fish schools. It represents potential solutions as particles that move through the search space to find

the optimal solution. Particles communicate and share information about their current best-known positions, allowing them to explore promising areas collaboratively. The Particle Swarm Optimization is effective for continuous optimization problems and can handle noisy or multimodal objective functions.

When using the create_pyDarwinOptions function, you can specify one of these algorithm values to choose the appropriate optimization strategy for your specific problem. Each algorithm has its strengths and limitations, and the choice of algorithm should be based on the problem's characteristics and the desired search space exploration behavior.

Please see pyDarwin documentation for more details.#'

Value

A list of pyDarwin options.

Examples

```
# Create pyDarwin options with default values
pyDarwinOptions <- create_pyDarwinOptions()
# Create pyDarwin options with custom author and algorithm
pyDarwinOptions <-
  create_pyDarwinOptions(author = "John Doe",
                         algorithm = "PSO")
```

`Dosepoint` *Create a new Dosepoint object and validate it*

Description

Create a new Dosepoint object and validate it

Usage

```
Dosepoint(
  DosepointName = "A1",
  State = "Present",
  tlag = c(),
  bioavail = c(),
  duration = c(),
  rate = c(),
  PMLStructure = character()
)
```

Arguments

DosepointName	A character string giving the name of the Dosepoint.
State	A character string representing the state of the Dosepoint. Possible values are: • None: current Dosepoint is not used. • Present (the default): current Dosepoint is used as is. • Searched: current Dosepoint is added as a token to be searched.
tlag	An optional structural parameter giving the time lag for the doses coming into current Dosepoint.
bioavail	An optional structural parameter giving the bioavailability of the doses coming into current Dosepoint.
duration	An optional structural parameter giving the duration of infusion for the doses coming into current Dosepoint.
rate	An optional structural parameter giving the rate of infusion for the doses coming into current Dosepoint.
PMLStructure	A character string that indicates bounded PML structure.

Value

A new Dosepoint object

See Also

list_Dosepoints()

Functions used for Dosepoint specification: create_ModelPK(), modify_Dosepoint()

Examples

```
TlagStParm <- StParm("Tlag",
                     Type = "LogNormal",
                     ThetaStParm = Theta(Name = "tvTlag",
                                         InitialEstimates = 0.1))

A1 <- Dosepoint(DosepointName = "A1",
                State = "Present",
                tlag = TlagStParm,
                bioavail = StParm("F"))
```

get_ModelTermsToMap *Get Model Terms to Map*

Description

This function retrieves the model terms that can be mapped from a set of PML models.

Usage

```
get_ModelTermsToMap(PMLParametersSets)
```

Arguments

PMLParametersSets	An object of class "PMLModels" containing PML model parameters.

Value

A list with two elements: "Required" and "Optional," representing the model terms that can be mapped.

See Also

create_ModelPK() create_ModelPD() create_CustomSpace()

Examples

```
# Load your PMLModels object
PMLParametersSets <-
  create_ModelPK(
    Absorption = c("First-Order", "Weibull"),
    CObs = Observation(
      ObservationName = "CObs",
      BQL = TRUE),
    A1 = Dosepoint(
      DosepointName = "A1",
      rate = StParm(StParmName = "Rate")),
    Weight = Covariate(
      Name = "Weight",
      Center = "Median")
)

# Get the model terms to map
terms_to_map <- get_ModelTermsToMap(PMLParametersSets)
print(terms_to_map$Required)
print(terms_to_map$Optional)
```

InitialEstimate *Create an object of class InitialEstimate*

Description

This function creates an object of class `InitialEstimate` that contains initial parameter estimates for a model `Theta()`. The estimates can be passed to the function as a single numeric value or as a vector of length three containing lower bound, estimate, and upper bound. If multiple sets of estimates are required, they can be passed as additional arguments, each separated by commas.

Usage

```
InitialEstimate(Initial = numeric(), ...)
```

Arguments

`Initial`	Numeric. Initial estimate for the model parameter.
`...`	Additional initial estimate(s) for the model parameter.

Value

An object of class `InitialEstimate`.

See Also

Functions used for Theta specification: `Theta()`, `create_ModelPD()`, `create_ModelPK()`, `modify_Theta()`

Examples

```
InitialEstimate(1)
InitialEstimate(c(0, 1, Inf), c(-Inf, 2, 10))
```

list_Covariates *List Covariates in the currrent PML set*

Description

This function lists the names of covariates in a given set of PMLParametersSets.

Usage

```
list_Covariates(PMLParametersSets, IncludeAll = FALSE, IncludeCustom = TRUE)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
IncludeAll	Logical. Should the names of covariates with None state or covariates inside structural parameters with None state be included or not.
IncludeCustom	Logical. Should the names of covariate, fcovariate and interpolate statements (from the PML code of custom spaces) be included or not. Default is TRUE.

Value

A character vector containing the names of covariates.

See Also

add_Covariate() remove_Covariate() Covariate()

Examples

```
PMLParametersSets <- get_PMLParametersSets()
PMLParametersSets <- add_Covariate(PMLParametersSets,
                                   Name  = "WT")
list_Covariates(PMLParametersSets)
```

---

list_Dosepoints *List Dosepoints in the current PML set*

Description

This function lists the names of dosepoints in a given set of PMLParametersSets.

Usage

```
list_Dosepoints(PMLParametersSets, IncludeAll = FALSE, IncludeCustom = TRUE)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
IncludeAll	Logical. Should the names of dosepoints with None state be included or not. Default is FALSE.
IncludeCustom	Logical. Should the names of custom dosepoint and dosepoint2 statements (from the PML code of custom spaces) be included or not. Default is TRUE.

Value

A character vector containing the names of dosepoints

See Also

modify_Dosepoint()

Examples

```
PMLParametersSets <-
  get_PMLParametersSets(
    Absorption = c("First-Order", "Gamma"))
list_Dosepoints(PMLParametersSets)
```

---

list_Observations *List Observations in the current PML set*

Description

This function lists the names of Observations in a given PMLModels class instance.

Usage

```
list_Observations(
  PMLParametersSets,
  IncludeCustom = TRUE,
  ObservationsOnly = TRUE
)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
IncludeCustom	Logical. Should the names of responses (observe, multi, ordinal, count, event and LL) from the PML code of custom spaces be included or not. Default is TRUE.
ObservationsOnly	Logical. If TRUE (default), only the names of observe responses are included in the PML code generated for custom spaces. Non-observed response names (such as multi, ordinal, count, event, and LL) are not included. Ignored if IncludeCustom == FALSE.

Value

A character vector containing the names of Observations

See Also

Observation() modify_Observation() remove_Observation()

Examples

```
PMLParametersSets <-
  create_ModelPK(
    Absorption = c("First-Order", "Gamma"),
    EliminationCpt = c(TRUE, FALSE))
list_Observations(PMLParametersSets)
```

`list_Omegas` *List Unique Omega Names*

Description

This function lists the unique names of Omega parameters in a given set.

Usage

```
list_Omegas(PMLParametersSets, IncludeAll = FALSE, IncludeCustom = TRUE)
```

Arguments

`PMLParametersSets`	`PMLModels` class instance or an element (one PML structure) of this class or `StParm` class.
`IncludeAll`	Logical. Whether should the omega names to be inlcuded from structural parameters, covariates or omegas with a `State == 'None'`.
`IncludeCustom`	Logical. Should the names of custom `ranef` statements (from the PML code of custom spaces) be included or not. Default is `TRUE`.

Value

A character vector containing the unique names of Omega parameters.

See Also

`Omega()` `modify_Omega()`

Examples

```
PMLParametersSets <- create_ModelPK()
list_Omegas(PMLParametersSets)
```

list_StParms *List Structural Parameters in the currrent PML set*

Description

This function lists the names of structural parameters in a given set of PMLParametersSets.

Usage

```
list_StParms(PMLParametersSets, IncludeAll = FALSE, IncludeCustom = TRUE)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
IncludeAll	Logical. Should the names of structural parameters with None state be included or not. Default is FALSE.
IncludeCustom	Logical. Should the names of custom stparm statements (from the PML code of custom spaces) be included or not. Default is TRUE.

Value

A character vector containing the names of structural parameters.

See Also

add_StParm() modify_StParm()

Examples

```
PMLParametersSets <- get_PMLParametersSets()
list_StParms(PMLParametersSets)
```

list_Thetas *List Unique Theta Names*

Description

This function lists the unique names of Theta parameters in a given set.

Usage

```
list_Thetas(PMLParametersSets, IncludeAll = FALSE, IncludeCustom = TRUE)
```

Arguments

PMLParametersSets	PMLModels class instance or an element (one PML structure) of this class or StParm class.
IncludeAll	Logical. Whether should the Theta names to be inlcuded from structural parameters, covariates or thetas with a State == 'None'.
IncludeCustom	Logical. Should the names of custom theta statements (from the PML code of custom spaces) be included or not. Default is TRUE.

Value

A character vector containing the unique names of Theta parameters.

See Also

Theta() modify_Theta()

Examples

```
PMLParametersSets <- create_ModelPD()
list_Thetas(PMLParametersSets)
```

---

modify_Dosepoint *Modify Dosepoint in PML models*

Description

Modify Dosepoint in PML models

Usage

```
modify_Dosepoint(
  PMLParametersSets,
  DosepointName,
  tlag,
  bioavail,
  duration,
  rate,
  PMLStructures = NULL
)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
DosepointName	A character string giving the name of the Dosepoint.
tlag	An optional structural parameter giving the time lag for the doses coming into current Dosepoint.
bioavail	An optional structural parameter giving the bioavailability of the doses coming into current Dosepoint.
duration	An optional structural parameter giving the duration of infusion for the doses coming into current Dosepoint.
rate	An optional structural parameter giving the rate of infusion for the doses coming into current Dosepoint.
PMLStructures	Character or character vector specifying names of PML structures in which the dosepoint statement will be modified. For the naming convention of PMLStructures, see Details section of get_PMLParametersSets().

Details

This function can only be used to modify the structural parameters in the built-in models (i.e., created using either create_ModelEmax() or create_ModelPK()).

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

list_Dosepoints()

Functions used for Dosepoint specification: Dosepoint(), create_ModelPK()

Examples

```
PMLParametersSets <-
  get_PMLParametersSets(CompartmentsNumber = c(1, 2, 3))
# update structural paramter type
PMLParametersSetsVMod <-
 modify_Dosepoint(PMLParametersSets,
            DosepointName = "A1",
            tlag = StParm(StParmName = "Tlag",
                          State = "Searched"))
```

`modify_Observation` *Modify Observation class in PML models*

Description

Modify Observation class in PML models

Usage

```
modify_Observation(
  PMLParametersSets,
  ObservationName,
  SigmasChosen,
  BQL,
  BQLValue,
  Frozen,
  ResetObs,
  Covariates,
  PMLStructures = NULL
)
```

Arguments

`PMLParametersSets`
: A list of PML parameters sets (`PMLModels` class instance).

`ObservationName`
: A character string giving the name of the Observation.

`SigmasChosen`
: a Sigmas class instance or a list specifying the chosen sigma values for different error models. 0s are treated as no values. Inside Observation class it is transormed and kept as Sigmas class. The list could contain the following error models:

  - Additive The additive error sigma value.
  - LogAdditive The log-additive error sigma value.
  - Proportional The proportional error sigma value.
  - AdditiveMultiplicative A numeric vector specifying the additive and multiplicative parts for the additive-multiplicative error model. The vector should have names `PropPart` and `AddPart`.
  - MixRatio A numeric vector specifying the proportional and additive parts for the mix-ratio error model. The vector should have names `PropPart` and `AddPart`.
  - Power A numeric vector specifying the standard deviation and power parts for the power error model. The vector should have names `StdevPart` and `PowerPart`.

`BQL`
: A logical value indicating whether the dataset contains BQL values and they should be taken into account (M3 method).

BQLValue	An optional numeric positive value of static LLOQ. Applicable only when BQL argument is TRUE. Any observed value less than or equal to that LLOQ value is treated as censored.
Frozen	A logical value indicating if the standard deviation (Stdev) is frozen.
ResetObs	A logical value indicating if the Observation variable should be reset to 0 after observation (doafter={A0=0;}). Applicable for elimination compartment.
Covariates	A list of covariates (Covariate instances) that should be included in the model, but not linked to any of structural parameters. Used with "Emax" PD models ('C' covariate is added automatically when creating a new model, but should be added manually when modifying the model).
PMLStructures	Character or character vector specifying names of PML structures in which the observation will be modified. For the naming convention of PMLStructures, see Details section of create_ModelPK() for PK models and create_ModelPD() for PD models.

Details

This function can only be used to modify the structural parameters in the built-in models (i.e., created using either create_ModelEmax() or create_ModelPK()).

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

list_Observations()

Functions used for Observation specification: Observation(), ObservationCustom(), Sigmas(), create_ModelPD(), create_ModelPK(), remove_Observation()

Examples

```
PMLParametersSets <-
  create_ModelPK(CompartmentsNumber = c(1, 2, 3))
# update structural paramter type
PMLParametersSetsVMod <-
 modify_Observation(
   PMLParametersSets,
   ObservationName = "CObs",
   SigmasChosen = Sigmas(Proportional = 0,
                         AdditiveMultiplicative =
                           list(PropPart = 0.1, AddPart = 10)))

print(PMLParametersSetsVMod)
```

modify_Omega *Modify Omega Parameters in PML Models*

Description

This function allows to modify Omega parameters in a list of PML models (PMLModels class instance created by get_PMLParametersSets()).

Usage

```
modify_Omega(
  PMLParametersSets,
  Name,
  InitialOmega,
  State,
  Frozen,
  PMLStructures = NULL
)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
Name	A character string specifying the name of the Omega.
InitialOmega	Numeric specifying the initial value of the Omega. Default value is 1.
State	Character specifying the presence of the Omega. Possible values are: • None The Omega does not exist in the specified PMLStructures. • Present The Omega exists in the specified PMLStructures (the default). • Searched The presence of the Omega is searched.
Frozen	A logical value indicating whether the Omega is frozen or not.
PMLStructures	Character or character vector specifying names of PML structures in which the Omega will be modified. For the naming convention of PMLStructures, see Details section of create_ModelPK() for PK models and create_ModelPD() for PD models.

Details

- If the specified Omega does not exist in the PML models, a warning will be issued, and no modifications will be made.
- The current functionality does not support modifying custom omegas (ranefs) that are defined within the PML code of custom model spaces.

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

`list_Omegas()`

Functions used for Omega specification: `Omega()`, `create_ModelPD()`, `create_ModelPK()`

Examples

```
PMLParametersSets12 <- create_ModelPK(CompartmentsNumber = c(1, 2))
# Modify an Omega parameter named "nV" with new Initial Estimate and
# Frozen flag
PMLParametersSets12Mod1 <-
  modify_Omega(PMLParametersSets12,
               Name = "nV",
               InitialOmega = 0.3,
               State = "Present",
               Frozen = TRUE,
               PMLStructures = "PK1IVC")

print(PMLParametersSets12Mod1)
```

---

modify_StParm *Modify structural parameter in PML models set*

Description

Modify structural parameter in PML models set

Usage

```
modify_StParm(
  PMLParametersSets,
  StParmName,
  Type = "LogNormal",
  State = "Present",
  ThetaStParm,
  OmegaStParm,
  Covariates,
  PMLStructures = NULL
)
```

Arguments

`PMLParametersSets`	A list of PML parameters sets (`PMLModels` class instance).
`StParmName`	Character specifying the name of the structural parameter to be modified.
`Type`	Character specifying the type of the structural parameter. Options are

- LogNormal The PML statement of the structural parameter will look like the following:

```
stparm(V = tvV * wt^dVdwt * exp(nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))
```

- LogNormal1 The PML statement of the structural parameter will look like the following:

```
stparm(V = (tvV + wt*dVdwt) * exp(nV + nVx0*( Occasion==0) + nVx1*( Occasion==1)))
```

- LogNormal2 The PML statement of the structural parameter will look like the following:

```
stparm(V = exp(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occason==1)))
```

- LogitNormal The PML statement of the structural parameter will look like the following:

```
stparm(V = ilogit(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))
```

- Normal The PML statement of the structural parameter will look like the following:

```
stparm(V = tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1))
```

`State` character string that indicates the presence of the structural parameter. Options are:

- None The structural parameter does not exist in the specified PMLStructures.
- Present The structural parameter exists in the specified PMLStructures (the default).
- Searched The presence of the structural parameter is searched.

`ThetaStParm` A Theta class instance inside the structural parameter. If not given, the associated Theta will be automatically created with its name set to "tv" + StParmName.

`OmegaStParm` An Omega class instance inside the structural parameter. If not given, the associated Omega will be automatically created with its name set to "n" + StParmName

`Covariates` A list of covariates (Covariate instances) that should be included in the structural parameter statement.

`PMLStructures` Character or character vector specifying names of PML structures to whichthe structural parameter will be added. For the naming convention of PMLStructures, see Details section of `get_PMLParametersSets()`.

Details

This function can only be used to modify the structural parameters in the built-in models (i.e., created using either `create_ModelEmax()` or `create_ModelPK()`) or in the custom models if they are added with `add_StParm()`.

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

`Dosepoint()` `list_StParms()`

Functions used for StParm specification: `StParm()`, `add_StParm()`, `create_ModelPD()`, `create_ModelPK()`, `modify_StParmCustom()`, `remove_StParm()`

Examples

```
PMLParametersSets <-
  get_PMLParametersSets(CompartmentsNumber = c(1, 2, 3))
# update structural parameter type
PMLParametersSetsVMod <-
 modify_StParm(PMLParametersSets,
            StParmName = "V",
            Type = "LogitNormal")
```

`modify_StParmCustom` *Modify custom structural parameter in PML spaces*

Description

Modify custom structural parameter in PML spaces

Usage

```
modify_StParmCustom(
  PMLParametersSets,
  StParmName,
  Type,
  State,
  ThetaStParm,
  OmegaStParm,
  Covariates,
  PMLStructures = NULL
)
```

Arguments

`PMLParametersSets`
: A list of PML parameters sets (`PMLModels` class instance).

`StParmName`
: Character specifying the name of the structural parameter to be added.

`Type`
: Character specifying the type of the structural parameter. Options are

- `LogNormal` The PML statement of the structural parameter will look like the following:

```
stparm(V = tvV * wt^dVdwt * exp(nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))
```

- `LogNormal1` The PML statement of the structural parameter will look like the following:

```
stparm(V = (tvV + wt*dVdwt) * exp(nV + nVx0*( Occasion==0) + nVx1*( Occasion==1)))
```

- `LogNormal2` The PML statement of the structural parameter will look like the following:

```
stparm(V = exp(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occason==1)))
```

- `LogitNormal` The PML statement of the structural parameter will look like the following:

```
stparm(V = ilogit(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))
```

- `Normal` The PML statement of the structural parameter will look like the following:

```
stparm(V = tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1))
```

`State` character string that indicates the presence of the structural parameter. Options are:

- `None` The structural parameter does not exist in the specified `PMLStructures`.
- `Present` The structural parameter exists in the specified `PMLStructures` (the default).
- `Searched` The presence of the structural parameter is searched.

`ThetaStParm` A Theta class instance inside the structural parameter. If not given, the associated Theta will be automatically created with its name set to "tv" + `StParmName`.

`OmegaStParm` An Omega class instance inside the structural parameter. If not given, the associated Omega will be automatically created with its name set to "n" + `StParmName`

`Covariates` A list of covariates (`Covariate` instances) that should be included in the structural parameter statement.

`PMLStructures` Character or character vector specifying names of PML structures to whichthe structural parameter will be added. For the naming convention of PMLStructures, see Details section of `get_PMLParametersSets()`.

Details

This function can be applied to the custom models. It allows modification of custom structural parameters defined in the PML code of these spaces.

When modifying a custom structural parameter, the corresponding `stparm` statement is removed from the PML code, and the updated parameter is added back as a `StParm` class using the provided arguments. Similarly, associated `fixef` and `ranef` statements related to the custom structural parameter are removed.

Please note that this function is specifically designed for modifying custom structural parameters. For non-custom parameters, use `modify_StParm()`.

Value

An updated list of PML models (`PMLModels` class instance) matching the specified options.

See Also

`Dosepoint()` `list_StParms()` `modify_StParm()`

Functions used for StParm specification: `StParm()`, `add_StParm()`, `create_ModelPD()`, `create_ModelPK()`, `modify_StParm()`, `remove_StParm()`

Examples

```
# Modify the custom structural parameter 'Cl':
OneCpt_CustomCode <-
  paste0(
    "\nderiv(A1 = - Cl * C)",
    "\ndosepoint(A1)",
    "\ndosepoint2(A1, tlag = 12)",
    "\nC = A1 / V",
    "\nerror(CEps = 0.01)",
    "\nobserve(CObs = C + CEps * sqrt(1 + C^2 * (CMultStdev/sigma())^2), bql = 0.01)",
    "\nstparm(V = tvV * exp(nV))",
    "\nstparm(Cl = tvCl * exp(nCl))",
    "\nstparm(CMultStdev = tvCMultStdev)",
    "\nfixef(tvV = c(, 5, ))",
    "\nfixef(tvCl = c(, 1, ))",
    "\nfixef(tvCMultStdev = c(, 0.1, ))",
    "\nranef(diag(nV) = c(1))",
    "\nranef(diag(nCl) = c(1))\n"
  )

OneCpt_CustomCode <-
 modify_StParmCustom(
   create_CustomSpace(OneCpt_CustomCode),
                      StParmName = "Cl",
                      Type = "Normal")
```

`modify_Theta` *Modify Theta Parameters in PML Models*

Description

This function allows to modify Theta parameter in a list of PML models (`PMLModels` class instance created by `create_ModelPK()` or create_ModelPD).

Usage

```
modify_Theta(
  PMLParametersSets,
  Name,
  InitialEstimates,
  Frozen,
  PMLStructures = NULL
)
```

Arguments

`PMLParametersSets`
: A list of PML parameters sets (`PMLModels` class instance).

Name	Character specifying the name of the Theta to be modified.
InitialEstimates	An InitialEstimate() class instance or a numerical value for the initial estimate of the Theta or a numeric vector length three with its elements representing the lower bound, initial estimate.
Frozen	A logical value indicating whether the Theta will be estimated or not.
PMLStructures	Character or character vector specifying names of PML structures in which the Theta parameter will be modified. For the naming convention of PMLStructures, see Details section of create_ModelPK() for PK models and create_ModelPD() for PD models..

Details

- If the specified Theta does not exist in the PML models, a warning will be issued, and no modifications will be made. Thetas associated with structural parameters in the proportional part of MixRatio and Additive+Proportional error models can also be modified.
- The current functionality does not support modifying custom thetas (fixefs) that are defined within the PML code of custom model spaces.

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

InitialEstimate()

Functions used for Theta specification: InitialEstimate(), Theta(), create_ModelPD(), create_ModelPK()

Examples

```
PMLParametersSets <- create_ModelPK(CompartmentsNumber = c(1, 2))
# Modify a Theta parameter named "tvV" with new Initial Estimates and
# Frozen flag
PMLParametersSetsMod1 <-
  modify_Theta(PMLParametersSets,
               Name = "tvV",
               Frozen = TRUE,
               InitialEstimates = 0.3)

print(PMLParametersSetsMod1)

PMLParametersSetsMod2 <-
  add_StParm(PMLParametersSets = PMLParametersSetsMod1,
             StParmName = "Duration",
             State = "Searched",
             PMLStructures = "PK2IVC",
             DosepointArgName = "duration")

PMLParametersSetsMod3 <-
  modify_Theta(PMLParametersSets = PMLParametersSetsMod2,
```

```
                Name = "tvDuration",
                InitialEstimates = c(2, 4, Inf))

print(PMLParametersSetsMod3)
```

`Observation` *Create an instance of Observation class.*

Description

This function creates a new instance of Observation object and validates it.

Usage

```
Observation(
  ObservationName = "CObs",
  SigmasChosen = Sigmas(Proportional = 0.1),
  BQL = FALSE,
  BQLValue = NA,
  Frozen = FALSE,
  ResetObs = FALSE,
  Covariates = list(),
  PMLStructure = character()
)
```

Arguments

`ObservationName`
: A character string giving the name of the Observation.

`SigmasChosen`
: a Sigmas class instance or a list specifying the chosen sigma values for different error models. 0s are treated as no values. Inside Observation class it is transormed and kept as Sigmas class. The list could contain the following error models:

- Additive The additive error sigma value.
- LogAdditive The log-additive error sigma value.
- Proportional The proportional error sigma value.
- AdditiveMultiplicative A numeric vector specifying the additive and multiplicative parts for the additive-multiplicative error model. The vector should have names `PropPart` and `AddPart`.
- MixRatio A numeric vector specifying the proportional and additive parts for the mix-ratio error model. The vector should have names `PropPart` and `AddPart`.
- Power A numeric vector specifying the standard deviation and power parts for the power error model. The vector should have names `StdevPart` and `PowerPart`.

BQL	A logical value indicating whether the dataset contains BQL values and they should be taken into account (M3 method).
BQLValue	An optional numeric positive value of static LLOQ. Applicable only when BQL argument is TRUE. Any observed value less than or equal to that LLOQ value is treated as censored.
Frozen	A logical value indicating if the standard deviation (Stdev) is frozen.
ResetObs	A logical value indicating if the Observation variable should be reset to 0 after observation (doafter={A0=0;}). Applicable for elimination compartment.
Covariates	A list of covariates (Covariate instances) that should be included in the model, but not linked to any of structural parameters. Used with "Emax" PD models ('C' covariate is added automatically when creating a new model, but should be added manually when modifying the model).
PMLStructure	Character specifying the name of PML structure in which the observation should be added. For the naming convention of PMLStructures, see Details section of get_PMLParametersSets().

Value

A new Observation object

See Also

Functions used for Observation specification: ObservationCustom(), Sigmas(), create_ModelPD(), create_ModelPK(), modify_Observation(), remove_Observation()

Examples

```
A0Obs <-
  Observation(ObservationName = "A0Obs",
              SigmasChosen = list(Additive = 2,
                                  Power = c(Stdev = 10, Power = 0.5)),
              Frozen = FALSE,
              ResetObs = TRUE,
              PMLStructure = "PK1FOC")

CObs <- Observation("CObs", Frozen = TRUE, PMLStructure = "2Cpt")
```

ObservationCustom *Create an instance of custom Observation class.*

Description

This function creates a new instance of custom Observation object and validates it. All PML responses are supported (observe, multi, LL, event, count, ordinal)

Usage

```
ObservationCustom(
  ObservationName = "CObs",
  Type = "observe",
  Statement = "",
  StatementNames = list(),
  Sigma = list(),
  Dobefore = c(),
  Doafter = c(),
  BQL = FALSE,
  BQLValue = NA,
  PMLStructure = character()
)
```

Arguments

`ObservationName`	A character string giving the name of the Observation.
`Type`	One of the following: `observe`, `multi`, `LL`, `event`, `count`, `ordinal`
`Statement`	A character string giving the RHS of response statement without `Type`.
`StatementNames`	A character vector giving the names of variables used in the `Statement`.
`Sigma`	a list specifying the chosen sigma value Should be given only if `Type == "observe"`
`Dobefore`	A character string specifying the sequence of operations to be performed before current observation event.
`Doafter`	A character string specifying the sequence of operations to be performed after current observation event.
`BQL`	A logical value indicating whether the dataset contains BQL values and they should be taken into account (M3 method).
`BQLValue`	An optional numeric positive value of static LLOQ. Applicable only when BQL argument is `TRUE`. Any observed value less than or equal to that LLOQ value is treated as censored.
`PMLStructure`	Character specifying the name of PML structure in which the observation should be added. For the naming convention of PMLStructures, see Details section of `create_ModelPK()`.

Value

A new Observation object

See Also

Functions used for Observation specification: `Observation()`, `Sigmas()`, `create_ModelPD()`, `create_ModelPK()`, `modify_Observation()`, `remove_Observation()`

`Omega` *Create an Omega instance with validation*

Description

This function creates an Omega instance with the given parameters and validates it.

Usage

```
Omega(
  Name = character(),
  InitialOmega = 1,
  State = "Present",
  Frozen = FALSE,
  StParmName = character(),
  PMLStructure = character()
)
```

Arguments

`Name`	A character string specifying the name of the Omega.
`InitialOmega`	Numeric specifying the initial value of the Omega. Default value is 1.
`State`	Character specifying the presence of the Omega. Possible values are: • `None` The Omega does not exist in the specified `PMLStructures`. • `Present` The Omega exists in the specified `PMLStructures` (the default). • `Searched` The presence of the Omega is searched.
`Frozen`	A logical value indicating whether the Omega is frozen or not.
`StParmName`	A character string specifying the corresponding structural parameter name.
`PMLStructure`	PML structure current omega belongs to.

Value

An Omega instance.

See Also

`list_Omegas()`

Functions used for Omega specification: `create_ModelPD()`, `create_ModelPK()`, `modify_Omega()`

Examples

```
nV <- Omega("nV")
```

output.CustomSpace *Output a Custom Space*

Description

This function generates the PML code representation of a custom space.

Usage

```
## S3 method for class 'CustomSpace'
output(x, ...)
```

Arguments

x	A CustomSpace object.
...	Additional arguments (not used).

Value

A character string containing the PML code.

pyDarwinOptionsGA *Create options for the Genetic Algorithm (GA) in pyDarwin.*

Description

This function allows you to set various options specific to the Genetic Algorithm (GA) in pyDarwin.

Usage

```
pyDarwinOptionsGA(
  elitist_num = 2,
  crossover_rate = 0.95,
  mutation_rate = 0.95,
  sharing_alpha = 0.1,
  selection = "tournament",
  selection_size = 2,
  crossover_operator = "cxOnePoint",
  mutate = "flipBit",
  attribute_mutation_probability = 0.1,
  niche_penalty = 20
)
```

Arguments

elitist_num	A positive integer specifying the number of best models from any generation to carry over, unchanged, to the next generation. Functions like the Hall of Fame in DEAP. Default: 2
crossover_rate	A real value (between 0.0 and 1.0) specifying the fraction of mating pairs that will undergo crossover. Default: 0.95
mutation_rate	A real value (between 0.0 and 1.0) specifying the probability that at least one bit in the genome will be "flipped", 0 to 1, or 1 to 0. Default: 0.95
sharing_alpha	A real value specifying the parameter of the niche penalty calculation. Default: 0.1
selection	A string specifying the selection algorithm for the GA. Currently, only "tournament" is available. Default: "tournament"
selection_size	A positive integer specifying the number of "parents" to enter in the selection. 2 is highly recommended, experience with other values is very limited. Default: 2
crossover_operator	A string specifying the algorithm for crossover. Only "cxOnePoint" (single-point crossover) is available. Default: "cxOnePoint"
mutate	A string specifying the algorithm for mutation. Currently, only "flipBit" is available. Default: "flipBit"
attribute_mutation_probability	A real value specifying the probability of any bit being mutated (real value between 0.0 and 1.0). Default: 0.1
niche_penalty	A positive real value used for the calculation of the crowding penalty. The niche penalty is calculated by first finding the "distance matrix", the pair-wise Mikowski distance from the present model to all other models. The "crowding" quantity is then calculated as the sum of: (distance/niche_radius)^sharing_alpha for all other models in the generation for which the Mikowski distance is less than the niche radius. Finally, the penalty is calculated as: exp((crowding-1)*niche_penalty)-1. The objective of using a niche penalty is to maintain diversity of models, to avoid premature convergence of the search by penalizing when models are too similar to other models in the current generation. Default: 20

Value

An object of class "pyDarwinOptionsGA" containing the specified GA options.

Examples

```
# Create GA options with default values
options <- pyDarwinOptionsGA()

# Create GA options with custom values
options <-
  pyDarwinOptionsGA(elitist_num = 4,
                    crossover_rate = 0.9,
                    mutation_rate = 0.8,
```

```
                    sharing_alpha = 0.2)
```

`pyDarwinOptionsGridAdapter` *Grid Adapter Options for pyDarwin*

Description

This function creates a list of grid adapter options for pyDarwin, which are used to configure the interaction between pyDarwin and grid computing environments.

Usage

```
pyDarwinOptionsGridAdapter(
  python_path = "~/darwin/venv/bin/python",
  submit_search_command = paste("qsub -b y -cwd -o {project_stem}_out.txt",
    "-e {project_stem}_err.txt -N '{project_name}'"),
  submit_command = paste("qsub -b y -o {results_dir}/{run_name}.out",
    "-e {results_dir}/{run_name}.err -N {job_name}"),
  submit_job_id_re = "Your job (\\w+) \\(\".+?\"\\) has been submitted",
  poll_command = "qstat -s z",
  poll_job_id_re = "^\\s+(\\w+)",
  poll_interval = 10,
  delete_command = "qdel {project_stem}-*"
)
```

Arguments

`python_path`	Required. Path to Python interpreter, preferably to the instance of the interpreter located in the virtual environment where pyDarwin is deployed. The path must be available to all grid nodes that run jobs.
`submit_search_command`	Required. A command that submits a search job to the grid queue. This command is used for the entire search.
`submit_command`	Required. A command that submits individual runs to the grid queue. The actual command submitted to the queue is constructed by pyDarwin. It should not include `<python_path> -m darwin.run_model`.
`submit_job_id_re`	Required. A regular expression pattern to extract the job ID after submission. The job ID must be captured with the first capturing group.
`poll_command`	Required. A command that retrieves finished jobs from the grid controller. If the controller/setup allows to specify ids/patterns in polling commands, do it. Otherwise, all finished jobs should be polled using commands `qstat -s z`.
`poll_job_id_re`	Required. A regular expression pattern to find a job ID in every line of the poll_command output. Similar to submit_job_id_re.

`poll_interval`	Optional. How often to poll jobs (in seconds). Default is 10 seconds.
`delete_command`	Optional. A command that deletes all unfinished jobs related to the search when you stop it. It may delete all of them by ID (e.g., `qdel {job_ids}`) or by mask (e.g., `qdel {project_stem}-*`)..

Value

A list containing the configured grid adapter options.

Examples

```
grid_options <- pyDarwinOptionsGridAdapter(
  python_path = "~/darwin/venv/bin/python",
  submit_search_command =
   "qsub -b y -cwd -o {project_stem}_out.txt -e {project_stem}_err.txt -N '{project_name}'",
  submit_command =
   "qsub -b y -o {results_dir}/{run_name}.out -e {results_dir}/{run_name}.err -N {job_name}",
  submit_job_id_re = "Your job (\\w+) \\(\".+?\"\\) has been submitted",
  poll_command = "qstat -s z",
  poll_job_id_re = "^\\s+(\\w+)",
  poll_interval = 10,
  delete_command = "qdel {project_stem}-*"
)
```

---

pyDarwinOptionsPenalty

Create pyDarwin Penalty Options

Description

Generates a list of penalty parameters to be used in pyDarwin create_pyDarwinOptions function.

Usage

```
pyDarwinOptionsPenalty(
  theta = 10,
  omega = 10,
  sigma = 10,
  convergence = 100,
  covariance = 100,
  correlation = 100,
  condition_number = 100,
  non_influential_tokens = 1e-05
)
```

Arguments

theta	numeric: Penalty added to fitness/reward for each estimated THETA. A value of 3.84 corresponds to a hypothesis test with 1 df and $p < 0.05$ (for nested models), and a value of 2 for 1 df corresponds to the Akaike information criterion. Default: 10
omega	numeric: Penalty added to fitness/reward for each estimated OMEGA element. Default: 10
sigma	numeric: Penalty added to fitness/reward for each estimated SIGMA element. Default: 10
convergence	numeric: Penalty added to fitness/reward for failing to converge. Default: 100
covariance	numeric: Penalty added to fitness/reward for failing the covariance step (real number). If a successful covariance step is important, this can be set to a large value (e.g., 100), otherwise, set to 0. Default: 100
correlation	numeric: Penalty added to fitness/reward if any off-diagonal element of the correlation matrix of estimates has an absolute value > 0.95 (real number). This penalty will be added if the covariance step fails or is not requested. Default: 100
condition_number	numeric: Penalty added if the covariance step fails or is not requested, e.g., PRINT=E is not included in $COV. Additionally, if the covariance is successful and the condition number of the covariance matrix is > 1000, then this penalty is added to the fitness/reward. Default: 100
non_influential_tokens	numeric: Penalty added to fitness/reward if any tokens do not influence the control file (relevant for nested tokens). Should be very small (e.g., 0.0001), as the purpose is only for the model with non-influential tokens to be slightly worse than the same model without the non-influential token(s) to break a tie. Default: 0.00001

Value

A list of penalty options in pyDarwin optimization process.

Examples

```
# Create penalty options with default values
penalty_options <- pyDarwinOptionsPenalty()
# Create penalty options with custom values
penalty_options_custom <-
  pyDarwinOptionsPenalty(theta = 3.84,
                         omega = 8,
                         sigma = 6,
                         convergence = 50,
                         covariance = 80,
                         correlation = 60,
                         condition_number = 70,
                         non_influential_tokens = 0.0001)
```

pyDarwinOptionsPostprocess *Create pyDarwin Postprocess Options*

Description

Generates a list of postprocessing options to be used in pyDarwin optimization process.

Usage

```
pyDarwinOptionsPostprocess(
  use_r = FALSE,
  post_run_r_code = "{project_dir}/simplefunc.R",
  r_timeout = 30,
  use_python = FALSE,
  post_run_python_code = "{project_dir}/simplefunc.py"
)
```

Arguments

use_r	Logical: Whether to use R for postprocessing. If set to TRUE, R will be used to execute the post-processing script specified in post_run_r_code. Default: FALSE.
post_run_r_code	Character: The file path to the R script that contains post-processing code. This script will be executed after the pyDarwin optimization process finishes. It can perform additional analysis or manipulations on the generated results.
r_timeout	Numeric: The time limit (in seconds) for the execution of the post-processing R script. If the R script takes longer to execute than this timeout value, it will be terminated. Default: 30
use_python	Logical: Whether to use Python for postprocessing. If set to TRUE, Python will be used to execute the post-processing script specified in post_run_python_code. Default: FALSE
post_run_python_code	Character: The file path to the Python script that contains post-processing code. This script will be executed after the pyDarwin optimization process finishes. It can perform additional analysis or manipulations on the generated results. Default: {project_dir}/simplefunc.py

Value

A list of postprocessing options in pyDarwin optimization process.

Examples

```
# Create postprocess options with default values
postprocess_options <- pyDarwinOptionsPostprocess()
# Create postprocess options with custom values
postprocess_options_custom <-
  pyDarwinOptionsPostprocess(use_r = TRUE,
                             post_run_r_code = "{project_dir}/postprocess.R",
                             r_timeout = 60,
                             use_python = TRUE,
                             post_run_python_code = "{project_dir}/postprocess.py")
```

`pyDarwinOptionsPSO` *Create options for the Particle Swarm Optimization (PSO) in pyDarwin.*

Description

This function allows you to set various options specific to the Particle Swarm Optimization (PSO) in pyDarwin.

Usage

```
pyDarwinOptionsPSO(
  inertia = 0.4,
  cognitive = 0.5,
  social = 0.5,
  neighbor_num = 20,
  p_norm = 2,
  break_on_no_change = 5
)
```

Arguments

`inertia`	A real value specifying the particle coordination movement as it relates to the previous velocity (commonly denoted as w). Default: 0.4
`cognitive`	A real value specifying the particle coordination movement as it relates to its own best known position (commonly denoted as c1). Default: 0.5
`social`	A real value specifying the particle coordination movement as it relates to the current best known position across all particles (commonly denoted as c2). Default: 0.5
`neighbor_num`	A positive integer specifying the number of neighbors that any particle interacts with to determine the social component of the velocity of the next step. A smaller number of neighbors results in a more thorough search (as the neighborhoods tend to move more independently, allowing the swarm to cover a larger section of the total search space) but will converge more slowly. Default: 20

p_norm A positive integer specifying the Minkowski p-norm to use. A value of 1 is the sum-of-absolute values (or L1 distance) while 2 is the Euclidean (or L2) distance. Default: 2

break_on_no_change
A positive integer specifying the number of iterations used to determine whether the optimization has converged. Default: 5

Value

An object containing the specified options for the Particle Swarm Optimization (PSO) algorithm.

Examples

```
# Create PSO options with default values
options <- pyDarwinOptionsPSO()

# Create PSO options with custom values
options <- pyDarwinOptionsPSO(inertia = 0.2,
                              cognitive = 0.8,
                              social = 0.7,
                              neighbor_num = 10)
```

---

remove_Covariate *Remove Covariate from PML models*

Description

Remove Covariate from PML models

Usage

```
remove_Covariate(
  PMLParametersSets,
  Name,
  StParmNames = NULL,
  PMLStructures = NULL
)
```

Arguments

PMLParametersSets
A list of PML parameters sets (PMLModels class instance).

Name Character specifying the name of the covariate to be removed.

StParmNames Character or character vector specifying names of structural parameters from which the covariate will be removed. Can be set to NULL or not specified, for such case the covariate will be removed from all structural parameters.

PMLStructures Character or character vector specifying names of PML structures from which the covariate will be removed. For the naming convention of PMLStructures, see Details section of see details section of get_PMLParametersSets().

Details

The current functionality does not support removing custom covariates that are defined within the PML code of custom model spaces.

Value

An updated list of PML models (`PMLModels` class instance) matching the specified options.

See Also

`list_Covariates()`

Functions used for Covariate specification: `Covariate()`, `add_Covariate()`, `create_ModelPD()`, `create_ModelPK()`

Examples

```
PMLParametersSets <- get_PMLParametersSets()

PMLParametersSetsWT <-
 add_Covariate(PMLParametersSets,
               Name = "WT",
               Type = "Continuous",
               State = "Present",
               Direction = "Forward",
               Center = 70)

PMLParametersSetsVonly <-
 remove_Covariate(PMLParametersSets = PMLParametersSetsWT,
               Name = "WT",
               StParmNames = "Cl")
```

---

remove_Observation *Remove Observation from PML models*

Description

Remove Observation from PML models

Usage

```
remove_Observation(PMLParametersSets, ObservationName, PMLStructures = NULL)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
ObservationName	A character string giving the name of the Observation.
PMLStructures	Character or character vector specifying names of PML structures from which the observation will be removed. For the naming convention of PMLStructures, see Details section of create_ModelPK() for PK models and create_ModelPD() for PD models.

Details

The current functionality does not support modifying custom observations that are defined within the PML code of custom model spaces.

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

list_Observations()

Functions used for Observation specification: Observation(), ObservationCustom(), Sigmas(), create_ModelPD(), create_ModelPK(), modify_Observation()

Examples

```
PMLParametersSets <-
  create_ModelPK(
    CompartmentsNumber = c(2, 3),
    Parameterization = "Micro",
    Absorption = c("First-Order", "Gamma"),
    ByVector = TRUE,
    ClosedForm = TRUE,
    EliminationCpt = TRUE)

remove_Observation(PMLParametersSets,
                   ObservationName = "A0Obs",
                   PMLStructures = "PK3GME")
```

---

remove_StParm *Remove structural parameter from PML models*

Description

Remove structural parameter from PML models

Usage

```
remove_StParm(PMLParametersSets, StParmName, PMLStructures = NULL)
```

Arguments

PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
StParmName	character specifying the name for the structural parameter to be removed.
PMLStructures	Character or character vector specifying names of PML structures from which the structural parameter will be removed. For the naming convention of PML-Structures, see Details section of get_PMLParametersSets().

Details

Please make sure that structural parameter to be removed is not essential for the model. Usually the user does not need to remove any structural parameter. The only case is related to structural parameters in Dosepoint().

Value

An updated list of PML models (PMLModels class instance) matching the specified options.

See Also

Dosepoint() list_StParms()

Functions used for StParm specification: StParm(), add_StParm(), create_ModelPD(), create_ModelPK(), modify_StParm(), modify_StParmCustom()

Examples

```
PMLParametersSets <- get_PMLParametersSets(CompartmentsNumber = c(1, 2))

PMLParametersSetsDuration <-
 add_StParm(PMLParametersSets,
            StParmName = "Duration",
            State = "Searched",
            DosepointArgName = "duration")

PMLParametersSetsDuration1CptOnly <-
 remove_StParm(PMLParametersSetsDuration,
               StParmName = "Duration",
               PMLStructures = "PK2IVC")
```

`run_pyDarwin` *Run pyDarwin Model Search*

Description

This function runs a pyDarwin model search using the specified parameters. It launches the search process and monitors its progress.

Usage

```
run_pyDarwin(
  InterpreterPath,
  Flags = c("-u", "-m"),
  DirectoryPath = ".",
  TemplatePath = "template.txt",
  TokensPath = "tokens.json",
  OptionsPath = "options.json",
  Wait = TRUE
)
```

Arguments

`InterpreterPath`	Path to the Python interpreter executable.
`Flags`	Flags to pass to the Python interpreter. Refer to Python documentation for details. Note that `-m` is essential (runs library module as a script and terminates option list).
`DirectoryPath`	Optional path to the directory containing template file, tokens file and options file. If that argument is given, it overrides the paths of `TemplatePath`, `TokensPath`, `OptionsPath` with warning. Default is current working directory.
`TemplatePath`	Path to the template file.
`TokensPath`	Path to the tokens JSON file.
`OptionsPath`	Path to the options JSON file.
`Wait`	Logical. If `TRUE`, the function waits for the search process to complete and returns the results. If `FALSE`, the process is launched and exits immediately.

Value

If `Wait` is `TRUE`, a list containing the results of the search, including a data frame for all models executed, a final model and properties of the final model generated by pyDarwin when available. If nothing is available, `messages.txt` file text is given. If `Wait` is `FALSE`, `messages.txt` file location is returned where raw output of pyDarwin run is stored.

Examples

```
## Not run:
result <- run_pyDarwin(
  InterpreterPath = "~/darwin/venv/bin/python",
  DirectoryPath = "~/project_folder",
  TemplatePath = "template.txt",
  TokensPath = "tokens.json",
  OptionsPath = "options.json"
)

## End(Not run)
```

Sigmas *Create an instance of Sigmas class.*

Description

This function creates a new instance of different error models object to be applied. 0s are treated as no values.

Usage

```
Sigmas(
  Additive = 0,
  LogAdditive = 0,
  Proportional = 0.1,
  AdditiveMultiplicative = list(PropPart = 0, AddPart = 0),
  MixRatio = list(PropPart = 0, AddPart = 0),
  Power = list(PowerPart = 0, StdevPart = 0),
  ObservationName = ""
)
```

Arguments

`Additive`	The additive error sigma value.
`LogAdditive`	The log-additive error sigma value.
`Proportional`	The proportional error sigma value.
`AdditiveMultiplicative`	A list specifying the additive and multiplicative parts for the additive-multiplicative error model. The list should have elements `PropPart` and `AddPart`. Alternatively the proportional part (`PropPart`) could be presented as `StParm`, see `StParm()`.
`MixRatio`	A list specifying the proportional and additive parts for the mix-ratio error model. The list should have elements `PropPart` and `AddPart`. Alternatively the proportional part (`PropPart`) could be presented as `StParm`, see `StParm()`.

Power A numeric vector specifying the standard deviation and power parts for the power error model. The vector should have names StdevPart and PowerPart.

ObservationName
A character string giving the name of the Observation.

Value

A Sigmas class instance.

See Also

Functions used for Observation specification: Observation(), ObservationCustom(), create_ModelPD(), create_ModelPK(), modify_Observation(), remove_Observation()

Examples

```
RSE_CObs <-
  Observation(SigmasChosen =
    Sigmas(MixRatio = list(PropPart = 2,
                           AddPart = 0.01),
           Proportional = 0))
models <-
  create_ModelPK(CompartmentsNumber = 2,
                 CObs = RSE_CObs)
print(models)
```

---

specify_EngineParams *Specify engine parameters for model execution*

Description

Use to define extra engine parameters for model execution.

Usage

```
specify_EngineParams(
  sort = FALSE,
  ODE = c("MatrixExponent", "DVERK", "DOPRI5", "AutoDetect", "Stiff"),
  rtolODE = 1e-06,
  atolODE = 1e-06,
  maxStepsODE = 50000L,
  numIterations = 1000L,
 method = c("FOCE-ELS", "QRPEM", "Laplacian", "Naive-Pooled", "FOCE-LB", "IT2S-EM",
    "FO"),
  stdErr = c("Sandwich", "Auto-Detect", "Hessian", "Fisher-Score", "None"),
  isCentralDiffStdErr = TRUE,
  stepSizeStdErr = 0.01,
```

```
  numIntegratePtsAGQ = 1L,
  numIterNonParametric = 0L,
  allowSyntheticGradient = FALSE,
  numIterMAPNP = 0L,
  numRepPCWRES = 0L,
  stepSizeLinearize = 0.002,
  numDigitLaplacian = 7L,
  numDigitBlup = 13L,
  mapAssist = 0L,
  iSample = 300L,
  iAcceptRatio = 0.1,
 impDist = c("Normal", "DoubleExponential", "Direct", "T", "Mixture-2", "Mixture-3"),
  tDOF = 4L,
  numSampleSIR = 10L,
  numBurnIn = 0L,
  freezeOmega = FALSE,
  MCPEM = FALSE,
  runAllIterations = FALSE,
  scramble = c("Owen", "Tezuka-Faur", "None")
)
```

Arguments

`sort`	Logical; Specifying whether or not to sort the input data by subject and time values. Default is `TRUE`.
`ODE`	Character; Specifying the solver used to numerically solve Ordinary Differential Equations (ODEs). Options are • `MatrixExponent` (the default), • `DVERK`, • `DOPRI5`, • `AutoDetect`, • `Stiff`. Note: both `DVERK` and `DOPRI5` are non-stiff solvers. NLME will automatically switches to `DVERK` if ODEs are nonlinear.
`rtolODE`	Numeric; Specifying relative tolerance for the ODE solver. Not applicable when `ODE == MatrixExponent`.
`atolODE`	Numeric; Specifying absolute tolerance for the ODE solver.
`maxStepsODE`	Numeric; Specifying maximum number of allowable steps or function evaluations for the ODE solver.
`numIterations`	Numeric; Specifying maximum number of iterations for estimation.
`method`	Character; Specifying engine method for estimation. Options are: • `FOCE-ELS` (the default), • `QRPEM`, • `Laplacian`, • `Naive-Pooled`,

- FOCE-LB,
- IT2S-EM,
- FO.

Note: if model involves any discontinuous observed variable (e.g., count data) or BQL data, NLME will switch from default method FOCE-ELS to Laplacian.

stdErr Character; Specifying method for standard error computations. Options are:

- Auto-Detect (the default),
- Sandwich,
- Hessian,
- Fisher-Score,
- None.

Here None means that standard error calculations are not performed. Since when method = QRPEM only Fisher-Score standard error type is available in NLME, any selected option except None will reset to stdErr = "Fisher-Score".

isCentralDiffStdErr

Logical; Default TRUE uses central difference for stdErr calculations. Set to FALSE for forward difference method.

stepSizeStdErr Numeric; Specifying the step size used for stdErr calculations.

numIntegratePtsAGQ

Numeric; Specifying the number of integration points for adaptive Gaussian quadrature (AGQ) algorithm. Only applicable to models with method set to either FOCE-ELS or Laplacian.

numIterNonParametric

Numeric; Specifying the number of iterations to perform non-parametric estimation. Only applicable when method is not set to Naive-Pooled (otherwise ignored).

allowSyntheticGradient

Logical, Set to TRUE to use synthetic gradient during the estimation process. Only applicable to population models when method is not set to Naive-Pooled (otherwise ignored).

numIterMAPNP Numeric; Specifying the number of iterations to perform Maximum A Posterior (MAP) initial Naive Pooling (NP) run before estimation. Only applicable to population models when method is not set to Naive-Pooled (otherwise ignored).

numRepPCWRES Numeric; Specifying the number of replicates to generate the PCWRES after the simple estimation. Only applicable to population models when method is not set to Naive-Pooled (otherwise ignored).

stepSizeLinearize

Numeric; Specifying the step size used for numerical differentiation when linearizing the model function during the estimation process.

numDigitLaplacian

Numeric; Specifying the number of significant decimal digits for the Laplacian/ELS algorithm to use to reach convergence.

numDigitBlup Numeric; Specifying the number of significant decimal digits for the individual estimation to use to reach convergence.

`mapAssist` Numeric; Specifying the period used to perform MAP assistance (`mapAssist = 0` means that MAP assistance is not performed). Only applicable when `method == "QRPEM"`.

`iSample` Numeric; Specifying the number of samples. Only applicable when `method == "QRPEM"`.

`iAcceptRatio` Numeric; Specifying the acceptance ratio. Only applicable when `method == "QRPEM"`.

`impDist` Character; Specifying the distribution used for important sampling, and options are

- `Normal` (the default),
- `DoubleExponential`,
- `Direct`,
- `T`,
- `Mixture-2`,
- `Mixture-3`.

Only applicable to the model with `method = "QRPEM"`.

`tDOF` Numeric; Specifying the degree of freedom (allowed value is between 3 and 30) for T distribution. Only applicable when `method =="QRPEM"` and `impDist == "T"`.

`numSampleSIR` Numeric; Specifying the number of samples per subject used in the Sampling Importance Re-Sampling (SIR) algorithm to determine the number of SIR samples taken from the empirical discrete distribution that approximates the target conditional distribution. Only applicable to population models with `method = "QRPEM"`.

`numBurnIn` Numeric; Specifying the number of burn-in iterations to perform at startup to adjust certain internal parameters. Only applicable to population models with `method = "QRPEM"`.

`freezeOmega` Logical; Set to `TRUE` to freeze Omega but not Theta for the number of iterations specified in the `numBurnIn`. Only applicable to population models with `method = "QRPEM"`.

`MCPEM` Logical; Set to `TRUE` to use Monte-Carlo sampling instead of Quasi-Random. Only applicable to population models with `method = "QRPEM"`.

`runAllIterations` Logical; Set to `TRUE` to execute all requested iterations specified in `numIterations`. Only applicable to population models with `method = "QRPEM"`.

`scramble` Character; Specifying the quasi-random scrambling method to use, and options are

- `Owen` (the default),
- `Tezuka-Faur`,
- `None`.

Only applicable to population models with `method = "QRPEM"`.

Value

Character

See Also

write_ModelTemplateTokens(), specify_SimParams()

Examples

```
# default
EstArgs <- specify_EngineParams()
# QRPEM method
EstArgs <-
  specify_EngineParams(
    sort = TRUE,
    ODE = "DVERK",
    rtolODE = 1e-5,
    atolODE = 1e-5,
    maxStepsODE = 6000,
    numIterations = 100,
    method = "QRPEM",
    numIterMAPNP = 3,
    stdErr = "Fisher-Score",
    isCentralDiffStdErr = FALSE,
    iSample = 350,
    impDist = "Mixture-2",
    scramble = "Tezuka-Faur")
```

---

specify_SimParams *Specify engine parameters for model simulation*

Description

Use to define engine parameters for model simulation.

Usage

```
specify_SimParams(
  numReplicates = 100L,
  seed = 1234L,
  sort = FALSE,
  ODE = c("MatrixExponent", "DVERK", "DOPRI5", "AutoDetect", "Stiff"),
  rtolODE = 1e-06,
  atolODE = 1e-06,
  maxStepsODE = 50000L
)
```

Arguments

numReplicates	Integer; Number of replicates to simulate the model
seed	Integer; Random number generator seed

sort	Logical; Specifying whether or not to sort the input data by subject and time values. Default is TRUE.
ODE	Character; Specifying the solver used to numerically solve Ordinary Differential Equations (ODEs). Options are • MatrixExponent (the default), • DVERK, • DOPRI5, • AutoDetect, • Stiff. Note: both DVERK and DOPRI5 are non-stiff solvers. NLME will automatically switches to DVERK if ODEs are nonlinear.
rtolODE	Numeric; Specifying relative tolerance for the ODE solver. Not applicable when ODE == MatrixExponent.
atolODE	Numeric; Specifying absolute tolerance for the ODE solver.
maxStepsODE	Numeric; Specifying maximum number of allowable steps or function evaluations for the ODE solver.

Value

Character

See Also

write_ModelTemplateTokens(), specify_EngineParams(), Table()

Examples

```
SimArgs1 <- specify_SimParams()

SimArgs2 <-
  specify_SimParams(
    numReplicates = 100,
    seed = 1,
    ODE = "DVERK")
```

---

stop_pyDarwin *Stop pyDarwin Model Search*

Description

This function stops a pyDarwin model search.

Usage

```
stop_pyDarwin(
  InterpreterPath,
  Flags = c("-u", "-m"),
  ForceStop = FALSE,
  DirectoryPath = "."
)
```

Arguments

InterpreterPath	Path to the Python interpreter executable.
Flags	Flags to pass to the Python interpreter. Refer to Python documentation for details. Note that `-m` is essential (runs library module as a script and terminates option list).
ForceStop	Logical. If `TRUE`, `-f` flag is added to force stop search immediately.- Default is `FALSE`.
DirectoryPath	the `DirectoryPath` argument of `run_pyDarwin()` or the parent folder of options file passed to that function. Default is current working directory.

Value

Returned code of `system2()` call.

Examples

```
## Not run:
stop_pyDarwin(
  InterpreterPath = "~/darwin/venv/bin/python",
  DirectoryPath = "~/project_folder")

## End(Not run)
```

StParm *Create an instance of a Structural parameter.*

Description

This function creates a new instance of a Structural parameter.

Usage

```
StParm(
  StParmName = character(),
  Type = "LogNormal",
  State = "Present",
  ThetaStParm = list(),
  OmegaStParm = list(),
  Covariates = list(),
  PMLStructure = character()
)
```

Arguments

StParmName	Character specifying the name of the structural parameter.

Type	Character specifying the type of the structural parameter. Options are
	• LogNormal The PML statement of the structural parameter will look like the following:
	`stparm(V = tvV * wt^dVdwt * exp(nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))`
	• LogNormal1 The PML statement of the structural parameter will look like the following:
	`stparm(V = (tvV + wt*dVdwt) * exp(nV + nVx0*( Occasion==0) + nVx1*( Occasion==1)))`
	• LogNormal2 The PML statement of the structural parameter will look like the following:
	`stparm(V = exp(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occason==1)))`
	• LogitNormal The PML statement of the structural parameter will look like the following:
	`stparm(V = ilogit(tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1)))`
	• Normal The PML statement of the structural parameter will look like the following:
	`stparm(V = tvV + wt*dVdwt + nV + nVx0*(Occasion==0) + nVx1*(Occasion==1))`

State	character string that indicates the presence of the structural parameter. Options are: • None The structural parameter does not exist in the specified PMLStructures. • Present The structural parameter exists in the specified PMLStructures (the default). • Searched The presence of the structural parameter is searched.
ThetaStParm	A Theta class instance inside the structural parameter. If not given, the associated Theta will be automatically created with its name set to "tv" + StParmName.
OmegaStParm	An Omega class instance inside the structural parameter. If not given, the associated Omega will be automatically created with its name set to "n" + StParmName
Covariates	A list of covariates (Covariate instances) that should be included in the structural parameter statement.

`PMLStructure` Character specifying the name of PML structure for which current parameter should be attributed. For the naming convention of PMLStructures, see Details section of `create_ModelPK()` for PK models and `create_ModelPD()` for PD models.

Value

An instance of a structural parameter.

See Also

Functions used for StParm specification: `add_StParm()`, `create_ModelPD()`, `create_ModelPK()`, `modify_StParm()`, `modify_StParmCustom()`, `remove_StParm()`

Examples

```
# Create a Structural parameter instance with default values
V <- StParm(StParmName = "V")

# Create a Structural parameter with Normal type:
V2 <- StParm("V2",
       Type = "Normal",
       ThetaStParm = Theta(Name = "tvV2", InitialEstimates = 0.1))

# Create a Structural parameter instance with covariates:
Cl <- StParm(
  StParmName = "Cl",
  Covariates = Covariate(
    Name = "Period",
    Type = "Occasion",
    State = "Searched",
    Categories = c(1,2),
    Omegas = list(Omega(Name = "nPeriodx1", 2),
                  Omega(Name = "nPeriodx2", 3))),
  PMLStructure = "1CFOE")
```

`Table` *Class initializer for NLME tables*

Description

Creates Table class object used to specify triggers and columns for tables output.

Usage

```
Table(
  Name = "table01.csv",
  TimesList = numeric(0),
```

```
  CovrSet = "",
  WhenDose = "",
  WhenObs = "",
  VariablesList = "",
  KeepSource = FALSE,
  TimeAfterDose = FALSE,
  IRES = FALSE,
  Weight = FALSE,
  IWRES = FALSE,
  Mode = "all",
  ForSimulation = FALSE
)
```

Arguments

Name	Character; Name of the generated table.
TimesList	Numeric; Time values for simulation. Applicable for time-based models only. Ignored when keepSource=TRUE.
CovrSet	Character; Vector of covariate names. Simulation point is added when the covariate value is set.
WhenDose	Character or logical; Vector of dosing compartment names. Alternatively if WhenDose == TRUE, triggers are added for all dosepoints for each PMLParametersSet separately; that approach is useful when different models in the set have different dosing compartments. Simulation point is added when the dose value is set.
WhenObs	Character; String of observed variables names. Simulation point is added when the observation value is set.
VariablesList	Character; List of variables from the model for simulation.
KeepSource	Logical; Set to TRUE to keep the number of rows appearing in the table the same as the number of rows in the input dataset.
TimeAfterDose	Set to TRUE to output time after dose.
IRES	Logical; Set to TRUE to output individual residuals. Valid only if WhenObs is specified and ForSimulation==FALSE.
Weight	Logical; Set to TRUE to output the weight of current observation. Valid only if WhenObs is specified and ForSimulation==FALSE.
IWRES	Logical; Set to TRUE to output individual weighted residuals. Valid only if WhenObs is specified and ForSimulation==FALSE.
Mode	Character; The mode of output. Options are all (default), unique, first. Only applicable to non time-based models for the case where only CovrSet is defined or the case where only CovrSet and VariablesList are defined. Since current version supports time-based models only, this argument is not applicable and won't change the output.
ForSimulation	Logical; Set to TRUE if the table should be generated during simulation, otherwise the table will be generated after fitting.

Details

If the table has a flag `ForSimulation==TRUE`, it will be ignored and won't be generated during estimation stage. Simulation stage should be added for simulation table generation. Tables with `ForSimulation==FALSE` will be ignored during simulation stage.

Value

A Table class used to store custom table information.

Examples

```
table01 <-
  Table(Name = "table01.csv",
      TimesList = seq(1,3,1),
      CovrSet = "WT",
      WhenDose = "A1",
      WhenObs = "CObs",
      VariablesList = "C",
      KeepSource = FALSE,
      TimeAfterDose = TRUE,
      IRES = TRUE,
      Weight = TRUE,
      IWRES = TRUE,
      ForSimulation = FALSE)
```

Theta *Create a new Theta instance with validation.*

Description

Create a new Theta instance with validation.

Usage

```
Theta(
  Name = character(),
  InitialEstimates = 1,
  State = "Present",
  Frozen = FALSE,
  StParmName = character(),
  PMLStructure = character()
)
```

Arguments

Name	A character string representing the name of the Theta instance.
InitialEstimates	An InitialEstimate() class instance or a numerical value for the initial estimate of the Theta or a numeric vector length three with its elements representing the lower bound, initial estimate.
State	Character specifying the presence of the Theta. Possible values are: • None The Theta does not exist in the specified PMLStructure. • Present The Theta exists in the specified PMLStructure (the default) • Searched The presence of the Theta is searched.
Frozen	A logical value indicating whether the Theta will be estimated or not.
StParmName	A character specifying the corresponding structural parameter name. Used for the Name of current Theta construction if it is not specified as 'tv' + StParmName.
PMLStructure	PML structure current theta belongs to

Value

A Theta instance.

See Also

InitialEstimate() StParm()

Functions used for Theta specification: InitialEstimate(), create_ModelPD(), create_ModelPK(), modify_Theta()

Examples

```
# Create a new Theta instance with a name 'tvV' and initial value 2 (no bounds)
theta <- Theta(Name = "tvV", InitialEstimates = 2)
```

---

write_ModelTemplateTokens

Prints NLME metamodel template file and token json file using given options, filepaths and data

---

Description

This function generates and writes the model template and tokens files based on the provided inputs.

Usage

```
write_ModelTemplateTokens(
  TemplateFilePath = "template.txt",
  TokensFilePath = "tokens.json",
  Description = "",
  Author = "",
  DataFilePath,
  DataMapping = NULL,
  ColDef = "",
  PMLParametersSets,
  EstArgs = specify_EngineParams(),
  SimArgs = "",
  Tables = list(),
  AppendixRows = "",
  OmegaSearchBlocks = list()
)
```

Arguments

TemplateFilePath	TemplateFilePath NLME template file path to be written (usually txt).
TokensFilePath	json file path to be written (usually json).
Description	A problem name to be outputted in Description section.
Author	The author information for the model to be outputted in Author section.
DataFilePath	A data file path used by NLME.
DataMapping	A named vector ModelTerm = DataTerm for the used data file.
ColDef	A character string specifying additional column definitions in NLME column definition format. See https://onlinehelp.certara.com/phoenix/8.4/index.html#t=Phoenix_UserDocs%2FPML%2FColumn_mappings.htm
PMLParametersSets	A list of PML parameters sets (PMLModels class instance).
EstArgs	Estimation arguments for the model template. Please use specify_EngineParams to specify the arguments passed to NLME.
SimArgs	Simulation arguments for the model template. Please use specify_SimParams to specify the arguments passed to NLME.
Tables	A list of Table class instances specifying properties of the tables to be generated after fitting or during simulation.
AppendixRows	Additional rows to include in the model template appendix in NLME column definition format. See https://onlinehelp.certara.com/phoenix/8.4/index.html#t=Phoenix_UserDocs%2FPML%2FColumn_mappings.htm
OmegaSearchBlocks	A list of character vectors representing omega names to try to build block omegas.

Details

Terms <DosepointName>_Duration or <DosepointName>_Rate could be used to map rate/duration columns for the current dosepoints. Term <ObservationName>BQL could be used to map BQL flag for the current observation. AMT term could be used to map different main dosepoints (i.e. A1 will be mapped for Gamma, Aa for First-Order absorption etc.) If AMT term is used, additional terms Duration or Rate could be used; current function will map it to the main dosepoint of each Parameter set. But it is possible to map duration/rate for some dosepoint directly using terms <DosepointName>_Duration or <DosepointName>_Rate, it will override Duration or Rate terms mapping for current dosepoint.

Value

A list containing statements written to template and tokens files.

See Also

specify_EngineParams(), specify_SimParams(), Table()

Examples

```
# Write model template and tokens files
PMLParametersSets <- create_ModelPK(CompartmentsNumber = c(1,2))
# write test data frame
TempFolder <- tempdir()
TemplateFilePath <- file.path(TempFolder, "template.txt")
TokensFilePath <- file.path(TempFolder, "tokens.json")
DataFilePath <- file.path(TempFolder, "Data.csv")
write.csv(data.frame(id = 'id',
                     time = 'time',
                     AMT = 'AMT',
                     Conc = 'Conc',
                     age = 'age',
                     Weight = 'Weight',
                     CObsBQL = 'CObsBQL'),
                     DataFilePath)
write_ModelTemplateTokens(TemplateFilePath = TemplateFilePath,
                          TokensFilePath = TokensFilePath,
                          Description = "1-2Cpts try",
                          Author = "Certara",
                          DataFilePath = DataFilePath,
                          DataMapping = c(ID = "id",
                                          time = "time",
                                          CObs = "Conc",
                                          AMT = "AMT",
                                          "age"),
                          ColDef = "",
                          PMLParametersSets = PMLParametersSets,
                          EstArgs = specify_EngineParams(method = "QRPEM"),
                          SimArgs = specify_SimParams(numReplicates = 1000L),
                          Tables = list(Table(Name = "simtable1.csv",
                                              KeepSource = TRUE,
```

```
                                        VariablesList = "C",
                                        ForSimulation = TRUE)),
                OmegaSearchBlocks = list(c("nCl", "nV"), c("nCl2", "nV2")))
```

`write_pyDarwinOptions` *Write pyDarwin options to a JSON file.*

Description

This function takes a list of pyDarwin options and writes them to a JSON file in the specified format. The options can be generated using the `create_pyDarwinOptions` function or customized manually. The resulting JSON file can be used as input for a pyDarwin model search.

Usage

```
write_pyDarwinOptions(
  pyDarwinOptions = create_pyDarwinOptions(),
  file = "options.json",
  pretty = TRUE,
  digits = NA,
  auto_unbox = TRUE
)
```

Arguments

`pyDarwinOptions`	A list containing the pyDarwin options to be written to the JSON file. Default is the result of calling `create_pyDarwinOptions()` with default arguments.
`file`	Character: The path to the JSON file where the options will be written. Default is a file named "options.json" in the current working directory.
`pretty`	adds indentation whitespace to JSON output. Can be TRUE/FALSE or a number specifying the number of spaces to indent. See `prettify()`
`digits`	max number of decimal digits to print for numeric values. Use `I()` to specify significant digits. Use `NA` for max precision.
`auto_unbox`	automatically `unbox()` all atomic vectors of length 1. It is usually safer to avoid this and instead use the `unbox()` function to unbox individual elements. An exception is that objects of class `AsIs` (i.e. wrapped in `I()`) are not automatically unboxed. This is a way to mark single values as length-1 arrays.

Value

None (invisible `NULL`).

Examples

```
# Write pyDarwin options to a JSON file
Options <-
  create_pyDarwinOptions(author = "John Doe",
                         algorithm = "GA",
                         population_size = 10)
write_pyDarwinOptions(Options,
                      file = file.path(tempdir(), "options.json"))
```

Index