

Package ‘Certara.RsNLME’

September 15, 2026

Title Pharmacometric Modeling

Version 3.2.0

Description Facilitate Pharmacokinetic (PK) and Pharmacodynamic (PD) modeling and simulation with powerful tools for Nonlinear Mixed-Effects (NLME) modeling. The package provides access to the same advanced Maximum Likelihood algorithms used by the NLME-Engine in the Phoenix platform. These tools support a range of analyses, from parametric methods to individual and pooled data, and support integrated use within the Pirana pharmacometric workbench <doi:10.1002/psp4.70067>. Execution is supported both locally or on remote machines.

Depends R (>= 4.0)

License LGPL-3

URL <https://certara.github.io/R-RsNLME/>

Encoding UTF-8

LazyData true

Suggests rlang, knitr, rmarkdown, testthat, magrittr, ellmer (>= 0.4.0), callr (>= 3.7.0), ps (>= 1.7.0), withr, digest, curl, flextable

Imports xml2, assertthat, Certara.NLME8 (>= 3.2.0), data.table, jsonlite, methods, utils, ssh

Collate 'NlmePmlModelInfo.R' 'NlmeRandParamsMapping.R'
'NlmeParamsMapping.R' 'NlmeDoseMapping.R'
'NlmeEmaxParameters.R' 'NlmeIndirectParameters.R'
'NlmeModelAbsorption.R' 'NlmeModelParameterization.R'
'NlmePkParameters.R' 'NlmeModelType.R' 'NlmeDataset.r'
'NlmeColumnMapping.r' 'error_model.r' 'NlmeRandomEffectBlock.r'
'pml_model.r' 'BootstrapParams.R' 'NlmeParallelMethod.R'
'NlmeUserAuthentication.R' 'NlmeParallelHost.R'
'nlme_job_helpers.R' 'SimpleNlmeJob.R' 'BootNlmeJob.R'
'CovariateEffectModel.R' 'engine_params.r' 'FitNlmeJob.R'
'GenerateControlfile.R' 'GenerateParamsfile.R'
'NlmeCovariateParameter.r' 'NlmeRemoteExecutor.R'
'NlmeScenario.R' 'NlmeTableDef.R' 'SortColumns.R'

'ProfileParameters.R' 'ProfileNlmeJob.R' 'ProfileVar.R'
 'RandomEffectsMethods.R' 'ShotgunNlmeJob.R' 'SortByNlmeJob.R'
 'StepwiseParams.R' 'StepwiseNlmeJob.R' 'acceptAllEffects.R'
 'addInfusion.R' 'addLabel.R' 'addTablesToColumnMapping.R'
 'add_input_dosingCycles.R' 'scmArchiveHelpers.R'
 'readFitSummary.R' 'jobCollectionHelpers.R'
 'loadBootstrapResult.R' 'bootstrap.r' 'bootstrapPrint.R'
 'built_in_models.r' 'cad_auth.R' 'checkHostParams.R'
 'colMapping.R' 'colMapping_validate.R' 'profilePrint.R'
 'profile_estimation.r' 'scmPrint.R' 'fitmodelHelperFunctions.R'
 'vpc.r' 'collectJob.R' 'copyModel.R' 'cov_validators.R'
 'covariateModel.R' 'covariateNames.R' 'createInitialMapping.R'
 'create_metamodel_from_model.R' 'create_model_from_metamodel.R'
 'create_model_info.R' 'data.r' 'dosing.r' 'editModel.R'
 'emaxmodel.R' 'extract_mmdl.R' 'fitmodel.R' 'fitmodelPrint.R'
 'fixedEffect.R' 'formatOutputForPirana.R'
 'generateCovarSearchArgsFile.R' 'getThetas.R'
 'get_omega_omegaSE.R' 'globals.R' 'hostParams.R'
 'imputeMissingCovariates.R' 'isPopulation.R' 'job.r'
 'jobLaunchHelpers.R' 'linearmodel.R' 'log_Execution.R'
 'map_covariates.R' 'map_dosepoints.R' 'mcp_analysis_plan.R'
 'mcp_blq.R' 'mcp_bootstrap_summary.R' 'mcp_compare.R'
 'mcp_covariate_search.R' 'mcp_data.R' 'mcp_deps.R'
 'mcp_execution.R' 'mcp_fit_spec.R' 'mcp_literature.R'
 'mcp_mmdl_scaffold.R' 'mcp_model_inspect.R' 'mcp_nonmem.R'
 'mcp_ofv_parity.R' 'mcp_perf.R' 'mcp_provider_runtime.R'
 'mcp_qc_readiness.R' 'mcp_repro_hooks.R' 'mcp_scm_summary.R'
 'mcp_sequential_lrt.R' 'mcp_textual_inspect.R'
 'mcp_tools_register.R' 'mcp_update_parameters.R'
 'mcp_validate.R' 'mcp_vpc_summary.R' 'mcp_workflow.R'
 'mmdl_io.R' 'modelVariableNames.R' 'model_designer.R'
 'null_default.R' 'observation.r' 'obtain_NLMELicense.R'
 'parameterNames.R' 'parseControlFile.R' 'parsePMLColMap.R'
 'parse_mmdl.R' 'pkemaxmodel.R' 'pkindirectmodel.R'
 'pklinearmodel.R' 'pkmodel.R' 'randomEffect.R'
 'run_metamodel.R' 'saveModel.R' 'saveUpdatedMetamodel.R'
 'scmAccess.R' 'scmSearchTable.R' 'scmSummary.R'
 'secondary_variable.r' 'shotgunSearch.R' 'simParams.R'
 'sortfit.R' 'stepwiseSearch.R' 'structural_param.r'
 'tableParams.R' 'update_PMLwithThetas.R' 'update_parameters.R'
 'writeColumnMapping.R' 'writeDefaultFiles.R'

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author James Craig [aut, cre],
 Michael Tomashevskiy [aut],
 Vitalii Nazarov [aut],
 Shuhua Hu [ctb],
 Fred Soltanshahi [aut],

Certara USA, Inc. [cph, fnd]

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

Repository CRAN

Date/Publication 2026-09-15 02:20:41 UTC

Contents

addADDL	4
addCovariate	5
addDoseCycle	8
addExtraDef	9
addInfusion	10
addLabel	11
addMDV	12
addReset	12
addSecondary	13
addSteadyState	14
bootstrap	15
cancelJob	18
collectJob	18
colMapping	21
copyModel	22
covariateNames	23
createModelInfo	24
dataMapping	25
doseNames	25
editModel	26
emaxmodel	27
engineParams	28
extraDoseLines	35
extraDoseNames	36
fitmodel	37
getRandomEffectNames	41
getSCMResults	42
getThetas	43
hostParams	44
imputeMissingCovariates	47
imputeMode	50
initFixedEffects	51
isPopulation	52
linearmodel	53
listCovariateEffectNames	54
listSCMRuns	55
modelVariableNames	56
obtain_NLMELicense	57
OneCpt_IVInfusionData	58
parsePMLColMap	59

pkcovbqlData	59
pkData	60
pkemaxmodel	61
pkindirectmodel	65
pklinearmodel	70
pkmodel	75
pkpdData	78
print.NlmePmlModel	79
randomEffect	79
read_mmdl	81
removeCovariate	82
remove_NLMELicense	83
residualEffectNames	84
scmSearchTable	84
secondaryParameterNames	85
shotgunSearch	86
simmodel	88
sortfit	91
stepwiseSearch	94
structuralParameter	97
structuralParameterNames	99
tableParams	99
textualmodel	101
update_Omegas	102
update_Sigmas	103
update_Thetas	105
vpcmodel	106
write_mmdl	109

Index 110

addADDL	<i>Adds ADDL extra column definition to model object</i>
---------	--

Description

Specify ADDL column definition in model object instead of specifying ADDL through [addDoseCycle](#)

Usage

```
addADDL(.Object, ADDL, II)
```

Arguments

.Object	Model object
ADDL	Column mapping argument specifying corresponding "ADDL" column in input data set
II	Column mapping argument specifying corresponding "II" column in input data set

Value

Modified NlmePmlModel object

Examples

```
pkData1 <- pkData
pkData1$ii <- 0
pkData1$addl <- 0
model <- pkmodel(numComp = 2,
  absorption = "FirstOrder",
  ID = "Subject",
  Time = "Act_Time",
  CObs = "Conc",
  Aa = "Amount",
  data = pkData1,
  modelName = "PkModel",
  workingDir = tempdir())
model <- addADDL(model, ADDL = "addl", II = "ii")
```

addCovariate

Add covariate to model object

Description

Add a continuous, categorical, or occasion covariate to model object and set covariate effect on structural parameters.

Usage

```
addCovariate(
  .Object,
  covariate,
  effect = NULL,
  type = c("Continuous", "Categorical", "Occasion"),
  direction = c("Forward", "Interpolate", "Backward"),
  option = c("Yes", "PlusOne", "No"),
  center = NULL,
  centerValue = NULL,
  levels = NULL,
  labels = NULL,
  isDiagonal = TRUE,
  values = NULL,
  isPositive = TRUE
)
```

Arguments

.Object	Model object
covariate	Name of covariate. If the involved model has columns mapped (i.e. model with columnMap = TRUE) use named character if the name of the covariate is different from the corresponding column in the input dataset, for example, covariate = c(BW = "BodyWeight"), where BW denotes the name of the covariate, and "BodyWeight" is the name of the corresponding column in the input dataset.
effect	Name of structural parameter(s) on which the covariate has an effect. Specify effect as character or character vector if the covariate has an effect on multiple structural parameters. Important for Occasion Covariates: When modifying an <i>existing</i> occasion covariate (e.g., changing option or values), you must list <i>all</i> structural parameters currently affected by this occasion covariate in the effect argument, even those whose effect relationship is not being changed in this specific call.
type	Type of covariate. Options are "Continuous", "Categorical", "Occasion".
direction	Direction of missing values propagation (if no covariate value is given). Options are "Forward", "Interpolate", "Backward", where "Interpolate" is only applicable to type = "Continuous".
option	Options are "Yes", "PlusOne", or "No". • "Yes": Apply the covariate effect using the standard method (multiplicative for LogNormal style, additive for Normal style). • "PlusOne": Apply the covariate effect using the "1 + effect" formulation. This is only applicable to continuous and categorical covariates where the affected structural parameter has style = "LogNormal". • "No": Remove the specified covariate effect from the specified structural parameter(s). The covariate itself (and its definition, e.g., fcovariate(Occ1)) remains part of the model, but the link between this covariate and the specified parameter(s) in the stparm() statement is removed. See the note for the effect argument regarding occasion covariates. Multiple options are not supported within a single call (i.e., all covariate effects listed in the effect argument for a single call must use the same option). If different options are required, use sequential calls to addCovariate.
center	Centering method. Options are "Mean", "Median", "Value" or "None". Only applicable to covariate type = "Continuous". Must include argument centerValue if center = "Value".
centerValue	Value used to center covariate. Only applicable if argument center = "Value" and type = "Continuous".
levels	Unique values of categorical or occasion covariate. Only applicable to covariate type = "Categorical" or type = "Occasion".
labels	Label names (in the same order as levels) for unique levels of categorical or occasion covariate in data. Only applicable to covariate type = "Categorical" or type = "Occasion" where its corresponding column in the input dataset has character type.
isDiagonal	Set to FALSE if inter-occasion covariance matrix is not diagonal matrix. Only applicable to covariate type = "Occasion".

values	Initial values for the diagonal elements of the inter-occasion covariance matrix (if isDiagonal = TRUE) or initial values for the lower triangular elements (including diagonal elements) of inter-occasion covariance matrix (if isDiagonal = FALSE) in a row-wise order. Only applicable for covariate type = "Occasion".
isPositive	Set to FALSE if covariate contains negative values. Only applicable to covariate type = "Continuous".

Details

The following relationships are applicable for covariates:

- direction = "Forward" is equivalent to PML code 'fcovariate(CovName)';
- direction = "Backward" is equivalent to PML code 'covariate(CovName)';
- direction = "Interpolate" is equivalent to PML code 'interpolate(CovName)'.
If the structural parameter has style = "LogNormal", the options are reflected in PML code as follows:
 - option = "Yes" is equivalent to `stparm(V = tvV * wt^dVdwt * exp(dVdsex1*(sex==1)) * exp(nV))`;
 - option = "PlusOne" is equivalent to `stparm(V = tvV * (1+wt*dVdwt) * (1+dVdsex1*(sex==1)) * exp(nV))`.

Value

Modified `NlmePmlModel` object

Examples

```
model <- pkmodel(
  numCompartments = 2,
  data = pkData,
  ID = "Subject",
  Time = "Act_Time",
  A1 = "Amount",
  CObs = "Conc",
  workingDir = tempdir()
)

# Add Gender covariate of type categorical
model <- addCovariate(model,
  covariate = "Gender",
  type = "Categorical",
  effect = c("V2", "Cl2"),
  levels = c(0, 1),
  labels = c("Female", "Male")
)

# Add BodyWeight covariate of type continuous
model <- addCovariate(model,
  covariate = "BodyWeight",
  type = "Continuous",
```

```

    direction = "Backward",
    center = "Mean",
    effect = c("V", "C1")
  )

```

addDoseCycle	<i>Adds a dosing cycle to model</i>
--------------	-------------------------------------

Description

Add Steady State or ADDL dosing cycle to model object.

Usage

```

addDoseCycle(
  .Object,
  type = "SteadyState",
  name,
  administration = "Bolus",
  amount = NULL,
  II = NULL,
  rate = NULL,
  duration = NULL,
  isSecondDose = FALSE,
  colName = NULL
)

```

Arguments

.Object	Model object
type	Specification of dose type. Options are "SteadyState" and "ADDL"
name	Dose point name. See doseNames
administration	Mechanism for administering dose. Options are "Bolus" or "Infusion"
amount	Optional. Column mapping argument specifying corresponding "ADDL" column in input data, or numeric value specifying dose amount.
II	Optional. Column mapping argument specifying corresponding "II" column in input data, or numeric value specifying delta time.
rate	Optional. Column mapping argument specifying corresponding "Rate" column in input data, or numeric specifying dose rate.
duration	Optional. Column mapping argument specifying corresponding "Duration" column in data, or numeric specifying duration value.
isSecondDose	Use second dose point on compartment
colName	Column name in input data corresponding to column mapping for "SteadyState" or "ADDL" as supplied in type argument.

Value

Modified NlmePmlModel object

See Also

[doseNames](#)

Examples

```
model <- addDoseCycle(pkmodel(columnMap = FALSE,
                             isPopulation = FALSE,
                             workingDir = tempdir()),
                     type = "SteadyState",
                     name = "A1",
                     amount = "Amount",
                     II = "II")
```

addExtraDef	<i>Adds user defined extra column/table definitions to column definition file</i>
-------------	---

Description

Adds user defined extra column/table definitions to column definition file

Usage

```
addExtraDef(.Object, value)
```

Arguments

.Object	PK/PD model
value	Character vector of extra column/table definitions

Value

Modified NlmePmlModel object

Examples

```
model <- pkmodel(columnMap = FALSE,
                 workingDir = tempdir())
model <- addExtraDef(model, c("addlcol(ADDL)",
                             "table(file=\"res.csv\", time(0), Ka, V, Cl, Tlg)"))
```

addInfusion	<i>Change existing dosing compartment to infusion</i>
-------------	---

Description

Allows user to switch any dosing compartment to infusion

Usage

```
addInfusion(  
  .Object,  
  doseCptName,  
  isDuration = FALSE,  
  isSecondDose = FALSE,  
  colName = NULL  
)
```

Arguments

.Object	Model object
doseCptName	Name of the compartment to which the dose is administered
isDuration	Set TRUE if duration is used to specify infusion information
isSecondDose	Set TRUE if doseCptName is specified in the model through dosepoint2 statement
colName	Name of the input data column that represents the corresponding infusion rate. If not provided, colName must be mapped through colMapping().

Value

Modified NlmePmlModel object

Examples

```
pkData1 <- pkData  
pkData1$A1_1 <- 0  
model <- pkmodel(numComp = 2,  
  absorption = "Intravenous",  
  ID = "Subject",  
  Time = "Act_Time",  
  CObs = "Conc",  
  A1 = "Amount",  
  data = pkData1,  
  modelName = "PkModel",  
  workingDir = tempdir())  
newModel <- addInfusion(model, "A1", FALSE, FALSE, "A1_1")
```

addLabel

Add levels and labels to categorical or occasion covariate

Description

Allows users to specify the name and the associated value for each category/occasion of a categorical/occasion covariate in a textual model object. Only applicable to the case where the corresponding input data column of a categorical/occasion covariate is of class character.

Usage

```
addLabel(.Object, covariate, levels, labels)
```

Arguments

.Object	Model object
covariate	Existing covariate name
levels	Unique values of categorical or occasion covariate column specified as numeric vector
labels	Unique values specifying corresponding label names for levels of categorical or occasion covariate column in data specified as character vector.

Value

Modified NlmePmlModel object

Examples

```
model <- pkmodel(columnMap = FALSE,
                 isPopulation = FALSE,
                 workingDir = tempdir())

model <- suppressWarnings(addCovariate(model,
                                     covariate = "Gender",
                                     type = "Categorical",
                                     effect = c("V"),
                                     levels = c(0, 1)))

model@isTextual <- TRUE
model <- addLabel(model, "Gender", c(1, 2), c("male", "female"))
```

addMDV	<i>Adds MDV extra column definition to model object</i>
--------	---

Description

Use to add MDV statement to model@userDefinedExtraDefs

Usage

```
addMDV(.Object, MDV)
```

Arguments

.Object	Model object
MDV	Column mapping argument specifying corresponding "MDV" column in input data set

Value

Modified NlmePmlModel object

Examples

```
pkData1 <- pkData
pkData1$MDV <- 0
model <- pkmodel(data = pkData1,
                  ID = "Subject",
                  Time = "Act_Time",
                  A1 = "Amount",
                  CObs = "Conc",
                  workingDir = tempdir()
                  )
model <- addMDV(model, MDV = "MDV")
```

addReset	<i>Adds reset instructions to the model</i>
----------	---

Description

Adds reset instructions to the model

Usage

```
addReset(.Object, low, hi, Reset = NULL)
```

```
## S4 method for signature 'NlmePmlModel'
addReset(.Object, low, hi, Reset = NULL)
```

Arguments

<code>.Object</code>	An 'NlmePmlModel' object to which you want to add reset instructions.
<code>low</code>	Lower value of reset range.
<code>hi</code>	Upper value of reset range.
<code>Reset</code>	Name of reset column in input data set for column mapping. The default is NULL.

Value

Depends on the specific methods

Returns the 'NlmePmlModel' object with updated reset information and definitions.

Functions

- `addReset(NlmePmlModel)`: Method for the 'NlmePmlModel' class
This method adds reset instructions to the NlmePmlModel object. It updates the reset information, checks column mappings if input data is not null, and adds a reset definition to user-defined extra definitions.

<code>addSecondary</code>	<i>Adds a secondary parameter to model definition</i>
---------------------------	---

Description

Adds a secondary parameter to model definition

Usage

```
addSecondary(.Object, name, definition, unit = "")

## S4 method for signature 'NlmePmlModel'
addSecondary(.Object, name, definition, unit = "")
```

Arguments

<code>.Object</code>	An 'NlmePmlModel' object to which you want to add a secondary parameter.
<code>name</code>	Name of the secondary parameter.
<code>definition</code>	Definition of secondary parameter.
<code>unit</code>	Optional units of the secondary parameter. The default is "".

Value

Depends on the specific methods

Returns the 'NlmePmlModel' object with the added secondary parameter.

Functions

- addSecondary(NlmePmlModel): Method for the 'NlmePmlModel' class
This method adds a secondary parameter to the NlmePmlModel object. It checks for duplicate parameter names, and if there is no duplicate, it adds the new secondary parameter to the object and updates the PML model.

Examples

```
model <- pkmodel(columnMap = FALSE,
                 absorption = "FirstOrder",
                 workingDir = tempdir())
model <- addSecondary(model, "Ke", "tvCl/tvV")
model <- addSecondary(
  model, "Tmax",
  "CalcTMax(tvA,tvCl/tvV)"
)
```

addSteadyState	<i>Adds Steady State extra column definition to model object</i>
----------------	--

Description

Use to add Steady State column definition statement to model@userDefinedExtraDefs

Usage

```
addSteadyState(.Object, SS, II, SSOffset = NULL)
```

Arguments

.Object	Model object
SS	Column mapping argument specifying corresponding "SS" column in input data set
II	Column mapping argument specifying corresponding "II" column in input data set
SSOffset	Optional. Column mapping argument specifying corresponding "SSOffset" column in input data set

Value

Modified NlmePmlModel object

Examples

```
pkData1 <- pkData
pkData1$SS <- 0
pkData1$II <- 0
model <- pkmodel(data = pkData1,
                  ID = "Subject",
                  Time = "Act_Time",
                  A1 = "Amount",
                  CObs = "Conc",
                  workingDir = tempdir()
                  )
model <- addSteadyState(model, SS = "SS", II = "II")
```

bootstrap	<i>Executes an NLME Bootstrap</i>
-----------	-----------------------------------

Description

Method to execute an NLME Bootstrap

Usage

```
bootstrap(
  model,
  hostPlatform = NULL,
  params,
  bootParams,
  runInBackground = FALSE,
  saveResult = TRUE,
  overwriteFitDir = FALSE,
  ...
)
```

Arguments

model	PK/PD model class object.
hostPlatform	Host definition for model execution. See hostParams . If missing, multicore local host with 4 threads is used.
params	Engine parameters. See engineParams . If missing, default parameters generated by <code>engineParams(model)</code> are used.
bootParams	Bootstrap parameters. See BootstrapParams . If missing, default parameters generated by <code>BootstrapParams()</code> are used.
runInBackground	Logical. When TRUE, the wrapper starts the engine asynchronously and returns a job object immediately; pass that object to collectJob() when the run has

finished to obtain the typed result. When `FALSE` (the default), the wrapper blocks until the engine completes and returns the result directly.

Background execution is supported only on **Linux** hosts, whether local or remote: a local host whose `hostType` is "linux" (the default on Linux workstations), or a remote host with `hostType` "linux", "RHEL", or "UBUNTU". It is not supported on Windows (`hostType` = "windows", including the default local host when R runs on Windows): leave the argument at `FALSE`. Passing `TRUE` on a Windows host stops with an error. Remote Windows hosts are not supported at all.

<code>saveResult</code>	Logical; if <code>TRUE</code> (default), the returned <code>rsnlme_boot</code> is written to <code><workingDir>/bootstrap_<sanitizedModelName></code> so the run becomes self-describing on disk. <code>workingDir</code> is the model's working directory (<code>model@modelInfo@workingDir</code>), not necessarily R's process <code>getwd()</code> . <code>sanitizedModelName</code> is derived from <code>model@modelInfo@modelName</code> by replacing every character outside <code>[A-Za-z0-9._-]</code> with <code>_</code> ; if the result is empty or contains no alphanumerics, the literal string <code>model</code> is used instead. The timestamp is the wall-clock start of the engine call, formatted as <code>YYYYMMDD_HHMMSS</code> in the local time zone. For <code>runInBackground</code> = <code>TRUE</code> , the save happens at <code>collectJob()</code> time rather than when <code>bootstrap()</code> returns. A failing write produces a warning, never an error.
<code>overwriteFitDir</code>	Logical (default <code>FALSE</code>). <code>bootstrap()</code> writes its aggregate artifacts (<code>dmp.txt</code> , <code>residuals.csv</code> , <code>Boot*.csv</code>) into <code>model@modelInfo@workingDir</code> - NOT into <code>hostPlatform@sharedDirectory</code> , which does not redirect aggregate output. Passing a model whose <code>workingDir</code> already holds a fit's own <code>dmp.txt/residuals.csv</code> (e.g. a fitted model's <code>workingDir</code> taken as-is) would silently overwrite that fit's outputs with a single bootstrap replicate's estimates, corrupting any downstream GOF/VPC that re-reads the directory. By default <code>bootstrap()</code> detects this and reroots the run into an isolated <code><workingDir>/bootstrap/<run-id>/</code> subdirectory instead (via <code>copyModel</code>), leaving the original fit untouched. Set <code>overwriteFitDir</code> = <code>TRUE</code> to force running in-place and overwrite those files anyway (not recommended); this always emits a warning.
<code>...</code>	Additional class initializer arguments for <code>BootstrapParams</code> or <code>hostParams</code> , or arguments available inside <code>engineParams</code> functions. If <code>engineParams</code> arguments are supplied through both <code>params</code> argument and additional argument (i.e., ellipsis), then the arguments in <code>params</code> will be ignored and only the additional arguments will be used with warning. If <code>hostParams</code> arguments are supplied through both <code>hostPlatform</code> argument and additional argument, then its values will be overridden by additional arguments. In addition, if <code>BootstrapParams</code> arguments are supplied through both <code>bootParams</code> argument and additional argument, then its slots will be overridden by additional arguments.

Value

When `runInBackground` = `FALSE`, an `rsnlme_boot` object: a named list of bootstrap result tables with class `c("rsnlme_boot", "list")` and metadata attributes. The list carries the existing aggregates (`BootOverall`, `BootTheta`, `BootOmega`, `BootOmegaCorrelation`, `BootOmegaStderr`, `BootVarCoVar`, `BootSecondary`) plus the per-replicate stacks and per-parameter CI tables emitted by recent NLME8 builds (`BootThetaStacked`, `BootOmegaStacked`, `BootSigmaStacked`, `BootOmegaCI`,

BootSigmaCI, BootEtaShrinkage, BootEpsShrinkage). When `bootParams@initialEstimates` is TRUE, a compact fitSummary table built from `dmp.txt + nlme7engine.log` is also embedded. See `?rsnlme_boot` for the full field and attribute contract and `?print.rsnlme_boot` for the rendered output.

When `saveResult = TRUE` (the default), the `rsnlme_boot` is also written to disk; see the `saveResult` parameter for the exact path.

When `runInBackground = TRUE`, the bare `BootNlmeJob` is returned; pass it to `collectJob()` to obtain the `rsnlme_boot` (the RDS write, if requested, happens at that point).

See Also

[hostParams](#), [engineParams](#), [BootstrapParams](#), [rsnlme_boot](#), [collectJob](#)

Examples

```
## Not run:
input_data <- pkData

model <-
  pkmodel(
    numCompartments = 2,
    data = input_data,
    ID = "Subject",
    Time = "Act_Time",
    A1 = "Amount",
    CObs = "Conc",
    workingDir = tempdir()
  )

# multicore
multicoreHost <- hostParams(
  sharedDirectory = tempdir(),
  parallelMethod = "Multicore",
  hostName = "local_multicore",
  numCores = 4
)

bootstrapdf <- bootstrap(model,
  hostPlatform = multicoreHost,
  params = engineParams(model),
  numReplicates = 5,
  randomNumSeed = 1234,
  runInBackground = FALSE
)

## End(Not run)
```

cancelJob	<i>Generic function for cancelling a job</i>
-----------	--

Description

Generic function for cancelling a job

Usage

```
cancelJob(.Object)

## S4 method for signature 'SimpleNlmeJob'
cancelJob(.Object)
```

Arguments

.Object A 'SimpleNlmeJob' object that you want to cancel

Value

Depends on the specific methods

Prints the 'SimpleNlmeJob' object after attempting to cancel the job. No return value.

Functions

- `cancelJob(SimpleNlmeJob)`: Method for cancelling a job of the 'SimpleNlmeJob' class
This method attempts to cancel a job of the 'SimpleNlmeJob' class. If the job is running on a local host or is not running in the background, it throws an error and does nothing. Otherwise, it uploads a 'STOP' command to the host's remote executor.

collectJob	<i>Collect the typed result of a finished NLME job</i>
------------	--

Description

Polls a backgrounded NlmeJob via [NlmeJobStatus](#) until the engine reports "Finished", then materialises the typed result. Remote-results download (when applicable) and CSV/log parsing are handled by the class-specific method body. `collectJob()` is the single canonical entry point for any `runInBackground = TRUE` run.

Usage

```
collectJob(job, timeout = Inf, pollInterval = 5, ...)

## S4 method for signature 'BootNlmeJob'
collectJob(job, timeout = Inf, pollInterval = 5, ...)

## S4 method for signature 'FitNlmeJob'
collectJob(job, timeout = Inf, pollInterval = 5, ...)

## S4 method for signature 'StepwiseNlmeJob'
collectJob(job, timeout = Inf, pollInterval = 5, ...)

## S4 method for signature 'ShotgunNlmeJob'
collectJob(job, timeout = Inf, pollInterval = 5, ...)

## S4 method for signature 'ProfileNlmeJob'
collectJob(job, timeout = Inf, pollInterval = 5, ...)

## S4 method for signature 'SortByNlmeJob'
collectJob(job, timeout = Inf, pollInterval = 5, ...)

## S4 method for signature 'NlmeSimulationJob'
collectJob(job, timeout = Inf, pollInterval = 5, ...)
```

Arguments

job	An NLME job object returned by a wrapper invoked with <code>runInBackground = TRUE</code> (bootstrap , fitmodel , stepwiseSearch , shotgunSearch , profilePertubate , sortfit , vpcmodel , simmodel).
timeout	Maximum time to wait, in seconds. Inf (the default) waits indefinitely.
pollInterval	Seconds between status checks. Defaults to 5.
...	Reserved for class-specific arguments; currently unused.

Details

Background jobs are only started on Linux hosts (local `hostType "linux"` or remote `"linux" / "RHEL" / "UBUNTU"`). Wrappers on Windows do not support `runInBackground = TRUE`; see `?fitmodel` for the platform rule shared by all job executors.

Dispatches on `BootNlmeJob`, `FitNlmeJob`, `StepwiseNlmeJob`, `ShotgunNlmeJob`, `ProfileNlmeJob`, `SortByNlmeJob`, and `NlmeSimulationJob` (the shared class behind both [vpcmodel](#) and [simmodel](#), discriminated by its validated `runMode` slot).

Value

A class-specific result object. For `BootNlmeJob`, an [rsnlme_boot](#). For `FitNlmeJob` and `SortByNlmeJob`, the same result list [fitmodel](#) / [sortfit](#) produces. For `StepwiseNlmeJob` and `ShotgunNlmeJob`, an `scmSearchResult` (zero-row when `Overall.csv` is missing). For `ProfileNlmeJob`, a `profileResult`

data frame (zero-row when Profile.csv is missing). For NlmeSimulationJob, the same named list of data.tables that [vpcmodel](#) / [simmodel](#) produces, with additive runMode/runTime/RsNLMEVersion list elements.

Functions

- `collectJob(BootNlmeJob)`: Wait for a backgrounded `BootNlmeJob` to finish and return its `rsnlme_boot`. Downloads remote artefacts idempotently when `file.path(localDir, "BootOverall.csv")` is absent, then reads the bootstrap CSVs from `job@localDir` into the typed `rsnlme_boot` (forwarding `job@boot` for replicate counts / confidence level and `job@argsList$engine` for the engine attribute) and – if `job@saveResult` is TRUE – writes the result to `<localDir>/bootstrap_<sanitizedModelName>_<YYYYMMDD_HHMMSS>.rds`, where `sanitizedModelName` is `job@model@modelInfo@modelName` with every character outside `[A-Za-z0-9._-]` replaced by `_` (falling back to `model` when empty) and the timestamp is `job@runStart` formatted as `YYYYMMDD_HHMMSS` in local time.
- `collectJob(FitNlmeJob)`: Wait for a backgrounded `FitNlmeJob` to finish and return the same result list that [fitmodel](#)'s foreground branch produces. Downloads remote artefacts idempotently when `file.path(localDir, "nlme7engine.log")` is absent, then reads engine outputs via `.get_resultList()` and – if `job@saveResult` is TRUE – writes the result to `<localDir>/fitmodel_<sanitizedModelName>_<YYYYMMDD_HHMMSS>.rds`, where `sanitizedModelName` is `job@model@modelInfo@modelName` with every character outside `[A-Za-z0-9._-]` replaced by `_` (falling back to `model` when empty) and the timestamp is `job@runStart` formatted as `YYYYMMDD_HHMMSS` in local time.
- `collectJob(StepwiseNlmeJob)`: Wait for a backgrounded `StepwiseNlmeJob` to finish and return the same `scmSearchResult` that [stepwiseSearch](#)'s foreground branch produces. Downloads remote artefacts idempotently when `file.path(localDir, "Overall.csv")` is absent, then reads the search results via `.read_covsearch_results()` (which archives the run folder when `job@archiveResults` is TRUE) and applies the stepwise-only BestScenario flag from `Stepwise.txt`. Returns an empty `scmSearchResult` (zero-row, `searchType` attribute set) when `Overall.csv` is missing.
- `collectJob(ShotgunNlmeJob)`: Wait for a backgrounded `ShotgunNlmeJob` to finish and return the same `scmSearchResult` that [shotgunSearch](#)'s foreground branch produces. Downloads remote artefacts idempotently when `file.path(localDir, "Overall.csv")` is absent, then reads the search results via `.read_covsearch_results()` (which archives the run folder when `job@archiveResults` is TRUE). Returns an empty `scmSearchResult` (zero-row, `searchType` attribute set) when `Overall.csv` is missing.
- `collectJob(ProfileNlmeJob)`: Wait for a backgrounded `ProfileNlmeJob` to finish and return the same `profileResult` that [profilePertubate](#)'s foreground branch produces. Downloads remote artefacts idempotently when `file.path(localDir, "Profile.csv")` is absent, then reads the profile table via `.read_profile_results()` and – if `job@saveResult` is TRUE – writes a self-describing `profile_<sanitizedModelName>_<YYYYMMDD_HHMMSS>.rds` alongside the result, where `sanitizedModelName` is `job@model@modelInfo@modelName` with every character outside `[A-Za-z0-9._-]` replaced by `_` (falling back to `model` when empty) and the timestamp is `job@runStart` formatted as `YYYYMMDD_HHMMSS` in local time. Returns an empty `profileResult` (zero-row, `runMode = "profile"` attribute set) when `Profile.csv` is missing.
- `collectJob(SortByNlmeJob)`: Wait for a backgrounded `SortByNlmeJob` to finish and return the same result list that [sortfit](#)'s foreground branch produces. Downloads remote arte-

facts idempotently when `file.path(localDir, "nlme7engine.log")` is absent, then reads engine outputs via `.get_resultList()` (the shared `fitmodel/sortfit` reader) with `runMode = "sortfit"` and – if `job@saveResult` is `TRUE` – writes the result to `<localDir>/sortfit_<sanitizedModelName>_<YYYYMMDD_HHMMSS>` where `sanitizedModelName` is `job@model@modelInfo@modelName` with every character outside `[A-Za-z0-9._-]` replaced by `_` (falling back to `model` when empty) and the timestamp is `job@runStart` formatted as `YYYYMMDD_HHMMSS` in local time.

- `collectJob(NlmeSimulationJob)`: Wait for a backgrounded `NlmeSimulationJob` (constructed by either `vpcmodel` or `simmodel`) to finish and return the same named list of `data.tables` the foreground branch produces. Sentinel is mode-specific: `predout.csv` when `job@params@isPopulation` is `TRUE` (population runs), otherwise `simout.csv` (individual runs). Reads the engine output via `.read_vpcsim_results()` and attaches the canonical `runMode / runTime / RsNLMEVersion` list elements.

See Also

[bootstrap](#), [fitmodel](#), [stepwiseSearch](#), [shotgunSearch](#), [profilePertubate](#), [sortfit](#), [vpcmodel](#), [simmodel](#), [NlmeJobStatus](#)

colMapping	<i>Add column mappings</i>
------------	----------------------------

Description

Piping compatible function for `modelColumnMapping` used to add column mappings from input data to model object

Usage

`colMapping(.Object, mappings = NULL, ...)`

Arguments

<code>.Object</code>	Model (<code>NlmePmlModel</code>) object
<code>mappings</code>	Named character vector specifying valid column names in the input data. Character vector names must be valid model variable names contained in <code>modelVariableNames(model)</code> .
<code>...</code>	optional pairs <code>ModelTerm = ColumnName</code> or <code>ModelTerm = "ColumnName"</code> . Has higher precedence than <code>mappings</code> if some <code>ModelTerm</code> is mapped twice in <code>mappings</code> and in <code>...</code> . For multiple mapping, i.e. id mapping, a vector should be provided with the names of columns. See example below.

Value

modified `NlmePmlModel` object

See Also

[dataMapping](#) [modelVariableNames](#)

Examples

```
pkData$id2 <- pkData$Subject
model <- pkmodel(columnMap = FALSE,
                 data = pkData,
                 workingDir = tempdir())

modelvar <- unlist(modelVariableNames(model))

colnames <- c("Subject", "Act_Time", "Amount", "Conc")
names(colnames) <- modelvar
# will map subject directly
colnames <- colnames[-c(1)]

model <- colMapping(model, colnames, id = c(Subject, id2))
# also possible:
model <- colMapping(model, colnames, id = c("Subject", "id2"))
# not recommended since only not quoted names are identified
# if both types are provided:
model <- colMapping(model, colnames, id = c("Subject", id2))
```

copyModel

Copy model object to iterate over base model

Description

Copies previously executed model into a new object and optionally accept all estimates returned from model execution. A new working directory is created and all files from base model are copied into it.

Usage

```
copyModel(model, acceptAllEffects = FALSE, modelName = "", workingDir = "")
```

Arguments

model	Model object to be copied
acceptAllEffects	Set to TRUE to accept all effects, update PML statements, and test.mdl file from original model run
modelName	New model name for subdirectory created for model output. Subdirectory is created in current working directory.
workingDir	Working directory to run the model. Current working directory will be used if workingDir not specified.

Value

Modified NlmePmlModel object

Examples

```
## Not run:
model <- pkmodel(
  parameterization = "Clearance",
  numCompartments = 2,
  data = pkData,
  ID = "Subject",
  Time = "Act_Time",
  A1 = "Amount",
  CObs = "Conc",
  workingDir = tempdir()
)
host <- hostParams(sharedDirectory = tempdir(),
  parallelMethod = "None",
  hostName = "local",
  numCores = 1)
job <- fitmodel(model,
  numIterations = 3,
  hostPlatform = host)

finalModelVPC <- copyModel(model,
  acceptAllEffects = TRUE,
  modelName = "model_VPC",
  workingDir = tempdir())

## End(Not run)
```

covariateNames	<i>Return covariate names</i>
----------------	-------------------------------

Description

Use to return character vector of covariate names available in model object.

Usage

```
covariateNames(model)
```

Arguments

model	Model object
-------	--------------

Value

Character vector of covariate names defined in model

Examples

```
model <- pkmodel(columnMap = FALSE,
                 workingDir = tempdir())
model <- addCovariate(model, covariate = "BW", effect = "V")
model <- addCovariate(model, covariate = "Age", effect = "Cl")

covariateNames(model)
```

createModelInfo	<i>Parse the model and get the list of terms</i>
-----------------	--

Description

Calls TDL5 to parse the model and get the list of terms

Usage

```
createModelInfo(model, ForceRun = FALSE)
```

Arguments

model	Model object
ForceRun	Set to TRUE to force run

Value

List of model information

Examples

```
## Not run:
model <- pkmodel(columnMap = FALSE,
                 workingDir = tempdir())
createModelInfo(model)

## End(Not run)
```

dataMapping	<i>Initialize input data for PK/PD model</i>
-------------	--

Description

Used to initialize input data for PK/PD model

Usage

```
dataMapping(.Object, data)
```

Arguments

.Object	Model object
data	Input data of class <code>data.frame</code> .

Value

Modified `NlmePmlModel` object

See Also

[colMapping](#)

Examples

```
model <- pkmodel(columnMap = FALSE,  
                 workingDir = tempdir())  
  
model <- dataMapping(model, pkData)
```

doseNames	<i>Return dose names</i>
-----------	--------------------------

Description

Use to return character vector of dose point names in model object.

Usage

```
doseNames(model)
```

Arguments

model	Model object
-------	--------------

Value

Character vector of dose names defined in model

Examples

```
model <- pkmodel(columnMap = FALSE,
                 workingDir = tempdir())

doses <- doseNames(model)
```

editModel

Directly edit PML text in model object

Description

Allows user to edit PML text in model object using internal text editor and return a new textual model containing the edited PML statements.

Usage

```
editModel(.Object)
```

Arguments

.Object Model object

Value

Modified NlmePmlModel object

Examples

```
model <- pkmodel(columnMap = FALSE,
                 workingDir = tempdir())

if (FALSE) {
  # will open an additional window with the model text:
  newModel <- editModel(model)
}
```

 emaxmodel

 Create an Emax or Imax model

Description

Use to create an Emax or Imax model

Usage

```
emaxmodel(
  isPopulation = TRUE,
  checkBaseline = FALSE,
  checkFractional = FALSE,
  checkInhibitory = FALSE,
  checkSigmoid = FALSE,
  data = NULL,
  columnMap = TRUE,
  modelName = "",
  workingDir = "",
  ...
)
```

Arguments

isPopulation	Is this a population model TRUE or individual model FALSE?
checkBaseline	Set to TRUE if the model contains a baseline response.
checkFractional	Set to TRUE to modify the default form for the model. Only applicable to models with checkBaseline = TRUE.
checkInhibitory	Set to TRUE to change the model from an Emax to an Imax model.
checkSigmoid	Set to TRUE to change the model to its corresponding sigmoid form.
data	Input dataset
columnMap	If TRUE (default) column mapping arguments are required. Set to FALSE to manually map columns after defining model using colMapping .
modelName	Model name for subdirectory created for model output in current working directory.
workingDir	Working directory to run the model. Current working directory will be used if workingDir not specified.
...	Arguments passed on to emaxmodel_MappingParameters
ID	Column mapping argument for input dataset column(s) that identify individual data profiles. Only applicable to population models isPopulation = TRUE.

- C Column mapping argument that represents the input dataset column for the independent variable that is treated as a covariate during the estimation/simulation process.
- EObs Column mapping argument that represents the input dataset column for the observed drug effect (i.e., the dependent variable).

Value

NlmePmlModel object

Column mapping

Note that quoted and unquoted column names are supported. Please see [colMapping](#).

Examples

```
model <- emaxmodel(data = pkpdData, ID = "ID", C = "CObs", EObs = "EObs")

model <- emaxmodel(
  checkBaseline = TRUE,
  checkFractional = TRUE,
  checkInhibitory = TRUE,
  data = pkpdData,
  ID = "ID",
  C = "CObs",
  EObs = "EObs"
)

# View PML Code
print(model)
```

engineParams

Main function to specify engine parameters

Description

Use to define engine parameters for model execution.

Usage

```
engineParams(
  model,
  sort = NULL,
  ODE = "MatrixExponent",
  rtolODE = 1e-06,
  atolODE = 1e-06,
  maxStepsODE = 50000,
  numIterations = 1000,
```

```

method = NULL,
stdErr = NULL,
isCentralDiffStdErr = TRUE,
stepSizeStdErr = NULL,
conditionNumber = c("CovarianceFixef", "CorrelationFixef", "CovarianceFull",
  "CorrelationFull"),
logTransform = NULL,
numIntegratePtsAGQ = 1,
numIterNonParametric = 0,
allowSyntheticGradient = FALSE,
fastOptimization = FALSE,
numIterMAPNP = 0,
numRepPCWRES = 0,
stepSizeLinearize = 0.002,
numDigitLaplacian = 7,
numDigitBlup = 13,
gradTolOuter = 2e-04,
stepTolOuter = 1e-04,
gradTolInner = 1.71e-05,
stepTolInner = 7.07e-08,
refDeltaLag1 = 0.001,
mapAssist = 0,
iSample = 300,
iAcceptRatio = 0.1,
impDist = "Normal",
tDOF = 4,
numSampleSIR = 10,
numBurnIn = 0,
freezeOmega = FALSE,
MCPEM = FALSE,
runAllIterations = FALSE,
scramble = "Owen",
emTolType = 0,
emConvLen = 10,
emConvCritVal = 5,
stepSizePartialDeriv = 1e-05,
numTimeStepPartialDeriv = 20
)

```

Arguments

model	Model object. The type of model (population or individual) is determined by the <code>model@isPopulation</code> slot. If <code>model@isPopulation</code> is TRUE, the model is treated as a population model; otherwise, it's treated as an individual model.
sort	Logical; Specifies whether to sort the input data by subject and time. If TRUE, data are sorted. If FALSE, data are not sorted. Defaults to FALSE if the model contains reset information (<code>model@hasResetInfo</code> = TRUE); otherwise, defaults to TRUE.

ODE	Character; Specifies the ODE solver to be used. Options are: "MatrixExponent", "DVERK", "DOPRI5", "AutoDetect", "Stiff", "LSODE". See Details section for a description of each solver.
rtoLODE	Numeric; Specifying relative tolerance for the numerical ODE solver.
atolODE	Numeric; Specifying absolute tolerance for the numerical ODE solver.
maxStepsODE	Numeric; Specifies the maximum number of steps allowed for the ODE solver.
numIterations	Integer; Specifies the maximum number of iterations for the estimation algorithm. Must be a non-negative integer, with a maximum value of 10000.
method	Character; Specifies the estimation method. For population models, options are: "QRPEM", "IT2S-EM", "FOCE-LB", "FO", "FOCE-ELS", "Laplacian", and "Naive-Pooled". For individual models, only "Naive-Pooled" is available. The default for population models depends on model characteristics: • If the model includes discontinuous observed variables (LL/multi/count/event), Below Quantifiable Limit (BQL) data, all residual SDs frozen, or no error() statement, FOCE-ELS is unavailable. The default is "Laplacian"; "QRPEM", "IT2S-EM", and "Naive-Pooled" are equally available. • Otherwise, the default is "FOCE-ELS".
stdErr	Character; Specifies the method for standard error computations. Options vary depending on the model type and estimation method: • Individual models: "Hessian" (default) or "None". • Population models with method = "QRPEM": "Fisher-Score" (default) or "None". • Population models with method = "IT2S-EM": "None" only. • Population models with method in c("FOCE-LB", "FO", "FOCE-ELS", "Laplacian", "Naive-Pooled"): "Sandwich" (default), "Hessian", "Fisher-Score", "Auto-Detect", or "None". "None" means that standard error calculations are not performed.
isCentralDiffStdErr	Logical; If TRUE (default), uses central difference for standard error calculations when applicable. If FALSE, uses forward difference.
stepSizeStdErr	Numeric; Specifies the relative step size used for the numerical computation of the Hessian matrix during standard error calculations. If not specified, a default value is used (0.001 for "Naive-Pooled" method, and 0.01 otherwise).
conditionNumber	Character; Selects the basis and scope used to compute the reported condition number when standard errors are produced. One of: "CovarianceFixef" (Default) Covariance matrix of the estimated fixed effects only. "CorrelationFixef" Correlation matrix of the estimated fixed effects only. "CovarianceFull" Covariance matrix of all estimated population parameters (fixed effects, standard deviations of residual errors, Omega). "CorrelationFull" Correlation matrix of all estimated population parameters. This is the only option directly comparable to the condition number calculated from NONMEM's reported eigenvalues, which are computed over all estimated THETA/OMEGA/SIGMA parameters.

This argument is independent of `stdErr`: it does not change which standard-error covariance block is computed, only how the reported eigenvalues and condition lines (in the engine's `out.txt`) are derived from that block. The basis actually used is echoed there as a `# conditionNumberBasis = ...` comment. See **Details** for the formulas.

logTransform Logical or NULL; Controls log-transformation behavior, particularly for models with a LogAdditive residual error (e.g., $C \cdot \exp(\epsilon)$). The internal engine parameter 'logtran' is set based on this argument and specific model characteristics as detailed below.

- NULL (default) or TRUE: When the model has exactly one residual error model and it is LogAdditive, this setting enables Log-Transform Both Sides (LTBS). In LTBS, predictions and observations are log-transformed, and the model is fit in the log-domain. This results in the internal `logtran` engine parameter being set to 1.
- FALSE: When the model has exactly one residual error model and it is LogAdditive, this setting results in the LogAdditive error being treated as a proportional/multiplicative error during fitting (by neglecting third and higher-order terms in the Taylor expansion of $\exp(\epsilon)$). This sets the internal `logtran` engine parameter to 0. For simulation, the error is treated as $\exp(\epsilon)$.

For other model configurations, the `logtran` parameter is determined as follows:

- If there are multiple residual error models or no residual error models, `logtran` is set to 0, irrespective of the `logTransform` value. (In the case of multiple errors, any LogAdditive errors present are treated as proportional).
- If there is a single residual error model that is not LogAdditive:
 - For built-in models: `logtran` is set to 0.
 - For textual models: `logtran` reflects the `logTransform` setting (it becomes 1 if `logTransform` is NULL or TRUE, and 0 if `logTransform` is FALSE). A warning is issued if `logTransform` is NULL or TRUE in this scenario, highlighting that LTBS is typically for LogAdditive errors and that error type identification can be challenging in textual models.

numIntegratePtsAGQ

Integer; Specifies the number of quadrature points per dimension to use for Adaptive Gaussian Quadrature (AGQ). Only applicable to population models when method is "FOCE-ELS" or "Laplacian".

- 1: Standard FOCE-ELS/LAPLACIAN computation (no AGQ).
- >1: AGQ is performed. The total number of quadrature points used is $(\text{number of ETAs})^{\text{numIntegratePtsAGQ}}$.

numIterNonParametric

Integer; Controls non-parametric (NP) optimization.

- 0: Disables NP optimization.
- 1: Enables NONMEM-style NP optimization using posthoc estimates as support points.
- >1: Enables an evolutionary NP algorithm, using `numIterNonParametric` as the number of generations.

	Only applicable to population models when method is not "Naive-Pooled".
allowSyntheticGradient	Deprecated.
fastOptimization	Controls the use of Automatic Differentiation (AD) during optimization. Accepts a logical value for backward compatibility or a character string specifying the AD mode. Only applicable to population models when method is "FOCE-ELS" or "Laplacian". Accepted values: FALSE (Default) Disables automatic differentiation. Finite differences are used for all gradient and Hessian computations. TRUE Enables inner-loop automatic differentiation (equivalent to "InnerAD"). Provided for backward compatibility. "InnerAD" Enables automatic differentiation for the inner optimization loop (random effects / etas). "OuterAD" Enables inner-loop AD plus population-level (outer-loop) automatic differentiation.
numIterMAPNP	Integer; Specifies the number of iterations for a preliminary Naive-Pooled (NP) optimization run before the main estimation. Applicable when the method is not "NAIVE-POOLED".
numRepPCWRES	Integer; Specifies the number of replicates to generate for Population Conditional Weighted Residuals (PCWRES) calculations. Setting this value to 0 disables PCWRES computation. Only applicable to population models when method is not set to "Naive-Pooled".
stepSizeLinearize	Numeric; Specifies the relative step size for numerical differentiation during model linearization.
numDigitLaplacian	Numeric; Specifies the optimization accuracy (NDIGIT) for the outer loop (thetas and sigmas) when using "FOCE-ELS" or "Laplacian" methods. Only applicable to population models.
numDigitBlup	Numeric; Specifies the optimization accuracy (NDIGIT) for the inner loop (optimization of etas). Also applies to the single optimization loop in the "NAIVE-POOLED" method.
gradTolOuter	Numeric; maximum gradient tolerance for the outer loop (Theta/Omega/Sigma optimization) of "FOCE-ELS" or "Laplacian" method. This tolerance controls how close the gradient must be to zero before the outer optimization is considered converged.
stepTolOuter	Numeric; maximum step tolerance for the outer loop (Theta/Omega/Sigma optimization) of "FOCE-ELS" or "Laplacian" method. This measures the relative change in the solution vector between iterations.
gradTolInner	Numeric; maximum gradient tolerance for the inner loop (Eta optimization) of "FOCE-ELS" or "Laplacian" method. A smaller value forces the algorithm to iterate until a very small gradient is achieved.

stepTolInner	Numeric; maximum step tolerance for the inner loop (Eta optimization) of "FOCE-ELS" or "Laplacian" method. This determines when the algorithm will terminate based on minimal changes in the solution vector.
refDeltaLag1	Numeric; tolerance for the change in the log-likelihood (LL) value during outer loop optimization of "FOCE-ELS" or "Laplacian" method. This parameter is used to check convergence by comparing the absolute change in LL between major iterations. If the change in LL is less than refDeltaLag1 and the optimization driver returns a specific termination code, the algorithm considers the solution sufficiently converged. This tolerance helps to avoid unnecessary iterations when improvements in LL become marginal.
mapAssist	Numeric; Controls the use of MAP assistance in the QRPEM algorithm. • 0: No MAP assistance. • >0: The inner ETAs optimization loop is used in the QRPEM outer optimization loop with a periodicity equal to the value of mapAssist. Only applicable to population models with method = "QRPEM".
iSample	Numeric; Specifies the number of sample points used in the QRPEM algorithm. Only applicable to population models with method = "QRPEM".
iAcceptRatio	Numeric; Specifies the acceptance ratio used in the QRPEM algorithm for scaling the covariance matrix. Only applicable to population models with method = "QRPEM".
impDist	Character; Specifies the importance sampling distribution used in the QRPEM algorithm. Options are: "Normal", "DoubleExponential", "Direct", "T", "Mixture-2", "Mixture-3". Only applicable to population models with method = "QRPEM". See Details for further information.
tDOF	Numeric; Specifies the degrees of freedom for the multivariate T distribution used in importance sampling. Only applicable when method = "QRPEM" and impDist = "T". Must be between 3 and 30.
numSampleSIR	Numeric; Specifies the number of samples per eta per subject used in the Sampling Importance Resampling (SIR) algorithm within QRPEM. Only applicable to population models with method = "QRPEM".
numBurnIn	Numeric; Specifies the number of burn-in iterations in the QRPEM algorithm. During burn-in, omegas can be frozen (see freezeOmega parameter). 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; Controls the sampling method used in the QRPEM algorithm. • FALSE: Quasi-Random sampling. • TRUE: Monte-Carlo sampling. 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; Specifies the scrambling method for quasi-random number generation in the QRPEM algorithm. Options are: "None", "Owen", "Faure-Tezuka". Only applicable to population models with method = "QRPEM".
emTolType	Numeric; QRPEM convergence check type. Options: • 0: Default (no rollout, LL & Theta and Sigma). • 1: LL & All Population Params (Theta, Omega, and Sigma) with rollout. • 2: LL with rollout. • 3: All Population Params with rollout. Only applicable to population models with method = "QRPEM".
emConvLen	Numeric; number of iterations over which convergence is checked in the QRPEM method. Only applicable to population models with method = "QRPEM" and emTolType being nonzero.
emConvCritVal	Numeric; critical value used in the QRPEM convergence check. It specifies the threshold improvement required to continue iterating. Only applicable to population models with method = "QRPEM" and emTolType being nonzero.
stepSizePartialDeriv	Numeric; Specifying the step size used to numerically calculate the partial derivatives of observed variables with respect to parameters. Only applicable to individual models.
numTimeStepPartialDeriv	Numeric; Specifying the number of time steps used to output the partial derivatives of observed variables with respect to parameters. Only applicable to individual models.

Details

Both "DVERK" and "DOPRI5" are non-stiff solvers. "AutoDetect" represents LSODA solver implementation, which solves the initial value problem for stiff or nonstiff systems of first order ordinary differential equations. "Stiff" is a LSODE (Livermore solver). It is best suited for stiff problems. "MatrixExponent" is a matrix exponential solver.

For the QRPEM method, the impDist parameter controls the importance sampling distribution. The ximpsampdof slot in the internal NlmeEngineExtraParams object is set based on impDist as follows:

- "Normal": ximpsampdof = 0
- "DoubleExponential": ximpsampdof = 1
- "Direct": ximpsampdof = 2
- "T": ximpsampdof is set to the value of tDOF.
- "Mixture-2": ximpsampdof = -2
- "Mixture-3": ximpsampdof = -3

The conditionNumber argument controls both the scope (which parameters) and the basis (covariance vs. correlation) used to derive the reported eigenvalues and condition in the engine's out.txt. Let C be the variance-covariance matrix of the selected parameters, as produced by the standard-error step (Hessian, sandwich, or Fisher-score, selected by stdErr); its diagonal entries are the squared standard errors of the estimates and its off-diagonals are their estimated covariances. Scope (which parameters make up C):

- "...Fixef": only the estimated fixed effects.
- "...Full": all estimated population parameters (fixed effects, standard deviations of residual errors, elements of variance/covariance matrix of the random effects).

Basis (how C is used):

- "Covariance...": (default scope is Fixef): eigenvalues are those of C directly, and `condition` = $\sqrt{\lambda_{\max} / \lambda_{\min}}$, where $\lambda_{\min}$ and $\lambda_{\max}$ are the smallest and largest strictly positive eigenvalues, respectively. This value is scale-dependent: parameters spanning many orders of magnitude (e.g. typical PK fixed effects) inflate it even when the fit is well-conditioned in correlation terms.
- "Correlation...": C is first normalised to the correlation matrix $R = D^{-1/2} * C * D^{-1/2}$, where $D = \text{diag}(C)$ (the parameter variances); reported eigenvalues are those of R and `condition` = $\lambda_{\max} / \lambda_{\min}$. The result is scale-invariant and reflects only the correlation structure of the estimates. If any diagonal of C is non-positive or non-finite the correlation matrix is undefined and both lines are suppressed.

Only "CorrelationFull" matches NONMEM's condition-number scope (all estimated THETA/OMEGA/SIGMA, correlation basis); use it for direct comparison with NONMEM output. The other three options are not NONMEM-comparable.

Value

List of engine parameters to be used during fitting or simulation

extraDoseLines	<i>Return extra dose lines</i>
----------------	--------------------------------

Description

Use to return extra dose lines for model object

Usage

```
extraDoseLines(model)
```

Arguments

model	Model object
-------	--------------

Value

List of extra dose information

Examples

```
data <- pkData
data$II <- 24
data$ADDL <- 1

model <-
pkmodel(
  parameterization = "Clearance",
  numCompartments = 2,
  data = data,
  ID = "Subject",
  Time = "Act_Time",
  A1 = "Amount",
  CObs = "Conc",
  workingDir = tempdir())
addDoseCycle(
  model,
  name = "A1",
  amount = 30000,
  II = 24,
  type = "ADDL",
  colName = "ADDL")

extraDoseLines(model)
```

extraDoseNames*Return extra dose names*

Description

Use to return extra dose names for model object

Usage

```
extraDoseNames(model)
```

Arguments

model	Model object
-------	--------------

Value

Character vector of extra dose names

Examples

```
data <- pkData
data$II <- 24
data$ADDL <- 1

model <-
pkmodel(
  parameterization = "Clearance",
  numCompartments = 2,
  data = data,
  ID = "Subject",
  Time = "Act_Time",
  A1 = "Amount",
  CObs = "Conc",
  workingDir = tempdir())
addDoseCycle(
  model,
  name = "A1",
  amount = 30000,
  II = 24,
  type = "ADDL",
  colName = "ADDL")

extraDoseNames(model)
```

fitmodel

Executes an NLME simple estimation

Description

Executes an NLME simple estimation

Usage

```
fitmodel(
  model,
  hostPlatform = NULL,
  params,
  simpleTables,
  runInBackground = FALSE,
  filesToReturn = "*",
  ...,
  saveResult = TRUE
)
```

Arguments

model	PK/PD model class object.
hostPlatform	Host definition for model execution. See hostParams . If missing, PhoenixM-PIDir64 is given and MPI is installed, MPI local host with 4 threads is used. If MPI is not found, local host without parallelization is used.
params	Engine parameters. See engineParams . If missing, default parameters generated by <code>engineParams(model)</code> are used.
simpleTables	Optional list of simple tables. See tableParams . By default a table named 'posthoc.csv' is returned with structural parameters values for all source data rows.
runInBackground	Logical. When TRUE, the wrapper starts the engine asynchronously and returns a job object immediately; pass that object to <code>collectJob()</code> when the run has finished to obtain the typed result. When FALSE (the default), the wrapper blocks until the engine completes and returns the result directly. Background execution is supported only on Linux hosts, whether local or remote: a local host whose <code>hostType</code> is "linux" (the default on Linux workstations), or a remote host with <code>hostType</code> "linux", "RHEL", or "UBUNTU". It is not supported on Windows (<code>hostType</code> = "windows", including the default local host when R runs on Windows): leave the argument at FALSE. Passing TRUE on a Windows host stops with an error. Remote Windows hosts are not supported at all.
filesToReturn	Used to specify which files to be outputted to the model directory and loaded as returned value. By default, all the applicable files listed in the Value section will be outputted to the model directory and loaded as returned value. Only those files listed in the Value section can be specified. Simple regex patterns are supported for the specification.
...	Additional arguments for hostParams or arguments available inside engineParams functions. If engineParams arguments are supplied through both <code>params</code> argument and additional argument (i.e., ellipsis), then the arguments in <code>params</code> will be ignored and only the additional arguments will be used with warning. If hostParams arguments are supplied through both the <code>hostPlatform</code> argument and the ellipses, values supplied to <code>hostPlatform</code> will be overridden by additional arguments supplied via the ellipses e.g., ...
saveResult	Logical; if TRUE (default), the returned list is written to <code><workingDir>/fitmodel_<sanitizedModelName></code> so the run becomes self-describing on disk. <code>sanitizedModelName</code> is derived from <code>model@modelInfo@modelName</code> by replacing every character outside <code>[A-Za-z0-9._-]</code> with <code>_</code>; if the result is empty or contains no alphanumerics, the literal string <code>model</code> is used instead. The timestamp is the wall-clock start of the engine call, formatted as <code>YYYYMMDD_HHMMSS</code> in the local time zone. For <code>runInBackground</code> = TRUE, the save happens at <code>collectJob()</code> time rather than when <code>fitmodel()</code> returns. A failing write produces a warning, never an error.

Value

if `runInBackground` is FALSE, a list with main resulted dataframes is returned:

- Overall
- ConvergenceData
- residuals
- Secondary
- StrCovariate - if continuous covariates presented
- StrCovariateCat - if categorical covariates presented
- theta
- posthoc table
- posthocStacked table
- Requested tables

`nlme7engine.log` textual output is returned and loaded with the main information related to fitting. `dmp.txt` structure with the results of fitting (including LL by subject information) is returned and loaded. These 2 files are returned and loaded irrespective of `filesToReturn` argument value.

For individual models, additional dataframe with partial derivatives is returned:

- ParDer

For population models and the method specified is NOT Naive-Pooled, additional dataframes are returned:

- omega
- Eta
- EtaStacked
- EtaEta
- EtaCov
- EtaCovariate - if continuous covariates presented
- EtaCovariateCat - if categorical covariates presented
- bluptable.dat

If standard error computation was requested and it was successful, additional dataframes are returned:

- thetaCorrelation
- thetaCovariance
- Covariance
- omega_stderr

If nonparametric method was requested (`numIterNonParametric > 0`) and the method specified in `engineParams` is NOT Naive-Pooled, additional dataframes are returned:

- nonParSupportResult
- nonParStackedResult
- nonParEtaResult
- nonParOverallResult

if `runInBackground` is TRUE, a `FitNlmeJob` object is returned. Pass it to `collectJob()` when the engine has finished to materialise the same list described above.

filesToReturn **with** Certara.Xpose.NLME

If filesToReturn is used and "ConvergenceData.csv" and "residuals.csv" are not in the patterns, these files won't be returned and loaded. These files are essential for Certara.Xpose.NLME::xposeNlmeModel and Certara.Xpose.NLME::xposeNlme functions. This makes impossible to use the resulted object in Certara.Xpose.NLME functions.

Non-loaded but returned files

The non-loaded but returned files in the model working directory are:

- err1.txt - concatenated for all runs detailed logs for all steps of optimization,
- out.txt - general pivoted information about results,
- doses.csv - information about doses given for all subjects,
- iniest.csv - information about initial estimates

Self-describing run context

The returned list also carries five run-context elements that make the object self-describing:

- model - the input NlmePmlModel as it was at the start of the run (with @modelInfo@workingDir pointing at the artifacts).
- params - the resolved NlmeEngineExtraParams actually used.
- runMode - the string "fitmodel".
- runTime - a list with start, end, and elapsed wall-clock times measured around the engine call. This is distinct from the engine-reported CPU time in nlme7engine.log.
- RsNLMEVersion - the package version that produced the result.

No host, machine name, user name, password, or private key path is ever placed in the returned value or in the saved file.

See Also

[tableParams](#), [hostParams](#), [engineParams](#), [collectJob](#)

Examples

```
## Not run:

# Define the host
host <- hostParams(sharedDirectory = tempdir(),
  parallelMethod = "None",
  hostName = "local",
  numCores = 1)

# Define the model
model <- pkmodel(numComp = 2,
  absorption = "FirstOrder",
  ID = "Subject",
  Time = "Act_Time",
  CObs = "Conc",
```

```

Aa = "Amount",
data = pkData,
modelName = "PkModel",
workingDir = tempdir())

Table01 <- tableParams(name = "SimTableObs.csv",
                        timesList = "0,1,2,4,4.9,55.1,56,57,59,60",
                        variablesList = "C, C0bs",
                        timeAfterDose = FALSE,
                        forSimulation = FALSE)

# Update fixed effects
model <- fixedEffect(model,
                     effect = c("tvV", "tvC1", "tvV2", "tvC12"),
                     value = c(16, 41, 7, 14))

# Define the engine parameters
params <- engineParams(model)
# Fit model
res <- fitmodel(model = model,
                hostPlatform = host,
                params = params,
                simpleTables = Table01)

## End(Not run)

```

getRandomEffectNames *Return random effect names in model*

Description

Use to return character vector of random effect names (if available) in model object

Usage

```
getRandomEffectNames(model)
```

Arguments

model	Model object
-------	--------------

Value

Characters vector of random effect names

Examples

```
model <- pkmodel(columnMap = FALSE,
                 workingDir = tempdir())
getRandomEffectNames(model)
```

getSCMResults	<i>Retrieve model and fitmodel-style results for a selected covariate search scenario.</i>
---------------	--

Description

Dispatches on the type of `x` to resolve an archived covariate-search run folder and scenario, then loads the base model and run context from `rsnlme_run.rds` and reconstructs the scenario-specific model. Use `listSCMRuns` to enumerate scenarios in one or more such folders.

Usage

```
getSCMResults(x, scenario = NULL, enableIDs = NULL)
```

Arguments

<code>x</code>	One of: • A search-result data frame (from <code>shotgunSearch()</code> / <code>stepwiseSearch()</code>) with a <code>searchRunDir</code> attribute. • A single-row data frame from <code>listSCMRuns()</code>. • An <code>NlmePmlModel</code> (resolves the latest archived run in its working directory). • A character path to an archived run folder (selects best scenario). • A character path to a scenario subfolder (uses that exact scenario).
<code>scenario</code>	Optional scenario identifier (label or token). Only allowed when <code>x</code> resolves to a run folder (not a specific scenario).
<code>enableIDs</code>	Optional comma-separated string or integer vector of enable IDs to select a scenario by its active PML <code>enable()</code> groups (matches the <code>enableIDs</code> column of <code>scenario_index.csv</code>). Only allowed when <code>x</code> resolves to a run folder.

Details

When `x` is a row-subset of a search-result data frame and the caller does not specify a scenario explicitly, selection is restricted to the subset: if the archive's flagged best scenario is filtered out, the row with the lowest value of the search's selection criterion ($-2LL$, AIC, or BIC; recorded for stepwise searches and always $-2LL$ for shotgun) *within the subset* is returned. For stepwise this need not match the engine's chosen winner, because the user has explicitly excluded that row by subsetting.

Value

An `scmResult` object (list-like, with a custom print method) containing three elements:

model The reconstructed `NlmePmlModel` for the scenario, with accepted parameter estimates.

fitmodelOutput A list compatible with `fitmodel()` output. In addition to the file-derived elements, this list carries the self-describing fields: `model` (the reconstructed scenario model), `params` (resolved `NlmeEngineExtraParams`), `runMode` ("stepwise" or "shotgun"), `runTime` (wall-clock list), and `RsNLMEVersion`.

scenarioInfo A single-row data frame with scenario metadata.

Archived run folders

Archived folders are created when `shotgunSearch()` or `stepwiseSearch()` is run with `archiveResults = TRUE`. During the engine run, NLME8 writes scenario outputs under a temporary `SCM_<runId>/` directory in the model working directory (`runId` is a wall-clock timestamp, optionally suffixed with `runLabel`). After the engine finishes, `collectJob()` materialises the search result and the package renames that tree into a durable folder named `stepwise_<runId>` or `shotgun_<runId>` (on remote hosts the bundle `SCM_<runId>.tar.gz` is extracted first, then the bundle and any loose `SCM_<runId>/` copy are removed). The final folder holds `scenario_index.csv`, per-scenario sub-folders, `Overall.csv`, optional stepwise logs, and `rsnlme_run.rds`.

See Also

[shotgunSearch](#), [stepwiseSearch](#), [summary\(\)](#), [print\(\)](#), [listSCMRuns](#)

getThetas

Return theta names and values

Description

Returns named character vector of theta values by parsing PML fixed effect statements

Usage

```
getThetas(model)
```

Arguments

`model` PK/PD model

Value

Character vector of theta names defined in model

Examples

```
## Not run:
model <- emaxmodel(
  checkBaseline = TRUE,
  checkFractional = TRUE,
  checkInhibitory = TRUE,
  data = pkpdData,
  ID = "ID",
  C = "CObs",
  EObs = "EObs"
)
getThetas(model)

## End(Not run)
```

hostParams

Create an NLME Parallel Host Configuration

Description

This helper function simplifies the creation and configuration of an `NlmeParallelHost` object, which defines the environment for running NLME jobs.

Usage

```
hostParams(
  sharedDirectory,
  installationDirectory = NULL,
  hostName = Sys.info()[["nodename"]],
  machineName = "127.0.0.1",
  hostType = Sys.info()[["sysname"]],
  numCores = 4,
  parallelMethod = "LOCAL_MPI",
  mpiCoresPerJob = NA_integer_,
  userName = "",
  privateKeyFile = NULL,
  userPassword = "",
  scriptPath = "",
  rLocation = "",
  isLocal = TRUE,
  gridQueue = "",
  gridWalltime = NA_integer_,
  gridMemory = "",
  gridJobNamePrefix = "",
  gridResourceExtra = ""
)
```

Arguments

sharedDirectory	character. The directory where temporary run folders are created. Defaults to the current working directory for local runs and to "~" for remote runs.
installationDirectory	character. The directory containing NLME libraries/scripts. Defaults to the INSTALLDIR environment variable for local runs and to file.path(sharedDirectory, "InstallDirNLME") for remote runs.
hostName	character. A display name for the host. Defaults to the system's network name (from Sys.info()[["nodename"]]) for local runs and to machineName for remote runs.
machineName	character. The IP address or network name of the host. Defaults to "127.0.0.1".
hostType	character. The host operating system. Defaults to the current OS (Sys.info()[["sysname"]]) for local runs and to "linux" for remote runs. While "windows" or "linux" are valid for local runs, for remote Linux hosts the following are officially supported: "RHEL" (for RHEL 8 and 9) and "UBUNTU" (for Ubuntu 22.04 and 24.04). Specifying one of these values correctly sets the PML_BIN_DIR variable.
numCores	numeric. The number of CPU cores to utilize. Defaults to 4.
parallelMethod	character. The parallel execution method. Options include: "None", "Multicore", "Multicore_MPI", "LOCAL_MPI", "SGE", "SGE_MPI", "TORQUE", "TORQUE_MPI", "LSF", "LSF_MPI", "SLURM", "SLURM_MPI". Defaults to "LOCAL_MPI".
mpiCoresPerJob	integer, optional. Per-scenario MPI rank for the "Multicore_MPI" parallel method. Leave at the default NA_integer_ to let NLME8 size the rank from the current batch (engine-aware auto policy). Set a positive integer to fix the per-job MPI width; it must divide numCores evenly. Ignored (with a warning) for other methods.
userName	character. The username for remote host authentication.
privateKeyFile	character. The path to an SSH private key file for remote authentication. See <code>ssh::ssh_connect()</code> for more details.
userPassword	character or function. The password or a callback function for remote authentication. See <code>ssh::ssh_connect()</code> for details.
scriptPath	character. The path to a script to run on a remote host before the main job starts. Ignored for local runs.
rLocation	character. The path to the Rscript executable on a remote host. Ignored for local runs.
isLocal	logical. Set to TRUE for a local host or FALSE for a remote host. Defaults to TRUE.
gridQueue	character, optional. Scheduler queue or partition to route grid jobs to (e.g. "bg.q" on SGE, "batch" on SLURM). This is the scheduler's queue/partition name for CPU jobs; it does not imply GPU or graphical execution. Empty (default) uses the cluster default queue. Used only by grid parallel methods ("SGE", "TORQUE", "LSF", "SLURM", and their _MPI variants); ignored (with a warning) otherwise.

gridWalltime	integer, optional. Wall-clock limit in seconds for grid jobs. Defaults to NA_integer_, which imposes no limit so long-running bootstrap/covariate-search work is not capped. Grid methods only.
gridMemory	character, optional. Memory request for grid jobs (e.g. "16G", mapped to SGE h_vmem / TORQUE vmem / SLURM --mem). Empty (default) requests no explicit memory. Grid methods only.
gridJobNamePrefix	character, optional. Prefix for the scheduler job name shown in qstat/queue. Empty (default) derives a name from the workflow and job type. Restricted to letters, digits, and underscores. Grid methods only.
gridResourceExtra	character, optional. Additional site-specific resource request, emitted verbatim using each scheduler's native syntax: appended to a resource list for the PBS-style schedulers (SGE #\$ -l <value>, TORQUE #PBS -l <value>, e.g. "h_data=4G"), or rendered as a standalone directive for SLURM (#SBATCH <value>, e.g. "--gres=gpu:1") and LSF (#BSUB <value>, e.g. "-R rusage[mem=4096]"). Empty (default) adds nothing. Grid methods only.

Details

Remote Windows hosts are not supported. Background execution (`runInBackground = TRUE` on `fitmodel()` and the other job wrappers) is available only on Linux hosts (local or remote); on Windows hosts use `runInBackground = FALSE`.

Value

An `NlmeParallelHost` object configured with the specified parameters.

Examples

```
host <- hostParams(sharedDirectory = tempdir(),
  parallelMethod = "LOCAL_MPI",
  hostName = "Local",
  numCores = 4)

# Local hybrid: run several models concurrently, each with its own MPI ranks.
# numCores = 10 with mpiCoresPerJob = 2 fits up to 5 models at a time,
# 2 MPI ranks each.
hybridHost <- hostParams(sharedDirectory = tempdir(),
  parallelMethod = "Multicore_MPI",
  hostName = "LocalHybrid",
  numCores = 10,
  mpiCoresPerJob = 2)

# Omit mpiCoresPerJob to let NLME8 size the per-job rank from the batch.
autoHybridHost <- hostParams(sharedDirectory = tempdir(),
  parallelMethod = "Multicore_MPI",
  hostName = "LocalHybridAuto",
  numCores = 10)
```

```
# Grid host routing long batch work to a non-interactive queue, with an
# informative job name shown in qstat/queue.
gridHost <- hostParams(sharedDirectory = "~",
                        parallelMethod = "SGE_MPI",
                        hostName = "sge",
                        numCores = 32,
                        gridQueue = "bg.q",
                        gridMemory = "16G",
                        gridJobNamePrefix = "BOOT_prod",
                        isLocal = FALSE)
```

imputeMissingCovariates

Impute missing covariate values

Description

Replaces user-defined sentinel values (default -99) in covariate columns with values that the NLME engine can consume, leaving the PML model code untouched. Within a reset block, masked rows are left as NA so the engine performs row-level propagation via `fcovariate()` / `covariate()` / `interpolate()`; only blocks (and subjects) where the engine has nothing to propagate receive an externally-computed fallback value.

Usage

```
imputeMissingCovariates(
  model,
  data = NULL,
  missingToken = c(-99),
  method = c("subject", "population"),
  continuousFn = stats::median,
  categoricalFn = imputeMode,
  centralValueWeighting = c("subject", "row"),
  stratifyBy = NULL,
  minStratumFraction = 0.1,
  keepOriginal = TRUE,
  logFile = NULL
)
```

Arguments

<code>model</code>	NlmePmlModel object. Must already have its covariate list populated (via <code>addCovariate()</code> for API-built models or <code>textualmodel()</code> for textual ones) and id mapped via <code>colMapping()</code> .
<code>data</code>	Optional data.frame to impute. Defaults to <code>model@inputData</code> . Never mutated by reference; an internal deep copy is used throughout.

missingToken	Vector of sentinel values. Default <code>c(-99)</code> . Supports numeric, NA, and character (e.g. <code>c(-99, NA, ".")</code>). Treated as a single missing-value predicate: a row is "missing" if its value matches any configured token. Tokens are not inferred: real NA values, <code>."</code> , and blank cells are only imputed when you list them here. Leaving them undeclared is an error rather than a silent no-op, because the NLME engine reads all three as missing.
method	"subject" (default) uses the cascade described in Details; "population" skips subject-level lookups and fills every masked row from the population or stratum central value.
continuousFn	Function applied to a non-empty numeric vector returning a single value. Default median. Must return a length-1 finite numeric.
categoricalFn	Function applied to a non-empty vector (character, factor, or numeric for occasion / numeric-coded categorical) returning a single value. Default <code>imputeMode()</code> – the package's exported modal-value helper with deterministic tie-breaking by first appearance and factor-level preservation. Pass a custom function (e.g. <code>function(x) sort(unique(x))[1]</code>) to override. Validation is type-driven , not model-driven: • For factor columns, the returned value must already be a level of the input factor; factor levels are never extended implicitly. This is a technical R constraint (factor codes cannot encode an unknown label), not a model-semantic rule. • For character / numeric columns, the returned value is written into the data verbatim, even when it is not among the categories declared in <code>model@covariateList[[name]]</code>. NLME treats out-of-declaration values per the model's default category semantics; the function does not second-guess that behaviour. If you need strict membership, supply a <code>categoricalFn</code> that enforces it.
centralValueWeighting	How the population / stratum central value is pooled. "subject" (default) is two-stage: the aggregator is applied to each subject's values, then to the resulting vector of subject-level summaries, so every subject carries equal weight. "row" applies the aggregator once across all non-missing rows, so subjects with more records count more. "subject" is the default because it matches how the NLME engine pools covariate values for <code>mean()</code> / <code>median()</code> centering (one summary per subject, then combined across subjects), and because row weighting otherwise tracks sampling density – a study-design artifact – rather than the population. Under "subject" a custom <code>continuousFn</code> / <code>categoricalFn</code> is invoked once per contributing subject and once across the subject summaries; a trimmed mean, for instance, then trims subject means rather than raw rows. The setting does not affect the "propagated" donor value, which is by definition computed within a single subject.
stratifyBy	Optional character vector of categorical column names. Population fallback is then computed within each stratum.
minStratumFraction	Minimum acceptable stratum size as a fraction of total subjects; a warning is issued if any stratum is smaller. Default 0.1.

keepOriginal	If TRUE (default), the raw pre-imputation values of each modified data column are preserved as <code><dataCol>_orig</code> . A pre-existing backup column triggers an error to prevent corrupting traceability with stale data.
logFile	Path to write the imputation log. Default NULL resolves to <code>file.path(model@modelInfo@workingDir, "imputeMissingCovariates.log")</code> , creating the working directory if it does not exist yet (it usually will not, since the model has not been run). If the directory cannot be created, the log falls back to <code>tempdir()</code> with a warning. An explicit logFile is never given this treatment: its parent directory must already exist, so a typo fails loudly instead of scattering directories. The returned path uses forward slashes on every platform so it can be copied straight out of the console.

Details

The function returns a modified clone of the input `NlmePmlModel` (with `@inputData` replaced by the imputed data `.frame`), along with the imputed data, a traceability summary, and a log file. All other model slots (`@columnMapping`, `@covariateList`, `@statements`, `@hasResetInfo`, `@resetInfo`, etc.) are preserved verbatim so custom mappings like `WT -> BodyWeight` survive the round-trip.

Imputation cascade (`method = "subject"`, default). The NLME engine propagates covariate values *within* a reset block but cannot carry them *across* blocks, so the cascade classifies each (subject, reset block):

1. **Block has at least one non-missing value** – masked rows are left as NA (source "engine"). The engine fills them via `fcovariate()` / `covariate()` / `interpolate()`.
2. **Block is fully missing but the subject has non-missing values in other blocks** – the masked rows are filled with the subject aggregate computed by `continuousFn` / `categoricalFn` over the subject's non-missing values across all blocks (source "propagated").
3. **Subject is fully missing across every block** – the masked rows are filled with the population (or stratum) central value (source "population" or "stratum").

`method = "population"` skips subject-level lookups and replaces every masked row with the population (or stratum) central value.

Note the two methods differ in their effect on covariate centering. `method = "subject"` leaves masked rows as NA, and the engine excludes missing rows when it computes `mean()` / `median()` centering values, so centering is unaffected. `method = "population"` writes a value into every masked row, and those values do take part in centering.

Value

A list with elements:

- `model` – clone of input `NlmePmlModel` with `@inputData` replaced by the imputed data `.frame`.
- `data` – imputed data `.frame` (same object as `result$model@inputData`).
- `summary` – data `.frame` with one row per (covariate, column, imputation_source, stratum) combination. Columns: `covariate`, `column`, `type`, `imputation_source`, `stratum`, `imputed_value`, `n_imputed`, `n_subjects_affected`. `imputed_value` is NA for "engine" (no value written; the engine fills at run time), a truncated list of distinct per-subject donors for "propagated", and the pooled central value for "stratum" / "population".
- `logFile` – absolute path of the written log file.

Examples

```
## Not run:
model <- pkmodel(...)
model <- addCovariate(model, "WT", type = "Continuous")
model <- colMapping(model, c(WT = "BodyWeight"))

result <- imputeMissingCovariates(model)
model <- result$model
fit <- fitmodel(model)
print(result$summary)

## End(Not run)
```

imputeMode	<i>Modal value with deterministic tie-break (default categorical aggregator)</i>
------------	--

Description

Returns the most frequent value in `x`. Used as the built-in default for `imputeMissingCovariates()`'s `categoricalFn` argument, and exposed publicly so callers can plug it into their own workflows or compose it with other rules.

Usage

```
imputeMode(x, missingToken = NULL)
```

Arguments

<code>x</code>	A non-empty atomic vector (character, factor, numeric, integer, or logical). Length-0 input errors.
<code>missingToken</code>	Optional atomic vector of sentinel values to exclude from the candidate set (compared after <code>as.character()</code> on both sides). Default <code>NULL</code> means no values are excluded.

Details

Properties:

- **Deterministic tie-break.** When two or more values tie for the highest frequency, the one that appears first in `x` wins. This makes the result stable across runs and independent of locale-sensitive ordering inside `table()`.
- **Factor-preserving.** If `x` is a factor, the return value is a factor with the same `levels(x)`. Plain character / numeric vectors return a length-1 value of the same atomic type.
- **NA-tolerant.** NA entries are ignored when computing the mode. If every entry is NA the function errors.

- **Optional token avoidance.** When missingToken is supplied, values matching any element of it are excluded from the candidate set. [imputeMissingCovariates\(\)](#) masks the input pool up-front so this is normally a no-op there; the parameter is useful when calling imputeMode() standalone on a raw vector that may still contain sentinels.

Value

A length-1 value of the same atomic type as x (factor for factor input).

Examples

```
imputeMode(c("M", "F", "M", "F"))           # "M" (ties: first wins)
imputeMode(factor(c("M", "F", "M"), levels = c("F","M"))) # factor "M"
imputeMode(c(70, 70, 65, -99), missingToken = -99)      # 70
```

initFixedEffects	<i>Display/Set initial estimates for fixed effects</i>
------------------	--

Description

Display/Set initial estimates for fixed effects

Usage

```
initFixedEffects(.Object)

## S4 method for signature 'NlmePmlModel'
initFixedEffects(.Object)

initFixedEffects(.Object) <- value

## S4 replacement method for signature 'NlmePmlModel'
initFixedEffects(.Object) <- value
```

Arguments

.Object	PK/PD model
value	Named numeric vector

Value

Named numeric vector of fixed effects estimates

See Also

[fixedEffect](#)

Examples

```

model <- pkmodel(
  numCompartments = 2,
  data = pkData,
  ID = "Subject",
  Time = "Act_Time",
  A1 = "Amount",
  CObs = "Conc",
  modelName = "TwCpt_IVBolus_FOCE_ELS",
  workingDir = tempdir()
)

# View initial/current fixed effect values
initFixedEffects(model)

# May also use as a 'replacement function' to set the values
initFixedEffects(model) <- c(tvV = 15, tvC1 = 5, tvV2 = 40, tvC12 = 15)

```

isPopulation

*Get or set whether a model is a population model***Description**

isPopulation() reports the population / individual mode of an `NlmePmlModel`. The replacement form `isPopulation(model) <- value` flips that mode and keeps the column mapping consistent: switching to individual removes the id mapping entry (the individual write path always injects a dummy id), and switching to population re-adds an unmapped id entry that is then auto-mapped against any matching data column name.

Usage

```

isPopulation(.Object)

## S4 method for signature 'NlmePmlModel'
isPopulation(.Object)

isPopulation(.Object) <- value

## S4 replacement method for signature 'NlmePmlModel'
isPopulation(.Object) <- value

```

Arguments

<code>.Object</code>	An <code>NlmePmlModel</code> object.
<code>value</code>	A single non-NA logical. TRUE for population mode, FALSE for individual mode.

Details

Random effects and author-written PML are left untouched; the engine decides how to execute a model whose PML disagrees with the mode.

Value

For the getter, a length-1 logical. For the setter, the updated NlmePmlModel.

See Also

[colMapping](#), [textualmodel](#), [pkmodel](#)

Examples

```
## Not run:
model <- pkmodel(columnMap = FALSE, data = pkData)
isPopulation(model)
isPopulation(model) <- FALSE

## End(Not run)
```

linearmodel

Create linear model

Description

Use to create a constant, linear, or quadratic PD model

Usage

```
linearmodel(
  isPopulation = TRUE,
  type = "Constant",
  data = NULL,
  columnMap = TRUE,
  modelName = "",
  workingDir = "",
  ...
)
```

Arguments

isPopulation	Is this a population model TRUE or individual model FALSE?
type	Model type. Options are "Constant", "Linear", "Quadratic".
data	Input dataset

columnMap	If TRUE (default) column mapping arguments are required. Set to FALSE to manually map columns after defining model using colMapping .
modelName	Model name for subdirectory created for model output in current working directory.
workingDir	Working directory to run the model. Current working directory will be used if workingDir not specified.
...	Arguments passed on to linearmodel_MappingParameters
ID	Column mapping argument for input dataset column(s) that identify individual data profiles. Only applicable to population models isPopulation = TRUE.
C	Column mapping argument that represents the input dataset column for the independent variable that is treated as a covariate during the estimation/simulation process.
E0bs	Column mapping argument that represents the input dataset column for the observed drug effect (i.e., the dependent variable).

Value

NlmePmlModel object

Column mapping

Note that quoted and unquoted column names are supported. Please see [colMapping](#).

Examples

```
model <- linearmodel(type = "Linear", data = pkpdData, ID = "ID", C = "C0bs", E0bs = "E0bs")

# View PML Code
print(model)
```

```
listCovariateEffectNames
```

Lists covariate effect names in the model

Description

This function lists the names of covariate effects in a provided pharmacokinetic/pharmacodynamic (PK/PD) model.

Usage

```
listCovariateEffectNames(.Object)

## S4 method for signature 'NlmePmlModel'
listCovariateEffectNames(.Object)
```

Arguments

.Object PK/PD model

Value

A vector of character strings containing the names of the covariate effects in the model.

Examples

```
model <- pkmodel(
  numCompartments = 2,
  data = pkData,
  ID = "Subject",
  Time = "Act_Time",
  A1 = "Amount",
  CObs = "Conc",
  workingDir = tempdir()
)
model <- addCovariate(model,
  covariate = "Gender",
  type = "Categorical",
  effect = c("V2", "C12"),
  levels = c(0, 1),
  labels = c("Female", "Male")
)
listCovariateEffectNames(model)
```

listSCMRuns

List archived covariate search scenarios.

Description

Dispatches on the type of `x` to discover archived covariate-search run folders and returns one row per scenario, sorted by timestamp descending (latest first). This function discovers only `stepwise_*` and `shotgun_*` folders, not transient `SCM_*` directories.

Usage

```
listSCMRuns(x)
```

Arguments

`x` One of:

- A search-result data frame (from `shotgunSearch()` / `stepwiseSearch()`) with a `searchRunDir` attribute. Lists scenarios for that single archived run.
- An `NlmePmlModel`. Scans the model's working directory for all archived covariate-search run folders.

- A character path to an archived run folder (contains `scenario_index.csv`). Lists scenarios for that single run.
- A character path to a working directory. Scans for all child `stepwise_*` and `shotgun_*` run folders.

Value

A data frame with one row per scenario. Columns: `searchType`, `SCMFolder` (path to the archived run folder; name kept for compatibility), `runFolder`, `Scenario`, `enableFlags`, `status`, `timestamp`, `runId`, `overallFile`, `stepwiseFile`. Returns an empty data frame if no runs are found.

Archived run folders

Archived folders are created when `shotgunSearch()` or `stepwiseSearch()` is run with `archiveResults = TRUE`. During the engine run, NLME8 writes scenario outputs under a temporary `SCM_<runId>/` directory in the model working directory (`runId` is a wall-clock timestamp, optionally suffixed with `runLabel`). After the engine finishes, `collectJob()` materialises the search result and the package renames that tree into a durable folder named `stepwise_<runId>` or `shotgun_<runId>` (on remote hosts the bundle `SCM_<runId>.tar.gz` is extracted first, then the bundle and any loose `SCM_<runId>/` copy are removed). The final folder holds `scenario_index.csv`, per-scenario sub-folders, `Overall.csv`, optional stepwise logs, and `rsnlme_run.rds`.

See Also

[shotgunSearch](#), [stepwiseSearch](#), [summary\(\)](#), [print\(\)](#), [getSCMResults](#)

modelVariableNames	<i>Return model variable names</i>
--------------------	------------------------------------

Description

Return a vector of model variable names from model object

Usage

```
modelVariableNames(model)
```

Arguments

model	Model object
-------	--------------

Value

Character vector of required model variable names

Examples

```
model <- pkmodel(columnMap = FALSE,
                 workingDir = tempdir())
modelVariableNames(model)
```

obtain_NLMELicense	<i>Obtain NLME License</i>
--------------------	----------------------------

Description

This function attempts to authenticate and obtain an NLME license using the specified installation directory and licensing tool.

Usage

```
obtain_NLMELicense(
  InstallDir = Sys.getenv("INSTALLDIR"),
  ForceAuth = FALSE,
  ForceLicenseGet = FALSE,
  verbose = getOption("verbose")
)
```

Arguments

InstallDir	A character string specifying the directory where the NLME Engine is installed e.g., INSTALLDIR environment variable. The cadlicensingtool executable is expected to be located within this directory, or within a subdirectory specified by the PML_BIN_DIR environment variable.
ForceAuth	A logical value indicating whether to force re-authentication even if already authenticated. Default is FALSE.
ForceLicenseGet	A logical value indicating whether to force obtaining the license even if already licensed. Default is FALSE.
verbose	A logical value indicating whether to print verbose output. Default is <code>getOption("verbose")</code> .

Details

This function checks for the presence of the necessary appsettings.json file as indicated by the CAD_CONFIG_FILE environment variable, runs the licensing tool to authenticate the user, and attempts to obtain an NLME license. It prints detailed messages if the verbose parameter is set to TRUE.

Value

A logical value indicating whether the license was successfully obtained.

Examples

```
## Not run:
INSTALLDIR <- Sys.getenv("INSTALLDIR")
if (INSTALLDIR == "") INSTALLDIR <- "C:/Program Files/Certara/NLME_Engine"
result <- obtain_NLMElicense(INSTALLDIR, verbose = TRUE)
if (result) {
  message("License obtained successfully!")
} else {
  message("Failed to obtain license.")
}

## End(Not run)
```

OneCpt_IVInfusionData *Pharmacokinetic dataset containing 100 subjects with single dose given by infusion*

Description

Pharmacokinetic dataset containing 16 subjects with single dose given by infusion.

Usage

```
OneCpt_IVInfusionData
```

Format

A data frame with 800 rows and 6 variables:

Subject Subject ID

Time Time point

Dose Amount of dose

CObs Observations of drug concentration in blood

Rate Rate of infusion

Duration Duration of infusion

Source

The data is simulated using a PK model described by a one-compartment model with IV infusion

parsePMLColMap	<i>Embed column definition info into the model</i>
----------------	--

Description

Add/update column definition information for the model object

Usage

```
parsePMLColMap(.Object, ForceRun = TRUE)
```

Arguments

.Object	Model (NlmePmlModel) object
ForceRun	Set to TRUE to force run

Details

Intended to be used by other packages

Value

modified NLMEPmlModel object with column mapping definitions

pkcovbqlData	<i>Pharmacokinetic pediatric dataset containing 80 subjects with single bolus dose.</i>
--------------	---

Description

Pharmacokinetic pediatric dataset containing 80 subjects with single bolus dose. Dataset includes covariates and observations Below Quantification Limit (BQL).

Usage

```
pkcovbqlData
```

Format

A data frame with 880 rows and 8 variables:

ID Subject ID

Time Nominal Time

Dose Amount of dose

CObs Observations of drug concentration in blood

LLOQ Lower Limit of Quantification

CObsBQL Variable that indicates whether the observed drug concentration is below the limit of quantification

BW Body weight

PMA Postmenstrual age

Source

The data is simulated using a one-compartment model with IV bolus, where the central volume is allometric weight scaled, and the clearance is scaled by a combination of allometric weight scaling and a sigmoidal maturation function driven by PMA. Germovsek E., et al, Pharmacokinetic–Pharmacodynamic Modeling in Pediatric Drug Development, and the Importance of Standardized Scaling of Clearance, Clin Pharmacokinet (2019) 58:39–52.

pkData

Pharmacokinetic dataset containing 16 subjects with single bolus dose

Description

Pharmacokinetic dataset containing 16 subjects with single bolus dose.

Usage

pkData

Format

A data frame with 112 rows and 8 variables:

Subject Subject ID

Nom_Time Nominal Time

Act_Time Actual Time

Amount Amount of dose

Conc Observations of drug concentration in blood

Age Age

BodyWeight Body weight

Gender Gender ("male", "female")

Source

Certara University

pkemaxmodel

*Create a PK/Emax or PK/Imax model***Description**

Use to create a PK/Emax or PK/Imax model

Usage

```
pkemaxmodel(
  isPopulation = TRUE,
  parameterization = "Clearance",
  absorption = "Intravenous",
  numCompartments = 1,
  isClosedForm = TRUE,
  isTlag = FALSE,
  hasEliminationComp = FALSE,
  isFractionExcreted = FALSE,
  isSaturating = FALSE,
  infusionAllowed = FALSE,
  isDuration = FALSE,
  isSequential = FALSE,
  isPkFrozen = FALSE,
  hasEffectsCompartment = FALSE,
  checkBaseline = FALSE,
  checkFractional = FALSE,
  checkInhibitory = FALSE,
  checkSigmoid = FALSE,
  isEmaxFrozen = FALSE,
  data = NULL,
  columnMap = TRUE,
  modelName = "",
  workingDir = "",
  ...
)
```

Arguments

isPopulation	Is this a population model TRUE or individual model FALSE?
parameterization	Type of parameterization. Options are "Clearance", "Micro", "Macro", or "Macro1".
absorption	Type of absorption. Options are "Intravenous", "FirstOrder", "Gamma", "InverseGaussian", "Weibull".
numCompartments	Value of either 1, 2, or 3.

isClosedForm	Set to TRUE to convert model from a differential equation to close form.
isTlag	Set to TRUE to add a lag time parameter to the model.
hasEliminationComp	Set to TRUE to add an elimination compartment to the model.
isFractionExcreted	Set to TRUE if elimination compartment (hasEliminationComp = TRUE) contains a fraction excreted parameter.
isSaturating	Set to TRUE to use Michaelis-Menten kinetics for elimination. Only applicable to models with paramteterization = "Clearance"
infusionAllowed	Set to TRUE if infusions allowed.
isDuration	Set to TRUE if infusions use duration instead of rate (must also set infusionAllowed = TRUE).
isSequential	Set to TRUE to freeze PK fixed effects and convert the corresponding random effects into covariates as well as remove the PK observed variable from the model.
isPkFrozen	Set to TRUE to freeze PK fixed effects and remove the corresponding random effects as well as the PK observed variable from the model.
hasEffectsCompartment	Set to TRUE to include an effect compartment into the model.
checkBaseline	Does Emax/Imax model have a baseline response?
checkFractional	Set to TRUE to modify the default form for the Emax/Imax model. Only applicable to models with checkBaseline = TRUE.
checkInhibitory	Set to TRUE to change the default Emax to Imax model.
checkSigmoid	Set to TRUE to change the Emax/Imax to its corresponding sigmoid form.
isEmaxFrozen	Set to TRUE to freeze PD fixed effects and remove the corresponding random effects as well as the PD observed variable from the model.
data	Input dataset
columnMap	If TRUE (default) column mapping arguments are required. Set to FALSE to manually map columns after defining model using colMapping .
modelName	Model name for subdirectory created for model output in current working directory.
workingDir	Working directory to run the model. Current working directory will be used if workingDir not specified.
...	Arguments passed on to pkindirectmodel_MappingParameters
ID	Column mapping argument for input dataset column(s) that identify individual data profiles. Only applicable to population models isPopulation = TRUE.
Time	Column mapping argument that represents the input dataset column for the relative time used in a study and only applicable to time-based models.

- A1 Column mapping argument that represents the input dataset column for the amount of drug administered. Only applicable to the following types of models:
- Models with absorption = "Intravenous" and parameterization set to either "Clearance", "Micro", or "Macro"
 - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull"
- Aa Column mapping argument that represents the input dataset column for the amount of drug administered and only applicable to models with absorption = "FirstOrder".
- A Column mapping argument that represents the input dataset column for the amount of drug administered and only applicable to models with absorption = "Intravenous" and parameterization = "Macro1".
- A1_Rate Column mapping argument that represents the input dataset column for the rate of drug administered. Only applicable to the following types of models:
- Models with absorption = "Intravenous", infusionAllowed = TRUE and parameterization set to either "Clearance", "Micro" or "Macro"
 - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull" and infusionAllowed = TRUE
- A1_Duration Column mapping argument that represents the input dataset column for the duration of drug administered. Only applicable to the following types of models:
- Models with absorption = "Intravenous", infusionAllowed = TRUE with isDuration = TRUE and parameterization set to either "Clearance", "Micro" or "Macro"
 - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull" and infusionAllowed = TRUE with isDuration = TRUE
- Aa_Rate Column mapping argument that represents the input dataset column for the rate of drug administered and only applicable to models with absorption = "FirstOrder", infusionAllowed = TRUE.
- Aa_Duration Column mapping argument that represents the input dataset column for the duration of drug administered and only applicable to models with absorption = "FirstOrder", infusionAllowed = TRUE, and isDuration = TRUE.
- A_Rate Column mapping argument that represents the input dataset column for the rate of drug administered and only applicable to models with absorption = "Intravenous", infusionAllowed = TRUE, and parameterization = "Macro1".
- A_Duration Column mapping argument that represents the input dataset column for the duration of drug administered and only applicable to models with absorption = "Intravenous", infusionAllowed = TRUE, isDuration = TRUE, and parameterization = "Macro1".
- A1Strip Column mapping argument that represents the input dataset column for the stripping dose and only applicable to models with parameterization = "Macro".
- CObs Column mapping argument that represents the input dataset column for the observations of drug concentration in the central compartment and only

applicable to models with parameterization being either set to either "Clearance" or "Micro".

C10bs Column mapping argument that represents the input dataset column for the observations of drug concentration in the central compartment and only applicable to models with parameterization being either set to either "Macro" or "Macro1".

A00bs Column mapping argument that represents the input dataset column for the observed amount of drug in the elimination compartment. (hasEliminationComp = TRUE).

E0bs Column mapping argument that represents the input dataset column for the observed drug effect.

nV If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nV.

nV2 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nV2.

nV3 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nV3.

nC1 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nC1.

nC12 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nC12.

nC13 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nC13.

nKa If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nKa.

nA If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nA.

nAlpha If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nAlpha.

nB If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nB.

nBeta If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nBeta.

nC If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nC.

nGamma If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nGamma.

nKe If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nKe.

nK12 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nK12.

nK21 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nK21.

nK13 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nK13.

nK31 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nK31.

nTlag If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nTlag.

nKm If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nKm.

nVmax If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nVmax.

nFe If `isSequential = TRUE` and `isFractionExcreted = TRUE`, mapped to the input dataset column that lists the values for random effect nFe.

nMeanDelayTime If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nMeanDelayTime.

nShapeParam If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nShapeParam.

nShapeParamMinusOne If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nShapeParamMinusOne.

Value

NlmePmlModel object

Column mapping

Note that quoted and unquoted column names are supported. Please see [colMapping](#).

Examples

```
model <- pkemaxmodel(
  parameterization = "Macro",
  data = pkpdData,
  Time = "Time",
  ID = "ID",
  A1 = "Dose",
  C1Obs = "CObs",
  EObs = "EObs"
)

# View the model as well as its associated column mappings
print(model)
```

pkindirectmodel

Create a PK/Indirect response model

Description

Use to create a PK/Indirect response model.

Usage

```

pkindirectmodel(
  isPopulation = TRUE,
  parameterization = "Clearance",
  absorption = "Intravenous",
  numCompartments = 1,
  isClosedForm = TRUE,
  isTlag = FALSE,
  hasEliminationComp = FALSE,
  isFractionExcreted = FALSE,
  isSaturating = FALSE,
  infusionAllowed = FALSE,
  isDuration = FALSE,
  isSequential = FALSE,
  isPkFrozen = FALSE,
  hasEffectsCompartment = FALSE,
  indirectType = "LimitedStimulation",
  isBuildup = TRUE,
  isExponent = FALSE,
  indirectFrozen = FALSE,
  data = NULL,
  columnMap = TRUE,
  modelName = "",
  workingDir = "",
  ...
)

```

Arguments

<code>isPopulation</code>	Is this a population model TRUE or individual model FALSE?
<code>parameterization</code>	Type of parameterization. Options are "Clearance", "Micro", "Macro", or "Macro1".
<code>absorption</code>	Type of absorption. Options are "Intravenous", "FirstOrder", "Gamma", "InverseGaussian", "Weibull".
<code>numCompartments</code>	Value of either 1, 2, or 3.
<code>isClosedForm</code>	Set to TRUE to convert model from a differential equation to close form.
<code>isTlag</code>	Set to TRUE to add a lag time parameter to the model.
<code>hasEliminationComp</code>	Set to TRUE to add an elimination compartment to the model.
<code>isFractionExcreted</code>	Set to TRUE if elimination compartment (<code>hasEliminationComp = TRUE</code>) contains a fraction excreted parameter.
<code>isSaturating</code>	Set to TRUE to use Michaelis-Menten kinetics for elimination. Only applicable to models with <code>parameterization = "Clearance"</code>

infusionAllowed	Set to TRUE if infusions allowed.
isDuration	Set to TRUE if infusions use duration instead of rate (must also set infusionAllowed = TRUE).
isSequential	Set to TRUE to freeze PK fixed effects and convert the corresponding random effects into covariates as well as remove the PK observed variable from the model.
isPkFrozen	Set to TRUE to freeze PK fixed effects and remove the corresponding random effects as well as the PK observed variable from the model.
hasEffectsCompartment	Set to TRUE to include an effect compartment into the model.
indirectType	Type of drug actions for the indirect response model. Options are "LimitedStimulation", "InfiniteStimulation", "LimitedInhibition", "InverseInhibition", "LinearStimulation", or "LogLinearStimulation".
isBuildup	Set to FALSE to have the drug actions affect the loss/degradation instead of the production.
isExponent	Set to TRUE to add an exponent parameter to the drug action term.
indirectFrozen	Set to TRUE to freeze PD fixed effects and remove the corresponding random effects as well as the PD observed variable from the model.
data	Input dataset
columnMap	If TRUE (default) column mapping arguments are required. Set to FALSE to manually map columns after defining model using colMapping .
modelName	Model name for subdirectory created for model output in current working directory.
workingDir	Working directory to run the model. Current working directory will be used if workingDir not specified.
...	Arguments passed on to pkindirectmodel_MappingParameters
ID	Column mapping argument for input dataset column(s) that identify individual data profiles. Only applicable to population models isPopulation = TRUE.
Time	Column mapping argument that represents the input dataset column for the relative time used in a study and only applicable to time-based models.
A1	Column mapping argument that represents the input dataset column for the amount of drug administered. Only applicable to the following types of models: - Models with absorption = "Intravenous" and parameterization set to either "Clearance", "Micro", or "Macro" - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull"
Aa	Column mapping argument that represents the input dataset column for the amount of drug administered and only applicable to models with absorption = "FirstOrder".
A	Column mapping argument that represents the input dataset column for the amount of drug administered and only applicable to models with absorption = "Intravenous" and parameterization = "Macro1".

- A1_Rate Column mapping argument that represents the input dataset column for the rate of drug administered. Only applicable to the following types of models:
- Models with absorption = "Intravenous", infusionAllowed = TRUE and parameterization set to either "Clearance", "Micro" or "Macro"
 - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull" and infusionAllowed = TRUE
- A1_Duration Column mapping argument that represents the input dataset column for the duration of drug administered. Only applicable to the following types of models:
- Models with absorption = "Intravenous", infusionAllowed = TRUE with isDuration = TRUE and parameterization set to either "Clearance", "Micro" or "Macro"
 - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull" and infusionAllowed = TRUE with isDuration = TRUE
- Aa_Rate Column mapping argument that represents the input dataset column for the rate of drug administered and only applicable to models with absorption = "FirstOrder", infusionAllowed = TRUE.
- Aa_Duration Column mapping argument that represents the input dataset column for the duration of drug administered and only applicable to models with absorption = "FirstOrder", infusionAllowed = TRUE, and isDuration = TRUE.
- A_Rate Column mapping argument that represents the input dataset column for the rate of drug administered and only applicable to models with absorption = "Intravenous", infusionAllowed = TRUE, and parameterization = "Macro1".
- A_Duration Column mapping argument that represents the input dataset column for the duration of drug administered and only applicable to models with absorption = "Intravenous", infusionAllowed = TRUE, isDuration = TRUE, and parameterization = "Macro1".
- A1Strip Column mapping argument that represents the input dataset column for the stripping dose and only applicable to models with parameterization = "Macro".
- C0bs Column mapping argument that represents the input dataset column for the observations of drug concentration in the central compartment and only applicable to models with parameterization being either set to either "Clearance" or "Micro".
- C10bs Column mapping argument that represents the input dataset column for the observations of drug concentration in the central compartment and only applicable to models with parameterization being either set to either "Macro" or "Macro1".
- A00bs Column mapping argument that represents the input dataset column for the observed amount of drug in the elimination compartment. (hasEliminationComp = TRUE).
- E0bs Column mapping argument that represents the input dataset column for the observed drug effect.
- nV If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nV.

nV2 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nV2.

nV3 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nV3.

nC1 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nC1.

nC12 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nC12.

nC13 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nC13.

nKa If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nKa.

nA If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nA.

nAlpha If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nAlpha.

nB If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nB.

nBeta If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nBeta.

nC If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nC.

nGamma If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nGamma.

nKe If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nKe.

nK12 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nK12.

nK21 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nK21.

nK13 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nK13.

nK31 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nK31.

nTlag If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nTlag.

nKm If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nKm.

nVmax If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nVmax.

nFe If `isSequential = TRUE` and `isFractionExcreted = TRUE`, mapped to the input dataset column that lists the values for random effect nFe.

nMeanDelayTime If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nMeanDelayTime.

nShapeParam If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nShapeParam.

nShapeParamMinusOne If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nShapeParamMinusOne.

Value

NlmePmlModel object

Column mapping

Note that quoted and unquoted column names are supported. Please see [colMapping](#).

Examples

```
model <- pkindirectmodel(
  parameterization = "Micro",
  data = pkpdData,
  ID = "ID",
  Time = "Time",
  A1 = "Dose",
  CObs = "CObs",
  EObs = "EObs"
)

# View PML Code
print(model)
```

pklinearmodel	Create PK linear model
---------------	------------------------

Description

Use to create a PK/PD model with PD described by either constant, linear, or quadratic model

Usage

```
pklinearmodel(
  isPopulation = TRUE,
  parameterization = "Clearance",
  absorption = "Intravenous",
  numCompartments = 1,
  isClosedForm = TRUE,
  isTlag = FALSE,
  hasEliminationComp = FALSE,
  isFractionExcreted = FALSE,
  isSaturating = FALSE,
  infusionAllowed = FALSE,
  isDuration = FALSE,
  isSequential = FALSE,
```

```

    isPkFrozen = FALSE,
    hasEffectsCompartment = FALSE,
    linearType = "Constant",
    isLinearFrozen = FALSE,
    data = NULL,
    columnMap = TRUE,
    modelName = "",
    workingDir = "",
    ...
)

```

Arguments

isPopulation	Is this a population model TRUE or individual model FALSE?
parameterization	Type of parameterization. Options are "Clearance", "Micro", "Macro", or "Macro1".
absorption	Type of absorption. Options are "Intravenous", "FirstOrder", "Gamma", "InverseGaussian", "Weibull".
numCompartments	Value of either 1, 2, or 3.
isClosedForm	Set to TRUE to convert model from a differential equation to close form.
isTlag	Set to TRUE to add a lag time parameter to the model.
hasEliminationComp	Set to TRUE to add an elimination compartment to the model.
isFractionExcreted	Set to TRUE if elimination compartment (hasEliminationComp = TRUE) contains a fraction excreted parameter.
isSaturating	Set to TRUE to use Michaelis-Menten kinetics for elimination. Only applicable to models with parameterization = "Clearance"
infusionAllowed	Set to TRUE if infusions allowed.
isDuration	Set to TRUE if infusions use duration instead of rate (must also set infusionAllowed = TRUE).
isSequential	Set to TRUE to freeze PK fixed effects and convert the corresponding random effects into covariates as well as remove the PK observed variable from the model.
isPkFrozen	Set to TRUE to freeze PK fixed effects and remove the corresponding random effects as well as the PK observed variable from the model.
hasEffectsCompartment	Set to TRUE to include an effect compartment into the model.
linearType	Type of PD model; Options are "Constant", "Linear", "Quadratic".
isLinearFrozen	Set to TRUE to freeze PD fixed effects and remove the corresponding random effects as well as the PD observed variable from the model.
data	Input dataset

columnMap	If TRUE (default) column mapping arguments are required. Set to FALSE to manually map columns after defining model using colMapping .
modelName	Model name for subdirectory created for model output in current working directory.
workingDir	Working directory to run the model. Current working directory will be used if workingDir not specified.
...	Arguments passed on to pkindirectmodel_MappingParameters
ID	Column mapping argument for input dataset column(s) that identify individual data profiles. Only applicable to population models isPopulation = TRUE.
Time	Column mapping argument that represents the input dataset column for the relative time used in a study and only applicable to time-based models.
A1	Column mapping argument that represents the input dataset column for the amount of drug administered. Only applicable to the following types of models: - Models with absorption = "Intravenous" and parameterization set to either "Clearance", "Micro", or "Macro" - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull"
Aa	Column mapping argument that represents the input dataset column for the amount of drug administered and only applicable to models with absorption = "FirstOrder".
A	Column mapping argument that represents the input dataset column for the amount of drug administered and only applicable to models with absorption = "Intravenous" and parameterization = "Macro1".
A1_Rate	Column mapping argument that represents the input dataset column for the rate of drug administered. Only applicable to the following types of models: - Models with absorption = "Intravenous", infusionAllowed = TRUE and parameterization set to either "Clearance", "Micro" or "Macro" - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull" and infusionAllowed = TRUE
A1_Duration	Column mapping argument that represents the input dataset column for the duration of drug administered. Only applicable to the following types of models: - Models with absorption = "Intravenous", infusionAllowed = TRUE with isDuration = TRUE and parameterization set to either "Clearance", "Micro" or "Macro" - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull" and infusionAllowed = TRUE with isDuration = TRUE
Aa_Rate	Column mapping argument that represents the input dataset column for the rate of drug administered and only applicable to models with absorption = "FirstOrder", infusionAllowed = TRUE.
Aa_Duration	Column mapping argument that represents the input dataset column for the duration of drug administered and only applicable to models

with absorption = "FirstOrder", infusionAllowed = TRUE, and isDuration = TRUE.

A_Rate Column mapping argument that represents the input dataset column for the rate of drug administered and only applicable to models with absorption = "Intravenous", infusionAllowed = TRUE, and parameterization = "Macro1".

A_Duration Column mapping argument that represents the input dataset column for the duration of drug administered and only applicable to models with absorption = "Intravenous", infusionAllowed = TRUE, isDuration = TRUE, and parameterization = "Macro1".

A1Strip Column mapping argument that represents the input dataset column for the stripping dose and only applicable to models with parameterization = "Macro".

C0bs Column mapping argument that represents the input dataset column for the observations of drug concentration in the central compartment and only applicable to models with parameterization being either set to either "Clearance" or "Micro".

C10bs Column mapping argument that represents the input dataset column for the observations of drug concentration in the central compartment and only applicable to models with parameterization being either set to either "Macro" or "Macro1".

A00bs Column mapping argument that represents the input dataset column for the observed amount of drug in the elimination compartment. (hasEliminationComp = TRUE).

E0bs Column mapping argument that represents the input dataset column for the observed drug effect.

nV If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nV.

nV2 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nV2.

nV3 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nV3.

nC1 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nC1.

nC12 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nC12.

nC13 If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nC13.

nKa If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nKa.

nA If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nA.

nAlpha If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nAlpha.

nB If isSequential = TRUE, mapped to the input dataset column that lists the values for random effect nB.

nBeta If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nBeta.

nC If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nC.

nGamma If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nGamma.

nKe If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nKe.

nK12 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nK12.

nK21 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nK21.

nK13 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nK13.

nK31 If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nK31.

nTlag If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nTlag.

nKm If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nKm.

nVmax If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nVmax.

nFe If `isSequential = TRUE` and `isFractionExcreted = TRUE`, mapped to the input dataset column that lists the values for random effect nFe.

nMeanDelayTime If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nMeanDelayTime.

nShapeParam If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nShapeParam.

nShapeParamMinusOne If `isSequential = TRUE`, mapped to the input dataset column that lists the values for random effect nShapeParamMinusOne.

Value

NlmePmlModel object

Column mapping

Note that quoted and unquoted column names are supported. Please see [colMapping](#).

Examples

```
model <- pklinearmodel(
  parameterization = "Clearance",
  linearType = "Constant",
  data = pkpdData,
  ID = "ID",
  Time = "Time",
  A1 = "Dose",
```

```

    CObs = "CObs",
    EObs = "EObs"
  )

  # View the model as well as its associated column mappings
  print(model)

```

pkmodel

*Creates a PK model***Description**

Use to create a PK model

Usage

```

pkmodel(
  isPopulation = TRUE,
  parameterization = "Clearance",
  absorption = "Intravenous",
  numCompartments = 1,
  isClosedForm = TRUE,
  isTlag = FALSE,
  hasEliminationComp = FALSE,
  isFractionExcreted = FALSE,
  isSaturating = FALSE,
  infusionAllowed = FALSE,
  isDuration = FALSE,
  isStdevFrozen = FALSE,
  data = NULL,
  columnMap = TRUE,
  modelName = "",
  workingDir = "",
  ...
)

```

Arguments

isPopulation	Is this a population model TRUE or individual model FALSE?
parameterization	Type of parameterization. Options are "Clearance", "Micro", "Macro", or "Macro1".
absorption	Type of absorption. Options are "Intravenous", "FirstOrder", "Gamma", "InverseGaussian", "Weibull".
numCompartments	Value of either 1, 2, or 3.

isClosedForm	Set to TRUE to convert model from a differential equation to close form.
isTlag	Set to TRUE to add a lag time parameter to the model.
hasEliminationComp	Set to TRUE to add an elimination compartment to the model.
isFractionExcreted	Set to TRUE if elimination compartment (hasEliminationComp = TRUE) contains a fraction excreted parameter.
isSaturating	Set to TRUE to use Michaelis-Menten kinetics for elimination. Only applicable to models with paramteterization = "Clearance"
infusionAllowed	Set to TRUE if infusions allowed.
isDuration	Set to TRUE if infusions use duration instead of rate (must also set infusionAllowed = TRUE).
isStdevFrozen	Set to TRUE to freeze value of standard deviation of residual error variable.
data	Input dataset
columnMap	If TRUE (default) column mapping arguments are required. Set to FALSE to manually map columns after defining model using colMapping .
modelName	Model name for subdirectory created for model output in current working directory.
workingDir	Working directory to run the model. Current working directory will be used if workingDir not specified.
...	Arguments passed on to pkmodel_MappingParameters
ID	Column mapping argument for input dataset column(s) that identify individual data profiles. Only applicable to population models isPopulation = TRUE.
Time	Column mapping argument that represents the input dataset column for the relative time used in a study and only applicable to time-based models.
A1	Column mapping argument that represents the input dataset column for the amount of drug administered. Only applicable to the following types of models: - Models with absorption = "Intravenous" and parameterization set to either "Clearance", "Micro", or "Macro" - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull"
Aa	Column mapping argument that represents the input dataset column for the amount of drug administered and only applicable to models with absorption = "FirstOrder".
A	Column mapping argument that represents the input dataset column for the amount of drug administered and only applicable to models with absorption = "Intravenous" and parameterization = "Macro1".
A1_Rate	Column mapping argument that represents the input dataset column for the rate of drug administered. Only applicable to the following types of models:

- Models with absorption = "Intravenous", infusionAllowed = TRUE and parameterization set to either "Clearance", "Micro" or "Macro"
 - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull" and infusionAllowed = TRUE
- A1_Duration Column mapping argument that represents the input dataset column for the duration of drug administered. Only applicable to the following types of models:
- Models with absorption = "Intravenous", infusionAllowed = TRUE with isDuration = TRUE and parameterization set to either "Clearance", "Micro" or "Macro"
 - Models with absorption set to either "Gamma", "InverseGaussian", or "Weibull" and infusionAllowed = TRUE with isDuration = TRUE
- Aa_Rate Column mapping argument that represents the input dataset column for the rate of drug administered and only applicable to models with absorption = "FirstOrder", infusionAllowed = TRUE.
- Aa_Duration Column mapping argument that represents the input dataset column for the duration of drug administered and only applicable to models with absorption = "FirstOrder", infusionAllowed = TRUE, and isDuration = TRUE.
- A_Rate Column mapping argument that represents the input dataset column for the rate of drug administered and only applicable to models with absorption = "Intravenous", infusionAllowed = TRUE, and parameterization = "Macro1".
- A_Duration Column mapping argument that represents the input dataset column for the duration of drug administered and only applicable to models with absorption = "Intravenous", infusionAllowed = TRUE, isDuration = TRUE, and parameterization = "Macro1".
- A1Strip Column mapping argument that represents the input dataset column for the stripping dose and only applicable to models with parameterization = "Macro".
- C0bs Column mapping argument that represents the input dataset column for the observations of drug concentration in the central compartment and only applicable to models with parameterization being either set to either "Clearance" or "Micro".
- C10bs Column mapping argument that represents the input dataset column for the observations of drug concentration in the central compartment and only applicable to models with parameterization being either set to either "Macro" or "Macro1".
- A00bs Column mapping argument that represents the input dataset column for the observed amount of drug in the elimination compartment. (hasEliminationComp = TRUE).

Value

NlmePmlModel object

Column mapping

Note that quoted and unquoted column names are supported. Please see [colMapping](#).

Examples

```
model <- pkmodel(
  parameterization = "Clearance",
  numCompartments = 2,
  data = pkData,
  ID = "Subject",
  Time = "Act_Time",
  A1 = "Amount",
  CObs = "Conc",
  workingDir = tempdir()
)

# View the model as well as its associated column mappings
print(model)
```

pkpdData	<i>Pharmacokinetic/Pharmacodynamic dataset containing 200 subjects with single bolus dose</i>
----------	---

Description

Pharmacokinetic/Pharmacodynamic dataset containing 200 subjects with single bolus dose.

Usage

```
pkpdData
```

Format

A data frame with 2600 rows and 5 variables:

ID Subject ID

Time Nominal Time

Dose Amount of dose

CObs Observations of drug concentration in blood

EObs Observations of drug effect

Source

The data is simulated using a PKPD model with PK described by a one-compartment model with IV bolus and PD described by an indirect response model with the loss inhibited.

```
print.NlmePmlModel      Print generic for class NlmePmlModel
```

Description

Prints model information, including PML and column mappings.

Usage

```
## S3 method for class 'NlmePmlModel'
print(x, ...)
```

Arguments

```
x          NlmePmlModel class instance
...        Arguments passed to methods.
```

Value

NULL

Examples

```
model <- pkmodel(columnMap = FALSE,
                  data = pkData,
                  workingDir = tempdir())
print(model)
```

```
randomEffect      Sets or updates the covariance matrix of random effects
```

Description

Built-in-model structural setter for the covariance matrix of random effects (omegas). Use when defining or changing diagonal vs. block membership (which etas belong to a section), together with initial values and section freeze status. Cannot be used on textual or edited models.

Usage

```
randomEffect(
  .Object,
  effect,
  value = NULL,
  isDiagonal = TRUE,
  isFrozen = FALSE,
  ...
)
```

Arguments

<code>.Object</code>	Model object
<code>effect</code>	One or more names of available random effects.
<code>value</code>	Initial values for the diagonal elements of the covariance matrix of random effects (if <code>isDiagonal = TRUE</code> , or initial values for the lower triangular elements (including diagonal elements) of the covariance matrix (if <code>isDiagonal = FALSE</code>) in a row-wise order.
<code>isDiagonal</code>	Set to <code>TRUE</code> to if the covariance matrix of the specified random effects is a diagonal matrix. or <code>FALSE</code> if not.
<code>isFrozen</code>	Set to <code>TRUE</code> to freeze the covariance matrix of random effects.
<code>...</code>	Additional arguments

Details

For value or freeze edits that keep an existing built-in section layout, see [update_Omegas\(\)](#). That updater also supports textual models for value/matrix edits through TDL5 (no freeze/unfreeze; frozen or `same()`-linked sections may reject edits; textual topology may be normalized – see [update_Omegas\(\)](#)).

Value

Modified `NlmePmlModel` object

See Also

[update_Omegas\(\)](#)

Examples

```
model <- pkmodel(
  numCompartments = 2,
  data = pkData,
  ID = "Subject",
  Time = "Act_Time",
  A1 = "Amount",
  CObs = "Conc",
  modelName = "TwCpt_IVBolus_FOCE_ELS",
  workingDir = tempdir()
)

model <-
  randomEffect(model,
    effect = c("nV", "nC1", "nC12"), value = rep(0.1, 3))
```

read_mmdl	<i>Read a metamodel file (.mmdl)</i>
-----------	--------------------------------------

Description

Imports a metamodel file into an [NlmePmlModel](#) object (textual) ready for execution. Canonical user-facing reader; supersedes the legacy [create_model_from_metamodel\(\)](#).

Usage

```
read_mmdl(file, directoryToRun = NULL)
```

Arguments

file	Path to a metamodel file (typically .mmdl).
directoryToRun	Optional working directory for the created model. If NULL (default), a subfolder named after the metamodel is used.

Details

If the dataset referenced by the `## DATA` block is missing from disk, `read_mmdl()` emits a warning containing the resolved path and returns the model with `@inputData = NULL`. Column mappings declared in the `## MAP` block are still applied. Attach the dataset later via `initColMapping(model) <- yourData` to re-run data-dependent validation. The `## PRE` script (if any) is skipped in this case because it requires the data file.

The dataset may be a plain delimited file or one written in the NLME engine's own layout (a `##name1, name2` header line, an optional `##unit1, unit2` units row, and `#` comment lines) such as the `data1.txt` found in a run directory. Units are carried on the returned data as its `Units` attribute so they are re-emitted when the model is run.

Value

A list with elements:

- `model`: an [NlmePmlModel](#)
- `params`: engine parameters (`NlmeEngineExtraParams` or a list, depending on the metamodel content) for any arguments provided in `## ESTARGS` blocks.

removeCovariate	<i>Remove covariate from structural parameters in a model object.</i>
-----------------	---

Description

Remove one or more covariates from structural parameters in a model object.

Usage

```
removeCovariate(.Object, covariate = NULL, paramName = NULL)
```

Arguments

.Object	Model object
covariate	Covariates to remove from model. If NULL all covariates will be removed from model.
paramName	Structural parameters for which to remove covariate effect(s) from. If NULL covariate effect will be removed from all structural parameters.

Value

Modified NlmePmlModel object

Examples

```
model <- pkmodel(
  numCompartments = 2,
  data = pkData,
  ID = "Subject",
  Time = "Act_Time",
  A1 = "Amount",
  CObs = "Conc",
  workingDir = tempdir()
)

# Add Gender covariate of type categorical
model <- addCovariate(model,
  covariate = "Gender",
  type = "Categorical",
  effect = c("V2", "Cl2"),
  levels = c(0, 1),
  labels = c("Female", "Male")
)

# Add BodyWeight covariate of type continuous
model <- addCovariate(model,
  covariate = "BodyWeight",
  type = "Continuous",
  direction = "Backward",
```

```

    center = "Mean",
    effect = c("V", "C1")
  )

  # Remove all covariates from model
  model <- removeCovariate(model)

```

remove_NLMELicense	<i>Remove NLME License</i>
--------------------	----------------------------

Description

This function attempts to remove an NLME license using the specified installation directory and licensing tool.

Usage

```
remove_NLMELicense(InstallDir = Sys.getenv("INSTALLDIR"))
```

Arguments

InstallDir	A character string specifying the directory where the NLME Engine is installed e.g., INSTALLDIR environment variable. The cadlicensingtool executable is expected to be located within this directory, or within a subdirectory specified by the PML_BIN_DIR environment variable.
------------	--

Details

The function checks for the presence of the necessary appsettings.json file in the specified directory or the CAD config file specified by the CAD_CONFIG_FILE environment variable, runs the licensing tool to log out the user, and attempts to remove the NLME license.

Value

A logical value indicating whether the license information was successfully removed.

Examples

```

## Not run:
INSTALLDIR <- Sys.getenv("INSTALLDIR")
if (INSTALLDIR == "") INSTALLDIR <- "C:/Program Files/Certara/NLME_Engine"
if (FALSE) { # to prevent unintended logout
  result <- remove_NLMELicense(INSTALLDIR)
}

## End(Not run)

```

residualEffectNames	<i>Return residual effect terms available in model</i>
---------------------	--

Description

Use to return character vector of residual effect names in model object

Usage

```
residualEffectNames(model)
```

Arguments

model	Object of class <code>NlmePmlModel</code>
-------	---

Value

Character vector of residual effect names

Examples

```
model <- pkemaxmodel(columnMap = FALSE)
residualEffectNames(model)
```

scmSearchTable	<i>Presentation table for SCM search results</i>
----------------	--

Description

Returns a plain data.frame for an `scmSearchResult` (or an archived run folder), shaped for a table formatter. Default `print()` / `summary()` output stays compact.

Usage

```
scmSearchTable(x, full = FALSE)
```

Arguments

x	An <code>scmSearchResult</code> from stepwiseSearch or shotgunSearch , or a path to an archived run folder.
full	Logical. Include the degrees-of-freedom column (df, stepwise only). Default FALSE.

Details

Stepwise reads StepwiseDetails.csv. Chosen becomes "Y" / "N", and the value columns are named from the Criteria column (Base -2LL, New AIC, ...); mixed or missing criteria keep the engine names BaseValue / NewValue / dValue. Model is the effect tested in that row (see Details).

Shotgun reads the archived scenario index merged with Overall.csv, exposing CovEffects (the enabled fixed effects) next to the fit criteria.

Certara.RsNLME does not import a table formatter; pass the result to one, e.g. `flextable::flextable(tab)`.

For an archived result the table comes from the archive files, so subsetting `x` first does not reduce the rows.

CovEffects in StepwiseDetails.csv is the *cumulative* enabled set for a candidate, not the single effect under test, so Model is recovered as a set difference within each Step: the step's base set is the intersection of its candidates when adding and their union when deleting. Steps with a single candidate or an unrecognised direction keep the full CovEffects string.

Value

A data.frame.

See Also

[stepwiseSearch](#), [shotgunSearch](#), [print.scmSearchResult](#), [summary.scmSearchResult](#)

Examples

```
## Not run:
tab <- scmSearchTable(stepwiseResult)
flextable::flextable(tab)

scmSearchTable(shotgunResult)
scmSearchTable(attr(stepwiseResult, "searchRunDir"))

## End(Not run)
```

secondaryParameterNames

Get secondary parameter names

Description

Returns character vector of secondary parameter names for model object.

Usage

```
secondaryParameterNames(model)
```

Arguments

model Object of class `NlmePmlModel`

Value

Character vector of secondary parameter names defined in model

Examples

```
model <- pkemaxmodel(columnMap = FALSE)
secondaryparams <- secondaryParameterNames(model)
```

shotgunSearch	<i>Executes an NLME shotgun covariate search</i>
---------------	--

Description

Executes an NLME shotgun covariate search

Usage

```
shotgunSearch(
  model,
  hostPlatform = NULL,
  params,
  covariateModel,
  runInBackground = FALSE,
  archiveResults = TRUE,
  runLabel = NULL,
  ...
)
```

Arguments

model	PK/PD model class object.
hostPlatform	Host definition for model execution. See hostParams . If missing, multicore local host with 4 threads is used.
params	Engine parameters. See engineParams . If missing, default parameters generated by <code>engineParams(model)</code> are used.
covariateModel	Covariate Effects Model providing the relationship between covariates and structural parameters to test (<code>covariateModel(model)</code>).
runInBackground	Logical. When TRUE, the wrapper starts the engine asynchronously and returns a job object immediately; pass that object to collectJob() when the run has

finished to obtain the typed result. When FALSE (the default), the wrapper blocks until the engine completes and returns the result directly.

Background execution is supported only on **Linux** hosts, whether local or remote: a local host whose `hostType` is "linux" (the default on Linux workstations), or a remote host with `hostType` "linux", "RHEL", or "UBUNTU". It is not supported on Windows (`hostType` = "windows", including the default local host when R runs on Windows): leave the argument at FALSE. Passing TRUE on a Windows host stops with an error. Remote Windows hosts are not supported at all.

<code>archiveResults</code>	Logical. When TRUE (default), NLME8 archives per-scenario results into a self-contained run folder under the model working directory. The returned data frame carries a <code>searchRunDir</code> attribute pointing to the archive folder. Archive failures are non-fatal (warn only).
<code>runLabel</code>	Optional character string appended to the auto-generated timestamp in the archive folder name, e.g. "baseModel" produces <code>shotgun_20260319_143045_baseModel</code> . Must contain only letters, digits, dots, hyphens, or underscores. If the resulting folder already exists, a numeric suffix (<code>_1</code> , <code>_2</code> , ...) is added with a warning. Ignored (with a warning) when <code>archiveResults</code> = FALSE.
...	Additional arguments for <code>hostParams</code> or arguments available inside <code>engineParams</code> functions. If <code>engineParams</code> arguments are supplied through both <code>params</code> argument and additional argument (i.e., ellipsis), then the arguments in <code>params</code> will be ignored and only the additional arguments will be used with warning. If <code>hostParams</code> arguments are supplied through both the <code>hostPlatform</code> argument and the ellipses, values supplied to <code>hostPlatform</code> will be overridden by additional arguments supplied via the ellipses e.g., ...

Value

if `runInBackground` = FALSE, an `scmSearchResult` data frame is returned with shotgun search results, i.e. the "Overall" comma separated file, plus a custom print method and a `searchType` attribute. The data frame also carries the run context as attributes: `params` (resolved `NlmeEngineExtraParams`), `runMode` ("shotgun"), `runTime` (wall-clock list(`start`, `end`, `elapsed`) measured around the engine call), and `RsNLMEVersion` (the package version that produced the result). If archiving succeeds it also carries a `searchRunDir` attribute. Otherwise (when `runInBackground` = TRUE) the `ShotgunNlmeJob` class object is returned. Backgrounded jobs can be materialised later via `collectJob`, which produces the same `scmSearchResult` (with archive when `archiveResults` = TRUE). Use `summary()` for a structured digest (best scenario, top ranking, run context). Use `scmSearchTable` for a presentation data.frame (stepwise decision table, or shotgun ranking plus covariate effects).

See Also

`hostParams`, `engineParams`, `collectJob`, `summary()`, `print()`, `scmSearchTable`, `getSCMResults`

Examples

```
## Not run:
# Define the model
```

```

model <- pkmodel(numCompartments = 2,
                 data = pkData,
                 ID = "Subject",
                 Time = "Act_Time",
                 A1 = "Amount",
                 CObs = "Conc",
                 workingDir = tempdir())

# Add Gender covariate of type categorical
model <- addCovariate(model,
                     covariate = "Gender",
                     type = "Categorical",
                     effect = c("V2", "Cl2"),
                     levels = c(0, 1),
                     labels = c("Female", "Male"))

# Add Bodyweight covariate of type continuous
model <- addCovariate(model,
                     covariate = "BodyWeight",
                     type = "Continuous",
                     direction = "Backward",
                     center = "Mean",
                     effect = c("V", "Cl"))

# Define the host
host <- hostParams(parallelMethod = "MULTICORE",
                  hostName = "local",
                  numCores = 8,
                  sharedDirectory = tempdir())

# Define the engine parameters
params <- engineParams(model, fastOptimization = TRUE, numIterations = 7)

# Define covariate model
cp <- covariateModel(model)

# Perform shotgun search
OverallIDF <- shotgunSearch(model = model,
                           hostPlatform = host,
                           params = params,
                           covariateModel = cp,
                           runInBackground = FALSE)

## End(Not run)

```

simmodel

Executes an NLME simulation

Description

Executes an NLME simulation

Usage

```
simmodel(
  model,
  simParams,
  params,
  hostPlatform = NULL,
  runInBackground = FALSE,
  ...
)
```

Arguments

model	PK/PD model class object.
simParams	Simulation parameters. See NlmeSimulationParams . If missing, default parameters generated by <code>NlmeSimulationParams()</code> are used.
params	Engine parameters. See engineParams . The common parameters include: sort, ODE, rtolODE, atolODE, maxStepsODE. If missing, default parameters generated by <code>engineParams(model)</code> are used.
hostPlatform	Host definition for model execution. See hostParams . If missing, simple local host is used.
runInBackground	Logical. When TRUE, the wrapper starts the engine asynchronously and returns a job object immediately; pass that object to <code>collectJob()</code> when the run has finished to obtain the typed result. When FALSE (the default), the wrapper blocks until the engine completes and returns the result directly. Background execution is supported only on Linux hosts, whether local or remote: a local host whose <code>hostType</code> is "linux" (the default on Linux workstations), or a remote host with <code>hostType</code> "linux", "RHEL", or "UBUNTU". It is not supported on Windows (<code>hostType</code> = "windows", including the default local host when R runs on Windows): leave the argument at FALSE. Passing TRUE on a Windows host stops with an error. Remote Windows hosts are not supported at all.
...	Additional class initializer arguments for NlmeSimulationParams , or arguments available inside hostParams or engineParams functions. If engineParams arguments are supplied through both <code>params</code> argument and additional argument (i.e., ellipsis), then the arguments in <code>params</code> will be ignored and only the additional arguments will be used with warning. If hostParams arguments are supplied through both <code>hostPlatform</code> argument and additional argument, then its slots will be overridden by additional arguments. In addition, if NlmeSimulationParams arguments are supplied through both <code>simParams</code> argument and additional argument, then its slots will be overridden by additional arguments.

Value

If `runInBackground` = FALSE, a named list of data.tables loaded from the engine's simulation outputs (any non-empty user-defined simulationTables, plus `predout.csv` / `simout.csv` depending on `params@isPopulation`). The list also carries `runMode` = "simulation", `runTime`, and

RsNLMEVersion elements alongside the data. Otherwise an NlmeSimulationJob object that can be materialised later via `collectJob()`, which produces the same result list.

See Also

`collectJob`, `vpcmodel`

Examples

```
## Not run:
SimTableObs <- tableParams(
  name = "SimTableObs.csv",
  timesList = "0,1,2,4,4.9,55.1,56,57,59,60",
  variablesList = "C, CObs",
  timeAfterDose = FALSE,
  forSimulation = TRUE
)

simParams <- NlmeSimulationParams(
  numReplicates = 2,
  simulationTables = SimTableObs
)
# Define the model
model <- pkmodel(
  numComp = 2,
  absorption = "Extravascular",
  ID = "Subject",
  Time = "Act_Time",
  CObs = "Conc",
  Aa = "Amount",
  data = pkData,
  modelName = "PkModel",
  workingDir = tempdir()
)

host <- hostParams(
  sharedDirectory = tempdir(),
  parallelMethod = "NONE",
  hostName = "local",
  numCores = 1
)

results <- simmodel(model, simParams, hostPlatform = host)
# with seed given additionally:
results <- simmodel(model, simParams, hostPlatform = host, seed = 3527)

## End(Not run)
```

sortfit	<i>Executes an NLME simple estimation with sort keys and given scenarios</i>
---------	--

Description

Executes an NLME simple estimation with sort keys and given scenarios

Usage

```
sortfit(
  model,
  hostPlatform = NULL,
  params,
  sortColumns,
  scenarios = list(),
  simpleTables,
  runInBackground = FALSE,
  filesToReturn = "*",
  ...,
  saveResult = TRUE
)
```

Arguments

model	PK/PD model class object.
hostPlatform	Host definition for model execution. See hostParams . If missing, PhoenixM-PIDir64 is given and MPI is installed, MPI local host with 4 threads is used. If MPI is not found, local host without parallelization is used.
params	Engine parameters. See engineParams . If missing, default parameters generated by engineParams(model) are used.
sortColumns	List of sort columns. See SortColumns . If missing, empty sort columns argument is used and NLME dataset is used as is.
scenarios	List of scenarios with different sets of covariates. See NlmeScenario If missing, all covariates effects are considered as enabled.
simpleTables	Optional list of simple tables. See tableParams . By default a table named 'posthoc.csv' is returned with structural parameters values for all source data rows.
runInBackground	Logical. When TRUE, the wrapper starts the engine asynchronously and returns a job object immediately; pass that object to collectJob() when the run has finished to obtain the typed result. When FALSE (the default), the wrapper blocks until the engine completes and returns the result directly. Background execution is supported only on Linux hosts, whether local or remote: a local host whose hostType is "linux" (the default on Linux workstations), or a remote host with hostType "linux", "RHEL", or "UBUNTU". It is

	not supported on Windows (<code>hostType = "windows"</code> , including the default local host when R runs on Windows): leave the argument at <code>FALSE</code> . Passing <code>TRUE</code> on a Windows host stops with an error. Remote Windows hosts are not supported at all.
<code>filesToReturn</code>	Used to specify which files to be outputted to the model directory and loaded as returned value. By default, all the applicable files listed in the <code>Value</code> section will be outputted to the model directory and loaded as returned value. Only those files listed in the <code>Value</code> section can be specified. Simple regex patterns are supported for the specification.
<code>...</code>	Additional arguments for <code>hostParams</code> or arguments available inside <code>engineParams</code> functions. If <code>engineParams</code> arguments are supplied through both <code>params</code> argument and additional argument (i.e., ellipsis), then the arguments in <code>params</code> will be ignored and only the additional arguments will be used with warning. If <code>hostParams</code> arguments are supplied through both the <code>hostPlatform</code> argument and the ellipses, values supplied to <code>hostPlatform</code> will be overridden by additional arguments supplied via the ellipses e.g., <code>...</code>
<code>saveResult</code>	When <code>TRUE</code> (the default), <code>collectJob()</code> writes a self-describing <code><workingDir>/sortfit_<sanitizedModelName></code> alongside the result. <code>sanitizedModelName</code> is derived from <code>model@modelInfo@modelName</code> by replacing every character outside <code>[A-Za-z0-9._-]</code> with <code>_</code> ; if the result is empty or contains no alphanumerics, the literal string <code>model</code> is used instead. The timestamp is the wall-clock start of the engine call, formatted as <code>YYYYMMDD_HHMMSS</code> in the local time zone. Honoured on both foreground and background paths (background runs save the RDS at the user-facing <code>collectJob(job)</code> call).

Details

All the results in tabular format have scenario column and sorts columns appended. The resulted logs (`nlme7engine.log`, `err1.txt`, `dmp.txt`, `out.txt`) are appended with a row delimiter where the name of the Scenario and sort values are specified.

Value

If `runInBackground = FALSE`, the same named list of data frames `fitmodel` produces, with scenario and sort columns appended (see Details). The list carries `params`, `runMode = "sortfit"`, `runTime`, and `RsNLMEVersion` attributes. Otherwise a `SortByNlmeJob` object; materialise later via `collectJob()`, which produces the same result list (and writes the RDS when `saveResult = TRUE`).

Non-loaded but returned files

The non-loaded but returned files in the model working directory are:

- `err1.txt` - concatenated for all runs detailed logs for all steps of optimization,
- `out.txt` - general pivoted information about results,
- `doses.csv` - information about doses given for all subjects,
- `iniest.csv` - information about initial estimates

See Also

[hostParams](#), [engineParams](#), [collectJob](#), [SortColumns](#), [NlmeScenario](#), [tableParams](#)

Examples

```
## Not run:
input_data <- pkData

model <-
  pkmodel(numCompartments = 2,
    data = input_data,
    ID = "Subject",
    Time = "Act_Time",
    A1 = "Amount",
    CObs = "Conc",
    workingDir = tempdir())

model <-
  addCovariate(model,
    covariate = "BodyWeight",
    direction = "Backward",
    center = "Mean",
    effect = c("V", "Cl"))

# multicore
multicoreHost <-
  hostParams(parallelMethod = "Multicore",
    hostName = "multicore",
    numCores = 4,
    sharedDirectory = tempdir())

# specify scenarios
CovariateEffectNames <- listCovariateEffectNames(model)
combinations <-
  combn(c("", CovariateEffectNames),
    length(CovariateEffectNames),
    simplify = FALSE)

scenarioNames <-
  lapply(combinations,
    function(x) {paste(x, collapse = " ")})

scenarios <-
  lapply(scenarioNames,
    function(x, CovariateEffectNames) {
      CovariateCombinations <- unlist(strsplit(x, " ", fixed = TRUE))
      scenarioIndex <-
        paste(which(CovariateEffectNames %in% CovariateCombinations,
          arr.ind = TRUE),
          collapse = ", ")
      NlmeScenario(trimws(x), scenarioIndex)
    },
    CovariateEffectNames)

res <-
  sortfit(model,
```

```

    hostPlatform = multicoreHost,
    params = engineParams(model, numIterations = 5, fastOptimization = TRUE),
    sortColumns = SortColumns("Gender"),
    scenarios = scenarios)

## End(Not run)

```

stepwiseSearch	<i>Executes an NLME stepwise covariate search</i>
----------------	---

Description

Executes an NLME stepwise covariate search

Usage

```

stepwiseSearch(
  model,
  hostPlatform = NULL,
  params,
  covariateModel,
  stepwiseParams,
  runInBackground = FALSE,
  archiveResults = TRUE,
  runLabel = NULL,
  updateInitialEstimates = FALSE,
  ...
)

```

Arguments

model	PK/PD model class object.
hostPlatform	Host definition for model execution. See hostParams . If missing, multicore local host with 4 threads is used.
params	Engine parameters. See engineParams . If missing, default parameters generated by <code>engineParams(model)</code> are used.
covariateModel	Covariate Effects Model providing the relationship between covariates and structural parameters to test (<code>covariateModel(model)</code>).
stepwiseParams	Stepwise parameters defining decision tree. See StepwiseParams
runInBackground	Logical. When TRUE, the wrapper starts the engine asynchronously and returns a job object immediately; pass that object to collectJob() when the run has finished to obtain the typed result. When FALSE (the default), the wrapper blocks until the engine completes and returns the result directly.

Background execution is supported only on **Linux** hosts, whether local or remote: a local host whose `hostType` is "linux" (the default on Linux workstations), or a remote host with `hostType` "linux", "RHEL", or "UBUNTU". It is not supported on Windows (`hostType` = "windows", including the default local host when R runs on Windows): leave the argument at FALSE. Passing TRUE on a Windows host stops with an error. Remote Windows hosts are not supported at all.

<code>archiveResults</code>	Logical. When TRUE (default), NLME8 archives per-scenario results into a self-contained run folder under the model working directory. The returned data frame carries a <code>searchRunDir</code> attribute pointing to the archive folder. Archive failures are non-fatal (warn only).
<code>runLabel</code>	Optional character string appended to the auto-generated timestamp in the archive folder name, e.g. "baseModel" produces <code>stepwise_20260319_143045_baseModel</code> . Must contain only letters, digits, dots, hyphens, or underscores. If the resulting folder already exists, a numeric suffix (<code>_1</code> , <code>_2</code> , ...) is added with a warning. Ignored (with a warning) when <code>archiveResults</code> = FALSE.
<code>updateInitialEstimates</code>	Logical. When TRUE, after each stepwise selection the shared input model's initial estimates are rewritten from the winning scenario's <code>dmp.txt</code> so subsequent candidate runs start from the previous step's estimates. Speeds up convergence on later steps and tends to stabilise the search. Default FALSE preserves the existing behavior. Implementation lives in NLME8; this flag is forwarded via the <code>NLME_SCM_UPDATE_INITIALS</code> environment variable (also propagated to remote run scripts).

Notes:

- When TRUE, the base (no-covariate) model runs on its own first and its converged estimates are written into the shared input model before the first forward-addition candidate batch, so that round starts from the base model's final estimates. When FALSE, the base model and the first-round candidates run together in a single parallel batch, because those candidate fits do not depend on the base estimates.
- Every selection that updates the model invalidates the stepwise compile cache, so the next batch of candidate runs will recompile the NLME executable from the rewritten model. Expect a small per-step compile-time overhead in exchange for the convergence benefit.
- The user's input `.mdl` file in the model working directory is snapshotted at the start of the search and restored on exit, so the working directory is left untouched regardless of how the search ends. The per-scenario archive (`initial.mdl`, `wiped.mdl`, `final.mdl`, etc.) is sourced from the per-scenario engine job directories and accurately reflects the evolving initials each scenario actually ran with.
- Only candidate models that are evaluated for the first time after a selection take advantage of the carried-forward initials; any candidate whose covariate set was already evaluated in an earlier step is reused from the search history with its original estimates, so each unique covariate configuration is fit exactly once per search. This keeps each mask's OFV consistent across the forward and backward phases.

... Additional arguments for [hostParams](#) or arguments available inside [engineParams](#) functions. If [engineParams](#) arguments are supplied through both `params` argument and additional argument (i.e., ellipsis), then the arguments in `params` will be ignored and only the additional arguments will be used with warning. If [hostParams](#) arguments are supplied through both the `hostPlatform` argument and the ellipses, values supplied to `hostPlatform` will be overridden by additional arguments supplied via the ellipses e.g., ...

Value

if `runInBackground = FALSE`, an `scmSearchResult` data frame is returned with stepwise search results, i.e. the "Overall" comma separated file, plus a custom print method and a `searchType` attribute. The data frame also carries the run context as attributes: `params` (resolved `NlmeEngineExtraParams`), `runMode` ("stepwise"), `runTime` (wall-clock list(`start`, `end`, `elapsed`) measured around the engine call), and `RsnlmeVersion` (the package version that produced the result). If archiving succeeds it also carries a `searchRunDir` attribute. Otherwise (when `runInBackground = TRUE`) the `StepwiseNlmeJob` class object is returned. Backgrounded jobs can be materialised later via [collectJob](#), which produces the same `scmSearchResult` (with archive when `archiveResults = TRUE`). Use [summary\(\)](#) for a structured digest (best scenario, top ranking, run context). Use [scmSearchTable](#) for a presentation data.frame (stepwise decision table, or shotgun ranking plus covariate effects).

See Also

[hostParams](#), [engineParams](#), [collectJob](#), [summary\(\)](#), [print\(\)](#), [scmSearchTable](#), [getSCMResults](#)

Examples

```
## Not run:
# Define the model
model <- pkmodel(numCompartments = 1,
                 data = pkData,
                 ID = "Subject",
                 Time = "Act_Time",
                 A1 = "Amount",
                 CObs = "Conc",
                 workingDir = tempdir())

# Add Gender covariate of type categorical
model <- addCovariate(model,
                     covariate = "Gender",
                     type = "Categorical",
                     effect = c("V", "Cl"),
                     levels = c(0, 1),
                     labels = c("Female", "Male"))

# Add Bodyweight covariate of type continuous
model <- addCovariate(model,
                     covariate = "BodyWeight",
                     type = "Continuous",
                     direction = "Backward",
```

```

        center = "Mean",
        effect = c("V", "C1"))

# Define the host
defaultHost <- hostParams(parallelMethod = "MULTICORE",
                          hostName = "local",
                          numCores = 8,
                          sharedDirectory = tempdir())

# Define the engine parameters
params <- engineParams(model, numIterations = 6)

# Define covariate model
cp <- covariateModel(model)

# Define the stepwise parameters
sp <- StepwiseParams(0.01, 0.001, "-2LL")

# Perform stepwise search
OverallDF <- stepwiseSearch(model = model,
                           hostPlatform = defaultHost,
                           params = params,
                           covariateModel = cp,
                           stepwiseParams = sp,
                           runInBackground = FALSE)

## End(Not run)

```

structuralParameter	<i>Set structural parameter in model object</i>
---------------------	---

Description

Use to specify the relationship of the structural parameter with corresponding fixed effect, random effect, and covariate.

Usage

```

structuralParameter(
  .Object,
  paramName,
  fixedEffName = NULL,
  randomEffName = NULL,
  style = "LogNormal",
  hasRandomEffect = NULL
)

```

Arguments

<code>.Object</code>	Model object
<code>paramName</code>	Name of the structural parameter
<code>fixedEffName</code>	Name of the corresponding fixed effect
<code>randomEffName</code>	Name of the corresponding random effect; only applicable to population models.
<code>style</code>	Use to specify the relationship of the structural parameter with its corresponding fixed effect, random effect, and covariate, if exists. • "LogNormal" (Default): The structural parameter is defined as $\text{Product} * \exp(\text{Eta})$ • "LogNormal1": The structural parameter is defined as $\text{Sum} * \exp(\text{Eta})$ • "LogNormal2": The structural parameter is defined as $\exp(\text{Sum} + \text{Eta})$ • "LogitNormal": The structural parameter is defined as $\text{ilogit}(\text{Sum} + \text{Eta})$ • "Normal": The structural parameter is defined as $\text{Sum} + \text{Eta}$ Product denotes the product of the corresponding fixed effect and covariate effect terms (if exists), Eta represents the corresponding random effect, and Sum denotes the sum of its corresponding fixed effect and covariate effect terms (if exists).
<code>hasRandomEffect</code>	Set to FALSE to remove the corresponding random effect from the model. Only applicable to population models. If NULL the system will automatically set <code>hasRandomEffect = TRUE</code> for population models, and <code>hasRandomEffect = FALSE</code> for individual models.

Value

Modified `NlmePmlModel` object

Examples

```
model <- pkindirectmodel(
  indirectType = "LimitedInhibition",
  isBuildup = FALSE,
  data = pkpdData,
  ID = "ID",
  Time = "Time",
  A1 = "Dose",
  CObs = "CObs",
  EObs = "EObs"
)

# Change style of structural parameter "Imax" to "LogitNormal"
# and rename fixed effect to "tvlogitImax"
model <- structuralParameter(model,
  paramName = "Imax",
  style = "LogitNormal", fixedEffName = "tvlogitImax"
)
```

```
# Remove random effect for structural parameter "IC50"
model <- structuralParameter(model,
  paramName = "IC50",
  hasRandomEffect = FALSE
)
```

structuralParameterNames

Get structural parameter names

Description

Returns character vector of structural parameter names for model object.

Usage

```
structuralParameterNames(model, omitEmpties = TRUE)
```

Arguments

model	Object of class <code>NlmePmlModel</code>
omitEmpties	Set to TRUE to omit empty names

Value

Character vector of structural parameter names defined in model

Examples

```
model <- pkemaxmodel(columnMap = FALSE)
stparams <- structuralParameterNames(model)
```

tableParams

Wrapper around NlmeTableDef/NlmeSimTableDef-classes initializers.

Description

Wrapper around NlmeTableDef/NlmeSimTableDef-classes initializers.

Usage

```
tableParams(
  name = "",
  timesList = numeric(0),
  covrSet = "",
  whenDose = "",
  whenObs = "",
  variablesList = "",
  keepSource = FALSE,
  timeAfterDose = FALSE,
  IRES = FALSE,
  Weight = FALSE,
  IWRES = FALSE,
  mode = "all",
  forSimulation = FALSE
)
```

Arguments

name	Name of the generated simulation file.
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. See covariateNames
whenDose	Character; Vector of dosing compartment names. 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.
Weight	Logical; Set to TRUE to output the weight of current observation. Valid only if whenObs is specified.
IWRES	Logical; Set to TRUE to output individual weighted residuals. Valid only if whenObs is specified.
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. Option "all" (default): it outputs all the rows invoked by specified covariates. Option "unique": if the values in a row are the same as the ones in the previous row for the current subject, then the row is omitted; otherwise, it is printed out. Option "first": it outputs only the first row for each subject.

forSimulation logical. Defining whether the table is for simulation purposes or for postprocessing after fit. Default is FALSE.

Value

NlmeTableDef object if forSimulation is FALSE, NlmeSimTableDef object otherwise.

Examples

```
Table1 <- tableParams(  
  name = "Table1.csv",  
  timesList = seq(0, 24, 2),  
  whenObs = c("CObs"),  
  variablesList = "C",  
  IRES = TRUE,  
  IWRES = TRUE,  
  Weight = TRUE)  
  
SimTable1 <- tableParams(  
  name = "SimTable1.csv",  
  variablesList = "CL, V",  
  keepSource = TRUE,  
  forSimulation = TRUE)
```

textualmodel	Create a textual model object
--------------	-------------------------------

Description

Use to create an empty model object and optionally supply location of .mdl file or PML code as a character string to initialize model with PML statements.

Usage

```
textualmodel(modelName = "", workingDir = "", data, mdl = NULL, pml = NULL)
```

Arguments

modelName	Model name to create subdirectory for model output in current working directory.
workingDir	Working directory to run the model. Current working directory will be used if workingDir not specified.
data	Input dataset
mdl	File path specifying location of .mdl file. Cannot be used together with pml.
pml	PML code as a character string. Cannot be used together with mdl.

Value

NlmePmlModel object

Examples

```
model <- textualmodel(data = pkData)
```

update_Omegas	<i>Update omega (random-effect) initial values and freeze status</i>
---------------	--

Description

Tidy/pipe-friendly updater for the covariance matrix of random effects (omegas). Complements `randomEffect()`, which remains the built-in structural setter for creating or changing diagonal vs. block membership.

Usage

```
update_Omegas(model, ..., freeze = NULL, unfreeze = NULL)
```

Arguments

model	Model object (built-in or textual)
...	Named diagonal specs and/or dimnamed matrices; see Details
freeze	Character vector of eta names whose containing section(s) should be frozen. Applicable to built-in models only.
unfreeze	Character vector of eta names whose containing section(s) should be unfrozen. Applicable to built-in models only.

Details

On **built-in** models, edit values or supply a full dimnamed matrix for an existing section, and use `freeze=unfreeze=` to toggle section freeze status without changing topology.

On **textual** models, value/matrix edits go through TDL5 override. Freeze and unfreeze are not supported and raise an error. Frozen or `same()`-linked sections may reject value edits. The textual override rebuilds from the complete omega matrix as a single `ranef(block(...))` clause, so section topology (diag vs block, multiple `ranef()` sections) is not preserved; use this path for numeric edits, not layout-sensitive textual models.

Each named argument in ... is either:

- a diagonal spec keyed by eta name – a scalar, or a named vector with `initial` (alias value); or
- a base R `matrix()` with matching row/column `dimnames` giving eta names, supplying the full covariance values for an existing block/diagonal section (the ... name itself is decorative).

Unlike `update_Thetas()`, a spec here cannot carry `freeze`: PML has no per-eta `freeze` inside a `diag()/block()` section, so freezing is necessarily a property of the whole section and is requested through `freeze=unfreeze=` instead. A `freeze` field inside a spec raises an error naming that alternative.

Occasion random effects are supported for both value and matrix edits, targeting the covariate's reference level; a matrix edit must name every random effect of that occasion covariate, because the stored values cover its whole inter-occasion covariance matrix. Editing a `same()`-linked eta directly raises an error naming the reference level on built-in models, or a post-hoc error on textual models (which cannot detect `same()` ahead of time).

Value

Modified `NlmePmlModel` object

See Also

`randomEffect()`

Examples

```
model <- pkmodel(numCompartments = 2, columnMap = FALSE, workingDir = tempdir())
model <- update_Omegas(model, nV = 0.15, nCl = 0.1)

# A dimnamed matrix sets a whole section at once; the dimnames identify the
# section, so the argument name is decorative.
model <- randomEffect(model,
  effect = c("nV", "nCl"), isDiagonal = FALSE, value = c(0.1, 0.01, 0.1)
)
blockOmega <- matrix(
  c(0.2, 0.03, 0.03, 0.15), nrow = 2,
  dimnames = list(c("nV", "nCl"), c("nV", "nCl"))
)
model <- update_Omegas(model, omegaBlock = blockOmega)

# freeze= acts on the whole section containing the named eta
model <- update_Omegas(model, freeze = "nV")
```

update_Sigmas

Update residual-error (sigma) initial values and freeze status

Description

Tidy/pipe-friendly updater for residual-error standard deviations (sigmas). Complements `residualError()`, which remains the built-in structural setter for changing error *type*, *BQL/LLOQ*, or power configuration.

Usage

```
update_Sigmas(model, ..., freeze = NULL, unfreeze = NULL)
```

Arguments

model	Model object (built-in or textual)
...	Named sigma specs keyed by epsilon or observation name; see Details
freeze	Character vector of epsilon/observation names to freeze. Applicable to built-in models only.
unfreeze	Character vector of epsilon/observation names to unfreeze. Applicable to built-in models only.

Details

For **built-in** models, the standard deviation of the residual error (SD) can be edited, and it can also be frozen or unfrozen through the `freeze=/unfreeze=` arguments (or by specifying `freeze` in a named spec). These edits do not change the error type. For **textual** models, SD edits go through TDL5 override; `freeze` and `unfreeze` are not supported and raise an error. In addition, a residual error whose `error()` statement declares `freeze` cannot be updated and raises an error.

Each named argument in `...` identifies a residual error by either its epsilon name (e.g. `CEps`) or its observation name (e.g. `CObs`). The argument can be given as either a scalar value (the initial SD) or a named vector spec that defines the initial SD (`initial`, alias `value`) and whether the SD is frozen (`freeze`, alias `frozen`). Observation names are resolved on built-in models only: a built-in model stores an observation alongside its epsilon, whereas linking the two on a textual model would require parsing the `observe()` statement, so textual models must be keyed by epsilon name.

The fixed effect corresponding to the proportional component of `AdditiveMultiplicative/MixRatio` error models (e.g. `CMultStdev`) is a regular fixed effect and should be updated with [update_Thetas\(\)](#) rather than this function.

Value

Modified `NlmePmlModel` object

See Also

[residualError\(\)](#)

Examples

```
model <- pkmodel(columnMap = FALSE, workingDir = tempdir())
model <- update_Sigmas(model, CEps = c(initial = 0.05, freeze = TRUE))
```

update_Thetas	<i>Update fixed-effect (theta) initial values, bounds, freeze, and enable</i>
---------------	---

Description

Tidy/pipe-friendly updater for one or more fixed effects (thetas) on built-in and textual (edited) models. Complements `fixedEffect()`, which remains the built-in structural setter when units are needed or when configuring thetas during model construction. This function does not set units.

Usage

```
update_Thetas(model, ...)
```

Arguments

<code>model</code>	Model object (built-in or textual)
<code>...</code>	Named theta specs; see Details

Details

Each named argument in `...` specifies the theta to update and its new spec, which may be:

- a scalar numeric, which sets the initial value only;
- an unnamed length-3 numeric vector `c(lower, initial, upper)`, which sets the lower bound, initial value, and upper bound;
- a named vector with any of `lower`, `initial` (alias `value`), `upper`, `freeze` (alias `frozen`), `enable`. Assign a logical value to `freeze`: `freeze = TRUE` freezes the associated fixed effect so that it is not estimated, whereas `freeze = FALSE` unfreezes it so that it can be estimated. Assign a non-negative whole number to `enable`; this enables the associated covariate effect, using the assigned value as the enable ID, so that it can be tested during covariate search procedures.

Omitted fields are preserved from the current model. A name that is not a fixed effect of `model` raises an error, so a typo is caught rather than reported as an applied edit.

`enable` applies to textual models only; built-in models have no `enable` slot, so a warning is issued and the field is ignored. Assigning a non-negative whole number enables the associated covariate effect and uses that number as the enable ID. `enable = FALSE` (or `NA`, or `NULL`) removes the enable flag, disabling the covariate effect. A negative number, a fraction such as `1.8`, or a bare `enable = TRUE`, which carries no ID, raises an error instead of being truncated or ignored.

Supply `enable = FALSE`, `NULL`, or `TRUE` through a **list** rather than `c()`. Because `c()` coerces logicals to numbers and drops `NULL`, a vector would silently turn the request into something else: `TRUE` becomes the valid enable ID 1, `FALSE` becomes the valid enable ID 0 (leaving the effect enabled), and `NULL` disappears entirely (leaving the flag in place). All three are therefore rejected inside `c()`, with a message naming the list form to use, for example `list(initial = 10, enable = FALSE)`. An enable ID and `enable = NA` are unaffected by coercion and may be given either way.

A bound the spec omits is preserved, and a bound the model does not have stays absent rather than being reported as `-Inf/Inf`. Passing `-Inf` as `lower`, or `Inf` as `upper`, explicitly removes that bound on built-in and textual models alike.

Value

Modified `NlmePmlModel` object

See Also

[fixedEffect\(\)](#)

Examples

```
model <- pkmodel(columnMap = FALSE, workingDir = tempdir())
model <- update_Thetas(model,
  tvV = 15,
  tvCl = c(lower = 0, initial = 5, upper = 20)
)
initFixedEffects(model)

# -Inf / Inf remove a bound that is currently set
model <- update_Thetas(model, tvCl = c(lower = -Inf))
```

vpcmodel

Perform visual predictive check for NLME models

Description

Perform visual predictive check for NLME models

Usage

```
vpcmodel(
  model,
  vpcParams,
  params,
  hostPlatform = NULL,
  runInBackground = FALSE,
  ...
)
```

Arguments

<code>model</code>	PK/PD model class object.
<code>vpcParams</code>	VPC argument setup. See NlmeVpcParams . If missing, default values generated by <code>NlmeVpcParams()</code> are used.
<code>params</code>	Engine argument setup. See engineParams . The following arguments are the subject of interest: <code>sort</code> , <code>ODE</code> , <code>rtolODE</code> , <code>atolODE</code> , <code>maxStepsODE</code> . If missing, default values generated by <code>engineParams(model)</code> are used.

hostPlatform	Host definition for model execution. See hostParams . If missing, simple local host is used.
runInBackground	Logical. When TRUE, the wrapper starts the engine asynchronously and returns a job object immediately; pass that object to collectJob() when the run has finished to obtain the typed result. When FALSE (the default), the wrapper blocks until the engine completes and returns the result directly. Background execution is supported only on Linux hosts, whether local or remote: a local host whose hostType is "linux" (the default on Linux workstations), or a remote host with hostType "linux", "RHEL", or "UBUNTU". It is not supported on Windows (hostType = "windows", including the default local host when R runs on Windows): leave the argument at FALSE. Passing TRUE on a Windows host stops with an error. Remote Windows hosts are not supported at all.
...	Additional class initializer arguments for NlmeVpcParams or hostParams, or arguments available inside engineParams functions. If engineParams arguments are supplied through both params argument and additional argument (i.e., ellipsis), then the arguments in params will be ignored and only the additional arguments will be used with warning. If hostParams arguments are supplied through both hostPlatform argument and additional argument, then its values will be overridden by additional arguments. In addition, if NlmeVpcParams arguments are supplied through both vpcParams argument and additional argument, then its slots will be overridden by additional arguments.

Value

If runInBackground = FALSE, a named list of data.tables loaded from the engine's VPC outputs (predcheck*.csv files, any user-defined simulationTables, plus predout.csv / simout.csv depending on params@isPopulation). The list also carries runMode = "vpc", runTime, and RsNLMEVersion elements alongside the data. Otherwise an NlmeSimulationJob object that can be materialised later via [collectJob\(\)](#), which produces the same result list.

See Also

[collectJob](#), [simmodel](#)

Examples

```
## Not run:
model <- pkmodel(
  numComp = 1,
  absorption = "Extravascular",
  ID = "Subject",
  Time = "Act_Time",
  CObs = "Conc",
  Aa = "Amount",
  data = pkData,
  modelName = "PkModel",
  workingDir = tempdir()
)
```

```

host <- hostParams(
  sharedDirectory = tempdir(),
  parallelMethod = "NONE",
  hostName = "local",
  numCores = 1
)

job <- fitmodel(model = model,
  hostPlatform = host)

finalModelVPC <- copyModel(model,
  acceptAllEffects = TRUE,
  modelName = "model_VPC",
  workingDir = tempdir())

# View the model
print(finalModelVPC)

# Set up VPC arguments to have PRED outputted to simulation output dataset "predout.csv"
vpcSetup <- NlmeVpcParams(outputPRED = TRUE)

# Run VPC using the default host, default values for the relevant NLME engine arguments
finalVPCJob <- vpcmodel(model = finalModelVPC, vpcParams = vpcSetup, hostPlatform = host)
# the same as:
# finalVPCJob <- vpcmodel(model = finalModelVPC, outputPRED = TRUE)

# Observed dataset predcheck0.csv
dt_ObsData <- finalVPCJob$predcheck0

# Simulation output dataset predout.csv
dt_SimData <- finalVPCJob$predout

# Add PRED from REPLICATE = 0 of simulation output dataset to observed input dataset
dt_ObsData$PRED <- dt_SimData[REPLICATE == 0]$PRED

# tidyvpc package VPC example:
# library(tidyvpc)
# library(magrittr)
# Create a regular VPC plot with binning method set to be "jenks"
# binned_VPC <- observed(dt_ObsData, x = IVAR, yobs = DV) %>%
# simulated(dt_SimData, ysim = DV) %>%
# binning(bin = "jenks") %>%
# vpcstats()

# plot_binned_VPC <- plot(binned_VPC)

# Create a pcVPC plot with binning method set to be "jenks"
# binned_pcVPC <- observed(dt_ObsData, x = IVAR, yobs = DV) %>%
# simulated(dt_SimData, ysim = DV) %>%
# binning(bin = "jenks") %>%
# predcorrect(pred = PRED) %>%
# vpcstats()

```

```
# plot_binned_pcVPC <- plot(binned_pcVPC)

## End(Not run)
```

write_mmdl

Write a metamodel file (.mmdl)

Description

Writes an [NlmePmlModel](#) object to a text based metamodel file (.mmdl).

Usage

```
write_mmdl(
  model,
  file,
  datafile,
  author = "",
  engineParams = NULL,
  tableParams = NULL,
  absolutePaths = FALSE
)
```

Arguments

model	An NlmePmlModel object.
file	Output metamodel file path to create/overwrite.
datafile	Input data file path to write into the ## DATA block.
author	Optional author string to write into the metamodel.
engineParams	Optional NlmeEngineExtraParams object created by engineParams() to serialize into the ## ESTARGS block. Only arguments that differ from engineParams() defaults for this model (and estimation method) are written, so the block lists explicit overrides only.
tableParams	Optional table definition(s) created by tableParams() (i.e., NlmeTableDef and/or NlmeSimTableDef , or a list of them) to serialize into the ## TABLES block.
absolutePaths	Logical; how to write the datafile path into the ## DATA block. FALSE (default) makes it relative to the metamodel directory (e.g. ./data.csv), producing portable metamodels. TRUE writes the normalized absolute path. When a relative path cannot be expressed (different Windows drive) the function warns and falls back to absolute.

Value

Invisible file.

Index

- * **NLME**
 - hostParams, [44](#)
- * **NlmeParallelHost**
 - hostParams, [44](#)
- * **datasets**
 - OneCpt_IVInfusionData, [58](#)
 - pkcovbqlData, [59](#)
 - pkData, [60](#)
 - pkpdData, [78](#)
- addADDL, [4](#)
- addCovariate, [5](#)
- addDoseCycle, [4](#), [8](#)
- addExtraDef, [9](#)
- addInfusion, [10](#)
- addLabel, [11](#)
- addMDV, [12](#)
- addReset, [12](#)
- addReset, NlmePmlModel-method
 - (addReset), [12](#)
- addSecondary, [13](#)
- addSecondary, NlmePmlModel-method
 - (addSecondary), [13](#)
- addSteadyState, [14](#)
- bootstrap, [15](#), [19](#), [21](#)
- BootstrapParams, [15–17](#)
- cancelJob, [18](#)
- cancelJob, SimpleNlmeJob-method
 - (cancelJob), [18](#)
- collectJob, [15–17](#), [18](#), [38–40](#), [43](#), [56](#), [86](#), [87](#), [89–92](#), [94](#), [96](#), [107](#)
- collectJob, BootNlmeJob-method
 - (collectJob), [18](#)
- collectJob, FitNlmeJob-method
 - (collectJob), [18](#)
- collectJob, NlmeSimulationJob-method
 - (collectJob), [18](#)
- collectJob, ProfileNlmeJob-method
 - (collectJob), [18](#)
- collectJob, ShotgunNlmeJob-method
 - (collectJob), [18](#)
- collectJob, SortByNlmeJob-method
 - (collectJob), [18](#)
- collectJob, StepwiseNlmeJob-method
 - (collectJob), [18](#)
- colMapping, [21](#), [25](#), [27](#), [28](#), [53](#), [54](#), [62](#), [65](#), [67](#), [70](#), [72](#), [74](#), [76](#), [77](#)
- copyModel, [16](#), [22](#)
- covariateNames, [23](#), [100](#)
- create_model_from_metamodel(), [81](#)
- createModelInfo, [24](#)
- dataMapping, [21](#), [25](#)
- doseNames, [8](#), [9](#), [25](#)
- editModel, [26](#)
- emaxmodel, [27](#)
- emaxmodel_MappingParameters, [27](#)
- engineParams, [15–17](#), [28](#), [38](#), [40](#), [86](#), [87](#), [89](#), [91](#), [92](#), [94](#), [96](#), [106](#), [107](#)
- engineParams(), [109](#)
- extraDoseLines, [35](#)
- extraDoseNames, [36](#)
- fitmodel, [19–21](#), [37](#), [92](#)
- fitmodel(), [46](#)
- fixedEffect, [51](#)
- fixedEffect(), [105](#), [106](#)
- getRandomEffectNames, [41](#)
- getSCMResults, [42](#), [56](#), [87](#), [96](#)
- getThetas, [43](#)
- hostParams, [15–17](#), [38](#), [40](#), [44](#), [86](#), [87](#), [89](#), [91](#), [92](#), [94](#), [96](#), [107](#)
- imputeMissingCovariates, [47](#)
- imputeMissingCovariates(), [50](#), [51](#)

imputeMode, 50
 imputeMode(), 48
 initFixedEffects, 51
 initFixedEffects, NlmePmlModel-method
 (initFixedEffects), 51
 initFixedEffects<- (initFixedEffects),
 51
 initFixedEffects<- , NlmePmlModel-method
 (initFixedEffects), 51
 isPopulation, 52
 isPopulation, NlmePmlModel-method
 (isPopulation), 52
 isPopulation<- (isPopulation), 52
 isPopulation<- , NlmePmlModel-method
 (isPopulation), 52

 linearmodel, 53
 linearmodel_MappingParameters, 54
 listCovariateEffectNames, 54
 listCovariateEffectNames, NlmePmlModel-method
 (listCovariateEffectNames), 54
 listSCMRuns, 42, 43, 55

 modelVariableNames, 21, 56

 NlmeEngineExtraParams, 109
 NlmeJobStatus, 18, 21
 NlmePmlModel, 81, 109
 NlmeScenario, 91, 92
 NlmeSimTableDef, 109
 NlmeSimulationParams, 89
 NlmeTableDef, 109
 NlmeVpcParams, 106, 107

 obtain_NLMELicense, 57
 OneCpt_IVInfusionData, 58

 parsePMLColMap, 59
 pkcovbqlData, 59
 pkData, 60
 pkemaxmodel, 61
 pkindirectmodel, 65
 pkindirectmodel_MappingParameters, 62,
 67, 72
 pklinearmodel, 70
 pkmodel, 53, 75
 pkmodel_MappingParameters, 76
 pkpdData, 78
 print(), 43, 56, 87, 96

 print.NlmePmlModel, 79
 print.scmSearchResult, 85
 profilePertubate, 19–21

 randomEffect, 79
 randomEffect(), 102, 103
 read_mmdl, 81
 remove_NLMELicense, 83
 removeCovariate, 82
 residualEffectNames, 84
 residualError(), 103, 104
 rsnlme_boot, 16, 17, 19

 scmSearchTable, 84, 87, 96
 secondaryParameterNames, 85
 shotgunSearch, 19–21, 43, 56, 84, 85, 86
 simmodel, 19–21, 88, 107
 SortColumns, 91, 92
 sortfit, 19–21, 91
 StepwiseParams, 94
 stepwiseSearch, 19–21, 43, 56, 84, 85, 94
 structuralParameter, 97
 structuralParameterNames, 99
 summary(), 43, 56, 87, 96
 summary.scmSearchResult, 85

 tableParams, 38, 40, 91, 92, 99
 tableParams(), 109
 textualmodel, 53, 101

 update_Omegas, 102
 update_Omegas(), 80
 update_Sigmas, 103
 update_Thetas, 105
 update_Thetas(), 103, 104

 vpcmodel, 19–21, 90, 106

 write_mmdl, 109