Package ‘assemblerr’

July 28, 2021

Title  Assembly of Pharmacometric Models

Description  Construct pharmacometric nonlinear mixed effect models by combining predefined model components and automatically generate model code for NONMEM. Models are created by combining parameter and observation models, algebraic relationships, compartments, and flows. Pharmacokinetic models can be assembled from the higher-order components: absorption, distribution, and elimination. The generated code is optimized for performance by recognizing, for example, linear differential equations or differential equations with an analytic solution.

Version  0.1.0

License  MIT + file LICENSE

Encoding  UTF-8

Suggests  testthat (>= 3.0.2), knitr, rmarkdown, covr, withr, markdown

Imports  purrr, rlang, magrittr, methods, glue, vctrs (>= 0.3.4), cli
          (>= 2.1.0), tidyselect

VignetteBuilder  knitr

RoxygenNote  7.1.1

ByteCompile  true

         ‘conversion-compartment-nm.R’ ‘input_variable.R’
         ‘conversion-input_variable-nm.R’ ‘meta.R’
         ‘conversion-observation-nm.R’ ‘parameter.R’

URL  https://github.com/UUPharmacometrics/assemblerr

BugReports  https://github.com/UUPharmacometrics/assemblerr/issues
R topics documented:

assemblerr-package ........................................... 3
algebraic .......................................................... 4
assemblerr_options ............................................. 5
check ................................................................. 6
compartment ......................................................... 7
flow ................................................................. 8
input_variable ...................................................... 10
model ................................................................. 10
model-variable-selection ........................................ 11
obs_additive ......................................................... 13
obs_combined ....................................................... 14
obs_proportional .................................................. 15
pk_absorption_fo .................................................. 17
pk_absorption_fo_lag ........................................... 18
pk_absorption_fo_transit ....................................... 19
pk_absorption_fo_zo ............................................ 21
pk_absorption_zo ................................................ 22
pk_absorption_zo_lag .......................................... 23
pk_distribution_1cmp .......................................... 25
pk_distribution_2cmp .......................................... 26
pk_distribution_3cmp .......................................... 27
pk_elimination_linear .......................................... 29
pk_elimination_linear_nl ..................................... 30
pk_elimination_nl ............................................... 31
pk_model ......................................................... 33
prm_logit_normal ............................................... 33
prm_log_normal .................................................. 35
prm_normal ....................................................... 36
prm_no_var ....................................................... 38
render .............................................................. 40
Description

Construct pharmacometric nonlinear mixed effect models by combining predefined model components and automatically generate model code for NONMEM. Models are created by combining parameter and observation models, algebraic relationships, compartments, and flows. Pharmacokinetic models can be assembled from the higher-order components: absorption, distribution, and elimination. The generated code is optimized for performance by recognizing, for example, linear differential equations or differential equations with an analytic solution.

Author(s)

Maintainer: Sebastian Ueckert <sebastian.ueckert@gmail.com> (ORCID) [copyright holder]

Other contributors:

- Mats O. Karlsson [scientific advisor]
- Andrew C. Hooker [scientific advisor]
- Rikard Nordgren [scientific advisor]
- Simon Carter [reviewer]
- Simon Buatois [reviewer]
- João A. Abrantes [reviewer]
- F. Hoffmann-La Roche Ltd. [funder]

See Also

Useful links:

- https://github.com/UUPharmacometrics/assemblerr
- Report bugs at https://github.com/UUPharmacometrics/assemblerr/issues
**algebraic**  

**Algebraic relationship**

**Description**

This building block defines a model variable as a function of other variables.

**Usage**

`algebraic(definition)`

**Arguments**

- `definition` A definition of the model variable

**Details**

Algebraic relationships are equations where one variable is defined as a function of multiple other variables. assemblerr uses R formulas to implement these equations. For example, the Emax dose response model

\[ \text{effect} = \frac{\text{emax} \times \text{dose}}{\text{ed}_{50} + \text{dose}} \]

could be declared as

`algebraic(effect~emax*dose/(ed50+dose))`

where the tilde `~` replaced the equal sign `=` in the definition.

**Value**

A building block of type 'algebraic'

**Examples**

```r
m <- model() +  
input_variable("dose") +  
prm_log_normal("emax", 10, 0.3) +  
prm_no_var("ed50", 5) +  
algebraic(effect~emax*dose/(ed50+dose)) +  
ob_additive(~effect)
```
**assemblerr_options**

---

**Options**

**Description**

This function creates a list of options for the use with the render function.

**Usage**

```r
assemblerr_options(
  prm.use_mu_referencing = FALSE,
  ode.use_special_advans = TRUE,
  ode.use_general_linear_advans = TRUE,
  ode.general_nonlinear_advan = "advan13",
  ode.general_linear_advan = "advan5",
  ode.preferred_trans_routines = c("trans2", "trans4"),
  issues.missing_variables = c("fix-warn", "fix", "ignore", "fail")
)
```

**Arguments**

- `prm.use_mu_referencing`: Use mu-referencing?
- `ode.use_special_advans`: Use analytic solution ADVANs?
- `ode.use_general_linear_advans`: Use ADVANs for linear ODEs?
- `ode.general_nonlinear_advan`: ADVAN to be used for non-linear ODEs
- `ode.general_linear_advan`: ADVAN to be used for linear ODEs
- `ode.preferred_trans_routines`: Order of TRANS routines to be tried
- `issues.missing_variables`: How to handle missing variables

**Details**

The function helps to create properly formatted list that can serve as input to the options= argument of the render() function.

**Value**

A list of options
Description

This function checks a model for existing issues.

Usage

check(model)

Arguments

model Model to check

Details

The function accepts a model object and returns a list of issues that can help to identify problems in a model. If no issues are found, a message and an empty list are produced. Issues can either be critical or non-critical, depending on whether a valid model could still be rendered.

The function currently detects the following issues:

- Undefined variables
- Lack of parameters
- Lack of observations
- Lack of distribution/elimination components (pk_model)
- Inconsistent capitalization of variable names

Value

An issue list (printed to the console by default)

Examples

```r
m <- model() +
    prm_log_normal("emax") +
    prm_log_normal("ed50") +
    obs_additive(eff~emax*dose/(ed50+dose))
check(m)

# fix issue
m <- m + input_variable("dose")
check(m)
```
compartment

---

**compartment**

*Compartmen*

---

**Description**

Defines name and volume of a compartment.

**Usage**

```r
compartment(name, volume = 1)
```

```r
cmp(name, volume = 1)
```

**Arguments**

- **name**: Name of the compartment
- **volume**: Volume as a number, formula or parameter name

**Details**

In most applications, compartments contain kinetically homogeneous amount of drug (applications where the compartment content represents other quantities are also possible). In `assemblerr`, a compartment is defined by providing a name and the compartment volume.

**Compartment names:**

Every compartment must have a valid name. A compartment name can contain letters, numbers as well as the underscore character, and needs to start with a letter. Adding a compartment with an already existing name will replace the definition of the compartment.

**Compartment volumes:**

The compartment volume can be provided as a number, R formula, or a parameter name. It will be used by `assemblerr` to replace references to the compartment concentration (e.g., `~C["central"]`) with the corresponding amount divided by volume (e.g., `~A["central"]/vc`).

**Value**

A building block of type ’compartment’

**See Also**

- `flow` for how to describe compartment kinetics
Examples

```r
# model with depot and central compartment
m <- model() +
  compartment("depot", volume = 1) +
  compartment("central", volume = "vc") +
  flow(~ka*A, from = "depot", to = "central") +
  flow(~cl*C, from = "central") +
  prm_log_normal("ka") +
  prm_log_normal("cl") +
  prm_log_normal("vc") +
  obs_additive(conc~C["central"])

render(
  model = m,
  options = assemblerr_options(
    ode.use_special_advans = FALSE,
    ode.use_general_linear_advans = FALSE
  )
)
```

flow

Flow between compartments

Description

This building block describes a flow between compartments.

Usage

```r
flow(definition, from = NA_character_, to = NA_character_)
```

Arguments

- `definition`: Equation describing the flow
- `from`: Name of the source compartment (NA for an inflow without source)
- `to`: Name of the sink compartment (NA for an outflow without sink)

Details

Flows define the connections between compartments and the equations according to which exchanges occur.

Flow equations:

The first function argument is the flow equation. It is defined using R formulas that can start with the tilde `~` operator and do not need to have a left-hand side (i.e., `~k0` is a valid flow definition).

Flow equations can contain the special variables `A` and `C` which can be used to refer to the amount and concentration in the compartment specified via the `from=` argument. For example, the following code creates a flow building block describing the first-order transfer from the depot to the central compartment.
flow(~ka*A, "depot", "central")

When the model is rendered, A and C will get replaced with the corresponding compartment reference. assembler will raise an error if A or C are used without specifying the from=} compartment (such as in an inflow).

**Compartment connections:**
The connection between compartments can be specified using the from=} and to=} arguments of the function. Setting either from=} or to=} to NA allows the definition of in and outflows without a source or sink. Setting both arguments to NA results in error.

**Conversion to differential equations:**
When flows are rendered they are converted to ordinary differential equations (ODEs). The connection between compartments together with the flow equations allow assembler to determine whether an analytic solution can be generated. This automatic optimization of differential equations can be disabled via the rendering options.

**Value**
A building block of type 'flow'

**Examples**

```r
# one-compartment model with first-order elimination
m <- model() +
  prm_log_normal("v") +
  prm_log_normal("cl") +
  compartment("central", volume = ~v) +
  flow(declaration(~cl*C), from = "central") +
  obs_additive(~C["central"])
# an analytic solution is generated
render(m)

# one-compartment model with Michaelis-Menten elimination
m2 <- model() +
  prm_log_normal("v") +
  prm_log_normal("vmax") +
  prm_no_var("km") +
  compartment("central", volume = ~v) +
  flow(declaration(~vmax*C/(km+C)), from = "central") +
  obs_additive(~C["central"])
# an ODE is generated
render(m2)
```
input_variable 

**Description**

These building block declare input variables, i.e., variables that are defined in the dataset.

**Usage**

```
input_variable(name)
```

```
dataset(path)
```

**Arguments**

<table>
<thead>
<tr>
<th>name</th>
<th>Variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Dataset path</td>
</tr>
</tbody>
</table>

**Details**

An input variable is defined in the dataset and is declared so that it can be used in the rest of the model definition. The function `input_variable()` declares a single variable whereas the `dataset()` function reads the header of the file provided and declares all variables found.

**Value**

A building block of type 'input_variable'

**Examples**

```r
m <- model() +
    input_variable("dose") +
    prm_log_normal("emax") +
    prm_log_normal("ed50") +
    obs_additive(eff~emax*dose/(ed50+dose))
render(m)
```

---

model 

---

**Description**

This function creates the basis for a general pharmacometric model, a flexible but verbose model type.
Usage

model()

Details

The function creates the fundament for a general pharmacometric model to which different building blocks can be added. The following building blocks are relevant for this model type:

- Parameters: prm_log_normal, prm_logit_normal, prm_no_var, prm_normal
- Observations: obs_additive, obs_combined, obs_proportional
- Algebraic relationships: algebraic
- Compartments: compartment
- Flows: flow
- Input variables: input_variable, dataset

The more specialized pk_model() is converted to a general model during the rendering process.

Value

A general pharmacometric model

Examples

m <- model() +
    input_variable("dose") +
    prm_log_normal("emax") +
    prm_log_normal("ed50") +
    obs_additive(eff~emax*dose/(ed50+dose))

render(m)

Description

The output task allows to select model variables using a concise mini language. You can select variables by name or using one of the helper functions described below.

Overview of selection features:

The selection of variables builds on the tidyselect package which implements a powerful variable selection language (see tidyselect::language). The following features are most relevant for the selection of model variables:

- | for selecting the union of several variables
- c() for combining selections
- ! for taking the complement of a set of variables
In addition, you can select variables using a combination of the following helper functions:

- `vars_prms()` selects all model parameters
- `vars_data()` selects all data defined variables
- `vars_eta()` selects all eta variables
- `vars_nm_std()` selects the standard NONMEM variables DV, PRED, RES, WRES, IPREDI, IWRESI
- `vars_starts_with()` selects variables that start with a prefix
- `vars_matches()` selects variables that match a regular expression

Usage

```r
vars_prms(vars)
vars_data(vars)
vars_eta(vars)
vars_nm_std(vars)
vars_starts_with(match, vars)
vars_matches(match, vars)
```

Arguments

- `vars` A character vector of variable names (taken from the selection context)
- `match` A character vector to match against

Value

A selection context

Examples

```r
m <- model() +
  input_variable("dose") +
  prm_log_normal("emax", median = 10, var_log = 0.09) +
  prm_log_normal("ed50", median = 50, var_log = 0.09) +
  algebraic(effect=emax*dose/(ed50 + dose)) +
  obs_proportional(~effect, var_prop = 1)

# output all model parameter and eta variables
render(m, tasks = tsk_output("prms", variables = vars_prms() | vars_eta()))
```
Description
This building block declares an observation model with an additive residual error model ($y = f + \epsilon$).

Usage
obs_additive(prediction, name, var_add = 1)

Arguments
- prediction: A definition of the model prediction
- name: A name for the observation (automatically derived if missing)
- var_add: Variance of the additive error

Details
Observation models specify the observed variable, how an observation is expected to diverge from the model (i.e., the residual unexplained variability model), and parameter values. The observation model type is selected through the function name. The observed variable as well as the parameters are specified as function arguments.

Specifying predictions:
The actual prediction from the model is the first argument of the function. It can be specified in a number of different ways:
- A name of a variable in the model: obs_additive("effect")
- A compartment concentration: obs_additive(~C["central"])  
- An equation: obs_additive(~base+slp*time)

If the definition contains a variable name on the left-hand side (as in conc~C["central"]), the variable will appear in the generated model code. This can be useful to make the model code more readable if the prediction is defined as a long equation.

Observation names:
The observation name can be specified via the name= argument and is automatically derived if the argument is left empty. Adding an observation model with an already existing name will replace the previous definition.

Error variance:
The variance of the error components are specified via the var_add= and var_prop= arguments of the function.

Value
A building block of type 'observation'
See Also

Other observation models: \texttt{obs\_combined()}, \texttt{obs\_proportional()}

Examples

```r
# additive RUV model for observing the variable WT
m <- model() +
  prm_log_normal(“wt”) +
  obs_additive(~wt)

# EMAX dose-response model with proportional RUV
m2 <- model() +
  input_variable(“dose”) +
  prm_no_var(“emax”) +
  prm_no_var(“ed50”) +
  obs_proportional(effect=emax*dose/(ed50+dose))
```

---

**obs\_combined**

*Observation with combined error*

### Description

This building block declares an observation model with a combined residual error model \( y = f + f\epsilon_1 + \epsilon_2 \).

### Usage

```r
obs\_combined(prediction, name, var\_prop = 0.1, var\_add = 1)
```

### Arguments

- **prediction**: A definition of the model prediction
- **name**: A name for the observation (automatically derived if missing)
- **var\_prop**: Variance of the proportional error component
- **var\_add**: Variance of the additive error component

### Details

Observation models specify the observed variable, how an observation is expected to diverge from the model (i.e., the residual unexplained variability model), and parameter values. The observation model type is selected through the function name. The observed variable as well as the parameters are specified as function arguments.

**Specifying predictions:**

The actual prediction from the model is the first argument of the function. It can be specified in a number of different ways:

- A name of a variable in the model: \texttt{obs\_additive(“effect“)}
• A compartment concentration: `obs_additive(~C["central"])
• An equation: `obs_additive(~base+slp*time)

If the definition contains a variable name on the left-hand side (as in `conc~C["central"]), the variable will appear in the generated model code. This can be useful to make the model code more readable if the prediction is defined as a long equation.

**Observation names:**
The observation name can be specified via the `name=` argument and is automatically derived if the argument is left empty. Adding an observation model with an already existing name will replace the previous definition.

**Error variance:**
The variance of the error components are specified via the `var_add=` and `var_prop=` arguments of the function.

### Value
A building block of type 'observation'

### See Also
Other observation models: `obs_additive()`, `obs_proportional()`

### Examples
```r
# additve RUV model for observing the variable WT
m <- model() +
  prm_log_normal("wt") +
  obs_additive(~wt)

# EMAX dose-response model with proportional RUV
m2 <- model() +
  input_variable("dose") +
  prm_no_var("emax") +
  prm_no_var("ed50") +
  obs_proportional(effect~emax*dose/(ed50+dose))
```

---

**obs_proportional**  
*Observation with proportional error*

### Description
This building block declares an observation model with a proportional residual error model ($y = f + f e_1$).

### Usage
```r
obs_proportional(prediction, name, var_prop = 0.1)
```
Arguments

- **prediction**: A definition of the model prediction
- **name**: A name for the observation (automatically derived if missing)
- **var_prop**: Variance of the proportional error

Details

Observation models specify the observed variable, how an observation is expected to diverge from the model (i.e., the residual unexplained variability model), and parameter values. The observation model type is selected through the function name. The observed variable as well as the parameters are specified as function arguments.

**Specifying predictions:**

The actual prediction from the model is the first argument of the function. It can be specified in a number of different ways:

- A name of a variable in the model: `obs_additive("effect")`
- A compartment concentration: `obs_additive(~C["central"])`
- An equation: `obs_additive(~base+slp*time)`

If the definition contains a variable name on the left-hand side (as in `conc~C["central"]`), the variable will appear in the generated model code. This can be useful to make the model code more readable if the prediction is defined as a long equation.

**Observation names:**

The observation name can be specified via the `name=` argument and is automatically derived if the argument is left empty. Adding an observation model with an already existing name will replace the previous definition.

**Error variance:**

The variance of the error components are specified via the `var_add=` and `var_prop=` arguments of the function.

Value

A building block of type 'observation'

See Also

Other observation models: `obs_additive()`, `obs_combined()`

Examples

```r
# additive RUV model for observing the variable WT
m <- model() +
  prm_log_normal("wt") +
  obs_additive(~wt)

# EMAX dose-response model with proportional RUV
m2 <- model() +
```
pk_absorption_fo

Description

This building block declares a first-order absorption component for a pharmacokinetic model.

Usage

pk_absorption_fo(prm_mat = prm_log_normal("mat", median = 0.5, var_log = 0.1))

Arguments

prm_mat Parameter model for the mean absorption time (MAT)

Details

PK components:

PK components can be added to a pk_model and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid.

A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:

```r
pkm <- pk_model() +
  pk_absorption_fo() +
  pk_distribution_1cmp() +
  pk_distribution_2cmp() +
  pk_elimination_linear() +
  obs_additive(conc~C["central"])
```

Parameter models:

All PK component functions allow the specification of the parameter model via their arguments. Arguments that refer to a parameter start with the prefix prm_. The default parameter model can be deduced from the default arguments in the usage section of the help entry. The parameter name, specified via the name= argument of the parameter model building block allows the renaming of the model parameters.

For example, the parameter prm_vc= refers to the central volume of distribution parameter in the one compartment distribution PK component and the default parameter model is a log-normal distribution. The following code block specifies a normal distribution parameter model and names the parameter v:
pk_absorption_fo_lag

pk_distribution_1cmp(
    prm_vc = prm_normal("v", mean = 50, var = 25)
)

Value

A building block of type 'pk_component'

See Also

pk_model() for the creation of PK models

Other absorption components: pk_absorption_fo_lag(), pk_absorption_fo_transit(), pk_absorption_fo_zo(), pk_absorption_zo_lag(), pk_absorption_zo()

-----------------------------------------------

pk_absorption_fo_lag     PK absorption first-order, lag-time

-----------------------------------------------

Description

This building block declares a first-order absorption with lag-time component for a pharmacokinetic model.

Usage

pk_absorption_fo_lag(
    prm_mat = prm_log_normal("mat", median = 0.5, var_log = 0.1),
    prm_mdt = prm_log_normal("mdt", median = 0.5, var_log = 0.1)
)

Arguments

prm_mat          Parameter model for the mean absorption time (MAT)
prm_mdt          Parameter model for the mean delay time (MDT)

Details

PK components:

PK components can be added to a pk_model and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid.

A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:
pk_absorption_fo_transit

```r
pkm <- pk_model() +
  pk_absorption_fo() +
  pk_distribution_1cmp() +
  pk_distribution_2cmp() +
  pk_elimination_linear() +
  obs_additive(conc~C["central"])
pkm
```

**Parameter models:**
All PK component functions allow the specification of the parameter model via their arguments. Arguments that refer to a parameter start with the prefix `prm_`. The default parameter model can be deduced from the default arguments in the usage section of the help entry. The parameter name, specified via the `name=` argument of the parameter model building block allows the renaming of the model parameters.

For example, the parameter `prm_vc` refers to the central volume of distribution parameter in the one compartment distribution PK component and the default parameter model is a log-normal distribution. The following code block specifies a normal distribution parameter model and names the parameter $v$:

```r
pk_distribution_1cmp(
  prm_vc = prm_normal("v", mean = 50, var = 25)
)
```

**Value**
A building block of type 'pk_component'

**See Also**
- `pk_model()` for the creation of PK models
- Other absorption components: `pk_absorption_fo_transit()`, `pk_absorption_fo_zo()`, `pk_absorption_fo()`, `pk_absorption_zo_lag()`, `pk_absorption_zo()`

---

**pk_absorption_fo_transit**

*PK absorption first-order, transit compartment*

**Description**
This building block declares a first-order absorption with transit compartments component for a pharmacokinetic model.

**Usage**

```r
pk_absorption_fo_transit(
  prm_mat = prm_log_normal("mat", median = 0.5, var_log = 0.1),
  transit_compartments = 1L,
  prm_mdt = prm_log_normal("mdt", median = 0.5, var_log = 0.1)
)
```
Arguments

prm_mat Parameter model for the mean absorption time (MAT)

transit_compartments Number of transit compartments

prm_mdt Parameter model for the mean delay time (MDT)

Details

PK components:

PK components can be added to a `pk_model` and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid.

A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:

```r
pkm <- pk_model() +
  pk_absorption_fo() +
  pk_distribution_1cmp() +
  pk_distribution_2cmp() +
  pk_elimination_linear() +
  obs_additive(conc~C["central"])
```

Parameter models:

All PK component functions allow the specification of the parameter model via their arguments. Arguments that refer to a parameter start with the prefix `prm_`. The default parameter model can be deduced from the default arguments in the usage section of the help entry. The parameter name, specified via the name= argument of the parameter model building block allows the renaming of the model parameters.

For example, the parameter `prm_vc=` refers to the central volumen of distribution parameter in the one compartment distribution PK component and the default parameter model is a log-normal distribution. The following code block specifies a normal distribution parameter model and names the parameter `v`:

```r
pk_distribution_1cmp(
  prm_vc = prm_normal("v", mean = 50, var = 25)
)
```

Value

A building block of type 'pk_component'

See Also

`pk_model()` for the creation of PK models

Other absorption components: `pk_absorption_fo_lag()`, `pk_absorption_fo_zo()`, `pk_absorption_fo()`, `pk_absorption_zo_lag()`, `pk_absorption_zo()"
**pk_absorption_fo_zo**

*PK absorption first-order, zero-order delay*

---

**Description**

This building block declares a first-order absorption with zero-order delay component for a pharmacokinetic model.

**Usage**

```r
pk_absorption_fo_zo(
  prm_mat = prm_log_normal("mat", median = 0.5, var_log = 0.1),
  prm_mdt = prm_log_normal("mdt", median = 0.5, var_log = 0.1)
)
```

**Arguments**

- `prm_mat` Parameter model for the mean absorption time (MAT)
- `prm_mdt` Parameter model for the mean delay time (MDT)

**Details**

**PK components:**

PK components can be added to a `pk_model` and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid. A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:

```r
pkm <- pk_model() +
  pk_absorption_fo() +
  pk_distribution_1cmp() +
  pk_distribution_2cmp() +
  pk_elimination_linear() +
  obs_additive(conc~C['central'])
pkm
```

**Parameter models:**

All PK component functions allow the specification of the parameter model via their arguments. Arguments that refer to a parameter start with the prefix `prm_`. The default parameter model can be deduced from the default arguments in the usage section of the help entry. The parameter name, specified via the name= argument of the parameter model building block allows the renaming of the model parameters.

For example, the parameter `prm_vc=` refers to the central volumen of distribution parameter in the one compartment distribution PK component and the default parameter model is a log-normal distribution. The following code block specifies a normal distribution parameter model and names the parameter `v`:

```r
```
pk_absorption_zo

pk_distribution_1cmp(
    prm_vc = prm_normal("v", mean = 50, var = 25)
)

Value
A building block of type 'pk_component'

See Also
pk_model() for the creation of PK models

Other absorption components: pk_absorption_fo_lag(), pk_absorption_fo_transit(), pk_absorption_fo(), pk_absorption_zo_lag(), pk_absorption_zo()

---

pk_absorption_zo PK absorption zero-order

Description
This building block declares a zero-order absorption component for a pharmacokinetic model.

Usage
pk_absorption_zo(prm_mat = prm_log_normal("mat", median = 0.5, var_log = 0.1))

Arguments
prm_mat Parameter model for the mean absorption time (MAT)

Details
PK components:
PK components can be added to a pk_model and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid.
A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:

```r
pkm <- pk_model() +
    pk_absorption_fo() +
    pk_distribution_1cmp() +
    pk_distribution_2cmp() +
    pk_elimination_linear() +
    obs_additive(conc~C["central"])

pkm
```
Parameter models:
All PK component functions allow the specification of the parameter model via their arguments. Arguments that refer to a parameter start with the prefix `prm_`. The default parameter model can be deduced from the default arguments in the usage section of the help entry. The parameter name, specified via the name= argument of the parameter model building block allows the renaming of the model parameters.

For example, the parameter `prm_vc` refers to the central volume of distribution parameter in the one compartment distribution PK component and the default parameter model is a log-normal distribution. The following code block specifies a normal distribution parameter model and names the parameter `v`:

```r
pk_distribution_1cmp(
  prm_vc = prm_normal("v", mean = 50, var = 25)
)
```

Value

A building block of type 'pk_component'

See Also

`pk_model()` for the creation of PK models
Other absorption components: `pk_absorption_fo_lag()`, `pk_absorption_fo_transit()`, `pk_absorption_fo_zo()`, `pk_absorption_fo()`, `pk_absorption_zo_lag()`

---

**pk_absorption_zo_lag**

*PK absorption zero-order, lag-time*

Description

This building block declares a zero-order absorption with lag-time component for a pharmacokinetic model.

Usage

```r
pk_absorption_zo_lag(
  prm_mat = prm_log_normal("mat", median = 0.5, var_log = 0.1),
  prm_mdt = prm_log_normal("mdt", median = 0.5, var_log = 0.1)
)
```

Arguments

- `prm_mat` Parameter model for the mean absorption time (MAT)
- `prm_mdt` Parameter model for the mean delay time (MDT)
Details

**PK components:**

PK components can be added to a `pk_model` and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid.

A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:

```r
pkm <- pk_model() +
    pk_absorption_fo() +
    pk_distribution_1cmp() +
    pk_distribution_2cmp() +
    pk_elimination_linear() +
    obs_additive(conc~C["central"])
pkm
```

**Parameter models:**

All PK component functions allow the specification of the parameter model via their arguments. Arguments that refer to a parameter start with the prefix `prm_`. The default parameter model can be deduced from the default arguments in the usage section of the help entry. The parameter name, specified via the name= argument of the parameter model building block allows the renaming of the model parameters.

For example, the parameter `prm_vc=` refers to the central volumen of distribution parameter in the one compartment distribution PK component and the default parameter model is a log-normal distribution. The following code block specifies a normal distribution parameter model and names the parameter `v`:

```r
pk_distribution_1cmp(
   prm_vc = prm_normal("v", mean = 50, var = 25)
)
```

**Value**

A building block of type `'pk_component'`

**See Also**

- `pk_model()` for the creation of PK models
- Other absorption components: `pk_absorption_fo_lag()`, `pk_absorption_fo_transit()`, `pk_absorption_fo_zo()`, `pk_absorption_fo()`, `pk_absorption_zo()`
Description

This building block declares a one compartment distribution component for a pharmacokinetic model.

Usage

```r
pk_distribution_1cmp(
  prm_vc = prm_log_normal("vc", median = 100, var_log = 0.1)
)
```

Arguments

- **prm_vc** Parameter model for the central volume of distribution

Details

**PK components:**

PK components can be added to a `pk_model` and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid. A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:

```r
pkm <- pk_model() +
  pk_absorption_fo() +
  pk_distribution_1cmp() +
  pk_distribution_2cmp() +
  pk_elimination_linear() +
  obs_additive(conc~C["central"])
```

**Parameter models:**

All PK component functions allow the specification of the parameter model via their arguments. Arguments that refer to a parameter start with the prefix `prm_`. The default parameter model can be deduced from the default arguments in the usage section of the help entry. The parameter name, specified via the name= argument of the parameter model building block allows the renaming of the model parameters.

For example, the parameter `prm_vc` refers to the central volume of distribution parameter in the one compartment distribution PK component and the default parameter model is a log-normal distribution. The following code block specifies a normal distribution parameter model and names the parameter `v`:
pk_distribution_2cmp

pk_distribution_2cmp(
    prm_vc = prm_normal("v", mean = 50, var = 25)
)

Value
A building block of type 'pk_component'

See Also
pk_model() for the creation of PK models

Other distribution components: pk_distribution_2cmp(), pk_distribution_3cmp()

---

pk_distribution_2cmp  PK distribution 2 compartments

Description
This building block declares a two compartment distribution component for a pharmacokinetic model.

Usage
pk_distribution_2cmp(
    prm_vc = prm_log_normal("vc", median = 100, var_log = 0.1),
    prm_vp = prm_log_normal("vp", median = 5, var_log = 0.1),
    prm_q = prm_log_normal("q", median = 50, var_log = 0.1)
)

Arguments
- **prm_vc**: Parameter model for the central volume of distribution
- **prm_vp**: Parameter model for the peripheral volume of distribution
- **prm_q**: Parameter model for the inter-compartmental clearance

Details
PK components:
PK components can be added to a pk_model and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid.
A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:
pk_distribution_3cmp

pkm <- pk_model() +
  pk_absorption_fo() +
  pk_distribution_1cmp() +
  pk_distribution_2cmp() +
  pk_elimination_linear() +
  obs_additive(conc~C["central"]) 

pkm

Parameter models:
All PK component functions allow the specification of the parameter model via their arguments.
Arguments that refer to a parameter start with the prefix prm_. The default parameter model can
be deduced from the default arguments in the usage section of the help entry. The parameter name,
specified via the name= argument of the parameter model building block allows the renaming of
the model parameters.
For example, the parameter prm_vc= refers to the central volume of distribution parameter in
the one compartment distribution PK component and the default parameter model is a log-normal
distribution. The following code block specifies a normal distribution parameter model and names
the parameter \( v \):

pk_distribution_1cmp(
  prm_vc = prm_normal("v", mean = 50, var = 25)
)

Value
A building block of type 'pk_component'

See Also

pk_model() for the creation of PK models

Other distribution components: pk_distribution_1cmp(), pk_distribution_3cmp()

pk_distribution_3cmp  PK distribution 3 compartments

Description
This building block declares a three compartment distribution component for a pharmacokinetic
model.

Usage

pk_distribution_3cmp(
  prm_vc = prm_log_normal("vc", median = 100, var_log = 0.1),
  prm_vp1 = prm_log_normal("vp1", median = 5, var_log = 0.1),
  prm_vp2 = prm_log_normal("vp2", median = 5, var_log = 0.1),
  prm_q1 = prm_log_normal("q1", median = 25, var_log = 0.1),
  prm_q2 = prm_log_normal("q2", median = 25, var_log = 0.1)
)
Arguments

prm_vc  Parameter model for the central volume of distribution
prm_vp1 Parameter model for the volume of the first peripheral compartment
prm_vp2 Parameter model for the volume of the second peripheral compartment
prm_q1  Parameter model for the inter-compartmental clearance between central and first peripheral compartment
prm_q2  Parameter model for the inter-compartmental clearance between central and second peripheral compartment

Details

PK components:
PK components can be added to a pk_model and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid.

A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:

```r
pkm <- pk_model() +
     pk_absorption_fo() +
     pk_distribution_1cmp() +
     pk_distribution_2cmp() +
     pk_elimination_linear() +
     obs_additive(conc~C["central"])
```

Parameter models:
All PK component functions allow the specification of the parameter model via their arguments. Arguments that refer to a parameter start with the prefix `prm`. The default parameter model can be deduced from the default arguments in the usage section of the help entry. The parameter name, specified via the `name=` argument of the parameter model building block allows the renaming of the model parameters.

For example, the parameter `prm_vc` refers to the central volume of distribution parameter in the one compartment distribution PK component and the default parameter model is a log-normal distribution. The following code block specifies a normal distribution parameter model and names the parameter `v`:

```r
pk_distribution_1cmp(
    prm_vc = prm_normal("v", mean = 50, var = 25)
)
```

Value

A building block of type 'pk_component'
Overview

pk_elimination_linear

PK elimination linear

Description

This building block declares a linear elimination component for a pharmacokinetic model.

Usage

pk_elimination_linear(
  prm_cl = prm_log_normal("cl", median = 50, var_log = 0.1)
)

Arguments

- prm_cl Parameter model for the clearance

Details

PK components:

PK components can be added to a pk_model and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid.

A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:

```r
pkm <- pk_model() +
  pk_absorption_fo() +
  pk_distribution_1cmp() +
  pk_distribution_2cmp() +
  pk_elimination_linear() +
  obs_additive(conc~C["central"])
```

Parameter models:

All PK component functions allow the specification of the parameter model via their arguments. Arguments that refer to a parameter start with the prefix prm_. The default parameter model can be deduced from the default arguments in the usage section of the help entry. The parameter name, specified via the name= argument of the parameter model building block allows the renaming of the model parameters.

For example, the parameter prm_vc refers to the central volumen of distribution parameter in the one compartment distribution PK component and the default parameter model is a log-normal
distribution. The following code block specifies a normal distribution parameter model and names the parameter $v$:

```r
pk_distribution_1cmp(
  prm_vc = prm_normal("v", mean = 50, var = 25)
)
```

**Value**

A building block of type 'pk_component'

**See Also**

*pk_model()* for the creation of PK models

Other elimination components: *pk_elimination_linear_nl()*.*pk_elimination_nl()*

---

**pk_elimination_linear_nl**

*PK elimination linear & nonlinear*

**Description**

This building block declares a mixed linear and nonlinear elimination component for a pharmacokinetic model.

**Usage**

```r
pk_elimination_linear_nl(
  prm_cllin = prm_log_normal("cllin", median = 50, var_log = 0.1),
  prm_vmax = prm_log_normal("vmax", median = 10, var_log = 0.1),
  prm_km = prm_log_normal("km", median = 0.5, var_log = 0.1)
)
```

**Arguments**

- **prm_cllin**: Parameter model for the linear clearance
- **prm_vmax**: Parameter model for $V_{max}$ (the maximal elimination rate)
- **prm_km**: Parameter model for $K_{M}$ (the half-maximal concentration)

**Details**

**PK components:**

PK components can be added to a *pk_model* and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid.

A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:
pk_elimination_nl

```r
pkm <- pk_model() +
  pk_absorption_fo() +
  pk_distribution_1cmp() +
  pk_distribution_2cmp() +
  pk_elimination_linear() +
  obs_additive(conc~C["central"])

pkm
```

Parameter models:
All PK component functions allow the specification of the parameter model via their arguments. Arguments that refer to a parameter start with the prefix `prm_`. The default parameter model can be deduced from the default arguments in the usage section of the help entry. The parameter name, specified via the name= argument of the parameter model building block allows the renaming of the model parameters.

For example, the parameter `prm_vc=` refers to the central volume of distribution parameter in the one compartment distribution PK component and the default parameter model is a log-normal distribution. The following code block specifies a normal distribution parameter model and names the parameter `v`:

```r
pk_distribution_1cmp(
  prm_vc = prm_normal("v", mean = 50, var = 25)
)
```

Value
A building block of type 'pk_component'

See Also
- `pk_model()` for the creation of PK models
- Other elimination components: `pk_elimination_linear()`, `pk_elimination_nl()`

Description
This building block declares a nonlinear elimination component for a pharmacokinetic model.

Usage
```
pk_elimination_nl(
  prm_clmm = prm_log_normal("clmm", median = 25, var_log = 0.1),
  prm_km = prm_log_normal("km", median = 0.5, var_log = 0.1),
  prm_vmax = NULL
)
```
Arguments

- `prm_clmm` Parameter model for the clearance
- `prm_km` Parameter model for KM (the half-maximal concentration)
- `prm_vmax` Parameter model for Vmax (the maximal elimination rate)

Details

**PK components:**

PK components can be added to a `pk_model` and exist in three different types: absorption, distribution, and elimination. The absorption component is optional, distribution and elimination are not and need to be added for the PK model to be valid.

A PK model can only have one component of each type and adding a component with an already existing type will replace the previous definition. For example, the distribution component will be a two compartment model in the following snippet:

```r
pkm <- pk_model() +
  pk_absorption_fo() +
  pk_distribution_1cmp() +
  pk_distribution_2cmp() +
  pk_elimination_linear() +
  obs_additive(conc~C["central"])
pkm
```

**Parameter models:**

All PK component functions allow the specification of the parameter model via their arguments. Arguments that refer to a parameter start with the prefix `prm_`. The default parameter model can be deduced from the default arguments in the usage section of the help entry. The parameter name, specified via the `name=` argument of the parameter model building block allows the renaming of the model parameters.

For example, the parameter `prm_vc=` refers to the central volumen of distribution parameter in the one compartment distribution PK component and the default parameter model is a log-normal distribution. The following code block specifies a normal distribution parameter model and names the parameter `v`:

```r
pk_distribution_1cmp(
  prm_vc = prm_normal("v", mean = 50, var = 25)
)
```

Value

A building block of type 'pk_component'

See Also

- `pk_model()` for the creation of PK models
- Other elimination components: `pk_elimination_linear_nl()`, `pk_elimination_linear()`
pk_model

Create a PK model

Description

This function creates the basis for a pharmacokinetic model.

Usage

pk_model()

Details

The function creates the fundament for a pharmacokinetic model to which different building blocks can be added. The following building blocks are relevant for this model type:

- Parameters: prm_log_normal, prm_logit_normal, prm_no_var, prm_normal
- Observations: obs_additive, obs_combined, obs_proportional
- Algebraic relationships: algebraic
- PK components: pk_absorption_fo, pk_absorption_fo_lag, pk_absorption_fo_transit, pk_absorption_fo_zo, pk_absorption_zo, pk_absorption_zo_lag, pk_distribution_1cmp, pk_distribution_2cmp, pk_distribution_3cmp, pk_elimination_linear, pk_elimination_linear_nl, pk_elimination_nl, pk_model
- Input variables: input_variable, dataset

Value

A pk_model

prm_logit_normal

Parameter with logit-normal distribution

Description

This building block declares a parameter model for a parameter that follows the normal distribution on the logit-scale.

Usage

prm_logit_normal(name, mean_logit = 0, var_logit = 1)

Arguments

name Parameter name
mean_logit Mean on the logit scale
var_logit Variance on the logit scale
Details

Parameter models specify type, name, and values for a parameter. The parameter model type is selected through the function name. The parameter name and values are provided as function arguments.

Parameter names:
Every parameter must have a valid name. A parameter name can contain letters, numbers as well as the underscore character. The name needs to start with a letter.
Adding a parameter with an already existing name will replace the definition of the parameter. For example, the parameter “base” will have a log-normal distribution in the following snippet:

```r
m <- model() +
  prm_normal("base") +
  prm_log_normal("base")
```

Parameter values:
The parameter values that a parameter model expects vary by type. For example, `prm_normal()` requires the mean and the variance, whereas for `prm_log_normal()` median and variance on the log scale need to be provided. The argument name should indicate what parameter value is expected.

MU-referencing:
`assemblerr` can include mu-referencing statements for parameter distributions that support it. The functionality can be activated by setting the option `prm.use_mu_referencing` to TRUE as shown in the following snippet:

```r
m <- model() +
  prm_normal("base") +
  prm_log_normal("slp") +
  obs_additive(response~base+slp*time)
render(
  model = m,
  options = assemblerr_options(prm.use_mu_referencing = TRUE)
)
```

Value
A building block of type 'parameter'

See Also
Other parameter models: `prm_log_normal()`, `prm_no_var()`, `prm_normal()`

Examples

# EMAX dose-response model with emax (log-normal) and ed50 (no variability) parameters
m2 <- model() +
  input_variable("dose") +
  prm_log_normal("emax", 10, 0.3) +
prm_log_normal

prm_no_var("ed50", 5) +
obs_proportional(effect-emax*dose/(ed50+dose))

# a log-normal parameter that is directly observed
m <- model() +
   prm_log_normal("wt") +
   obs_additive("wt")

---

**prm_log_normal**

Parameter with log-normal distribution

**Description**

This building block declares a parameter model for a parameter that follows the normal distribution on the log scale.

**Usage**

```r
prm_log_normal(name, median = 1, var_log = 0.1)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Parameter name</td>
</tr>
<tr>
<td>median</td>
<td>Median (on the normal scale)</td>
</tr>
<tr>
<td>var_log</td>
<td>Variance on the log scale</td>
</tr>
</tbody>
</table>

**Details**

Parameter models specify type, name, and values for a parameter. The parameter model type is selected through the function name. The parameter name and values are provided as function arguments.

**Parameter names:**

Every parameter must have a valid name. A parameter name can contain letters, numbers as well as the underscore character. The name needs to start with a letter.

Adding a parameter with an already existing name will replace the definition of the parameter. For example, the parameter “base” will have a log-normal distribution in the following snippet:

```r
m <- model() +
   prm_normal("base") +
   prm_log_normal("base")
```

**Parameter values:**

The parameter values that a parameter model expects vary by type. For example, `prm_normal()` requires the mean and the variance, whereas for `prm_log_normal()` median and variance on the log scale need to be provided. The argument name should indicate what parameter value is expected.
MU-referencing:
assemblerr can include mu-referencing statements for parameter distributions that support it. The functionality can be activated by setting the option `prm.use_mu_referencing` to `TRUE` as shown in the following snippet:

```r
m <- model() +
  prm_normal("base") +
  prm_log_normal("slp") +
  obs_additive(response~base+slp*time)

render(
  model = m,
  options = assemblerr_options(prm.use_mu_referencing = TRUE)
)
```

**Value**

A building block of type 'parameter'

**See Also**

Other parameter models: `prm_logit_normal()`, `prm_no_var()`, `prm_normal()`

**Examples**

```r
# EMAX dose-response model with emax (log-normal) and ed50 (no variability) parameters
m2 <- model() +
  input_variable("dose") +
  prm_log_normal("emax", 10, 0.3) +
  prm_no_var("ed50", 5) +
  obs_proportional(effect~emax*dose/(ed50+dose))

# a log-normal parameter that is directly observed
m <- model() +
  prm_log_normal("wt") +
  obs_additive("wt")
```

---

**prm_normal**

Parameter with normal distribution

**Description**

This building block declares a parameter model for a parameter that follows the normal distribution.

**Usage**

`prm_normal(name, mean = 1, var = 0.1)`
**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Parameter name</td>
</tr>
<tr>
<td>mean</td>
<td>Mean</td>
</tr>
<tr>
<td>var</td>
<td>Variance</td>
</tr>
</tbody>
</table>

**Details**

Parameter models specify type, name, and values for a parameter. The parameter model type is selected through the function name. The parameter name and values are provided as function arguments.

**Parameter names:**

Every parameter must have a valid name. A parameter name can contain letters, numbers as well as the underscore character. The name needs to start with a letter. Adding a parameter with an already existing name will replace the definition of the parameter. For example, the parameter “base” will have a log-normal distribution in the following snippet:

```r
m <- model() +
    prm_normal("base") +
    prm_log_normal("base")
```

**Parameter values:**

The parameter values that a parameter model expects vary by type. For example, `prm_normal()` requires the mean and the variance, whereas for `prm_log_normal()` median and variance on the log scale need to be provided. The argument name should indicate what parameter value is expected.

**MU-referencing:**

`assemblerr` can include mu-referencing statements for parameter distributions that support it. The functionality can be activated by setting the option `prm.use_mu_referencing` to `TRUE` as shown in the following snippet:

```r
m <- model() +
    prm_normal("base") +
    prm_log_normal("slp") +
    obs_additive(response~base+slp*time)

render(
    model = m,
    options = assemblerr_options(prm.use_mu_referencing = TRUE)
)
```

**Value**

A building block of type 'parameter'

**See Also**

Other parameter models: `prm_log_normal()`, `prm_logit_normal()`, `prm_no_var()`
Examples

# EMAX dose-response model with emax (log-normal) and ed50 (no variability) parameters
m2 <- model() +
  input_variable("dose") +
  prm_log_normal("emax", 10, 0.3) +
  prm_no_var("ed50", 5) +
  obs_proportional(effect=emax*dose/(ed50+dose))

# a log-normal parameter that is directly observed
m <- model() +
  prm_log_normal("wt") +
  obs_additive(~wt)

### prm_no_var

**Parameter without variability**

**Description**

This building block declares a parameter model for a parameter that does not vary between subjects.

**Usage**

`prm_no_var(name, value = 1)`

**Arguments**

- `name`: Parameter name
- `value`: Parameter value

**Details**

Parameter models specify type, name, and values for a parameter. The parameter model type is selected through the function name. The parameter name and values are provided as function arguments.

**Parameter names:**

Every parameter must have a valid name. A parameter name can contain letters, numbers as well as the underscore character. The name needs to start with a letter.

Adding a parameter with an already existing name will replace the definition of the parameter. For example, the parameter "base" will have a log-normal distribution in the following snippet:

```r
m <- model() +
  prm_normal("base") +
  prm_log_normal("base")
```
Parameter values:
The parameter values that a parameter model expects vary by type. For example, `prm_normal()` requires the mean and the variance, whereas for `prm_log_normal()` median and variance on the log scale need to be provided. The argument name should indicate what parameter value is expected.

MU-referencing:
`assemblerr` can include mu-referencing statements for parameter distributions that support it. The functionality can be activated by setting the option `prm.use_mu_referencing` to `TRUE` as shown in the following snippet:

```r
m <- model() +
  prm_normal("base") +
  prm_log_normal("slp") +
  obs_additive(response~base+slp*time)

render(  
  model = m,
  options = assemblerr_options(prm.use_mu_referencing = TRUE)
)
```

Value
A building block of type 'parameter'

See Also
Other parameter models: `prm_log_normal()`, `prm_logit_normal()`, `prm_normal()`

Examples
# EMAX dose-response model with emax (log-normal) and ed50 (no variability) parameters
m2 <- model() +
  input_variable("dose") +
  prm_log_normal("emax", 10, 0.3) +
  prm_no_var("ed50", 5) +
  obs_proportional(effect=emax*dose/(ed50+dose))

# a log-normal parameter that is directly observed
m <- model() +
  prm_log_normal("wt") +
  obs_additive(~wt)
Description

This function generates the code for a model object, prints it to the console or writes it to a file.

Usage

render(
  model,
  filename = NULL,
  target_tool = "nonmem",
  tasks = tsk_estimation(),
  options = assemblerr_options()
)

Arguments

model            A model object
filename         Name of the model file to create or NULL
target_tool      Name of the target tool (currently only 'nonmem')
tasks            A task specification
options          List of options for model generation

Details

The generated code will be written to the file specified by filename= or printed to the console if the filename is set to NULL. Only 'nonmem' is currently supported as a target_tool= option. The tasks= argument allows the specification of model tasks and the options= argument customizes the generated code.

Task specification:

Tasks are building blocks that allow to specify what a model should "do". Like other model building blocks, they can be combined using the + operator. For example, the following adds an estimation task and an xpose4 output task to the generated code:

render(m, tasks = tsk_estimation() + tsk_output_xpose4())

The default argument (tasks=tsk_estimation()) adds an FOCE estimation task to the code.

Rendering options:

The options= argument allows to modify the rendering process and, hence, the generated code. Options are provided as a list and the assemblerr_options() function helps to generate list with the proper formatting.

The following code block renders the model m with automatic mu-referencing for the model parameters

render(m, options = assemblerr_options(prm.use_mu_referencing = TRUE))
Value

The model code as a character vector

Examples

```r
m <- model() +
  input_variable("dose") +
  prm_log_normal("emax") +
  prm_log_normal("ed50") +
  obs_additive(eff-emax*dose/(ed50+dose))

# render to console
render(m)

# render to file
## Not run:
setwd(tempdir())
render(m, "run1.mod")
## End(Not run)

# render to console with estimation & output task
render(m, tasks = tsk_estimation() + tsk_output_xpose4())
```

---

tsk_estimation  Task estimation

Description

This function defines an estimation task allowing to specify the estimation algorithm, estimation options, and whether standard errors should be obtained.

Usage

```r
tsk_estimation(algorithm = "foce", se = FALSE, target_options = list())
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>The estimation algorithm to use for the task (&quot;foce&quot;, &quot;foce-inter&quot;, &quot;foce-no-inter&quot;, &quot;fo&quot;, &quot;imp&quot;, &quot;saem&quot;)</td>
</tr>
<tr>
<td>se</td>
<td>Whether to calculate parameter uncertainties</td>
</tr>
<tr>
<td>target_options</td>
<td>List of additional options that should be passed to NONMEM</td>
</tr>
</tbody>
</table>
Details

Tasks:
Tasks are building blocks that allow to specify what a model should “do”. Like other model building blocks, they can be combined using the + operator. However, they should not be added to a model but rather provided via the tasks= argument to the render function, e.g.,

```
render(m, tasks = tsk_estimation() +
      tsk_output_xpose4())
```

Estimation tasks:
Estimation tasks provide details on the parameter estimation process, in terms of estimation algorithm, estimation options and whether standard errors should be obtained.

Algorithm:
The algorithm argument allows to select the estimation algorithm among the following options:

- **foce**: First-order conditional estimation with interaction detection
- **foce-inter**: First-order conditional estimation with interaction
- **foce-no-inter**: First-order conditional estimation without interaction
- **fo**: First-order estimation
- **imp**: Importance sampling
- **saem**: Stochastic approximation expectation maximization

The default algorithm "foce" detects whether the observation model includes an epsilon-eta interaction and includes the INTERACTION option accordingly. The **foce-inter** option forces the use of the INTERACTION argument independent of the residual error model. **foce-no-inter** enforces no interaction.

Each algorithm includes a set of default options that the package authors consider sensible defaults (for example MAXEVAL=999999 for FOCE). These defaults can be overwritten using the target_options= argument which is described below.

Standard errors:
The se= argument allows to request the calculation of parameter standard errors. When standard errors are requested (se=TRUE) it will result in the inclusion of the $COVARIANCE record in the generated control stream.

Target options:
The target_options= argument provides a mechanism to specify additional estimation options for the selected algorithm. The options should be provided as a list, e.g.,

```
   tsk_estimation(algorithm = "foce", target_options = list(mceta=100))
```

The provided options are passed verbatim to the target tool and not checked by assemblerr for correctness.

The target_options= argument

Multiple estimation tasks:
A sequence of estimation tasks can be specified in assemblerr by combining multiple estimations, for example

```
render(m, tasks = tsk_estimation("foce") + tsk_estimation("imp"))
```
will create model code that contains an FOCE as well as an importance sampling estimation step.

Value

A building block of type 'estimation_task'

See Also

Other tasks: tsk_output()

Examples

```r
m <- model() +
  input_variable("dose") +
  prm_log_normal("emax", median = 10, var_log = 0.09) +
  prm_log_normal("ed50", median = 50, var_log = 0.09) +
  algebraic(effect~emax*dose/(ed50 + dose)) +
  obs_proportional(~effect, var_prop = 1)

# add estimation task using importance sampling, covariance step
# and user-defined ISAMPLE option
render(
  model = m,
  tasks = tsk_estimation(
    algorithm = "imp",
    se = TRUE,
    target_options = list(isample=1000)
  )
)
```

---

### tsk_output

#### Task output

**Description**

These functions define output tasks that include the selected variables in the output of the generated model.

**Usage**

```r
tsk_output(filename = "sdtab", variables)

  tsk_output_xpose4()
```

**Arguments**

- `filename` The filename for the output file
- `variables` The model variables that be included in the output
**Details**

**Tasks:**
Tasks are building blocks that allow to specify what a model should “do”. Like other model building blocks, they can be combined using the + operator. However, they should not be added to a model but rather provided via the tasks= argument to the render function, e.g.,

```
render(m, tasks = tsk_estimation() +
       tsk_output_xpose4())
```

**Output tasks:**
For NONMEM, an output task defines the $TABLE records by specifying the filename= as well as the variables= to include.
The variables can be specified by providing a character vector of variable names (e.g., `variables = c("cl", "V")`) or by using a set of variable selection helpers (e.g., `variables = vars_prms()`).
The latter is shorter if many variables are to be selected and allows the specification of tasks independent from the model. The details of the variable selection language can be found on the help pages for `model-variable-selection`.

**xpose4 output task:**
The `tsk_output_xpose4()` function includes $TABLE records that follow the output conventions of the model diagnostic package xpose4. It is a shortcut for the following two output tasks:

```
xpose4_output <- tsk_output("sdtab", variables = any_of(c("id", "time")) | vars_nm_std()) +
    tsk_output("patab", variables = vars_prms() | vars_eta())
```

**Value**
A building block of type 'output_task'

**See Also**
Other tasks: `tsk_estimation()`

**Examples**
```
m <- model() +
   input_variable("dose") +
   prm_log_normal("emax", median = 10, var_log = 0.09) +
   prm_log_normal("ed50", median = 50, var_log = 0.09) +
   algebraic(effect=emax*dose/(ed50 + dose)) +
   obs_proportional(~effect, var_prop = 1)
# output model parameters to file 'prms'
render(m, tasks = tsk_output("prms", variables = vars_prms()))
# output variables required by xpose4
render(m, tasks = tsk_output_xpose4())
```
Index

* absorption components
  pk_absorption_fo, 17
  pk_absorption_fo_lag, 18
  pk_absorption_fo_transit, 19
  pk_absorption_fo_zo, 21
  pk_absorption_zo, 22
  pk_absorption_zo_lag, 23

* distribution components
  pk_distribution_1cmp, 25
  pk_distribution_2cmp, 26
  pk_distribution_3cmp, 27

* elimination components
  pk_elimination_linear, 29
  pk_elimination_linear_nl, 30
  pk_elimination_nl, 31

* observation models
  obs_additive, 11
  obs_combined, 11
  obs_proportional, 11

* parameter models
  prm_log_normal, 35
  prm_logit_normal, 33
  prm_no_var, 38
  prm_normal, 36

* tasks
  tsk_estimation, 41
  tsk_output, 43

algebraic, 4, 11, 33
assemblerr (assemblerr-package), 3
assemblerr-package, 3
assemblerr_options, 5
check, 6
cmp (compartment), 7
compartment, 7, 11
dataset, 11, 33
dataset (input_variable), 10
flow, 7, 8, 11
input_variable, 10, 11, 33
model, 10
model-variable-selection, 11, 44
obs_additive, 11, 13, 15, 16, 33
obs_combined, 11, 14, 16, 33
obs_proportional, 11, 14, 15, 33
pk_absorption_fo, 17, 19, 20, 22–24, 33
pk_absorption_fo_lag, 18, 18, 19, 20, 22–24, 33
pk_absorption_fo_transit, 18, 19, 19, 22–24, 33
pk_absorption_fo_zo, 18–20, 21, 23, 24, 33
pk_absorption_zo, 18–20, 22, 22, 24, 33
pk_absorption_zo_lag, 18–20, 22, 23, 23, 33
pk_distribution_1cmp, 25, 27, 29, 33
pk_distribution_2cmp, 26, 26, 29, 33
pk_distribution_3cmp, 26, 27, 27, 33
pk_elimination_linear, 29, 31–33
pk_elimination_linear_nl, 30, 30, 32, 33
pk_elimination_nl, 30, 31, 31, 33
pk_model, 17, 18, 20–22, 24–26, 28–30, 32, 33, 33
pk_model(), 11, 18–20, 22–24, 26, 27, 29–32
prm_log_normal, 11, 33, 34, 35, 37, 39
prm_logit_normal, 11, 33, 33, 36, 37, 39
prm_no_var, 11, 33, 34, 36, 37, 38
prm_normal, 11, 33, 34, 36, 36, 39
render, 40
tidyselect::language, 11
tsk_estimation, 41, 44
tsk_output, 43, 43
tsk_output_xpose4 (tsk_output), 43
vars_data (model-variable-selection), 11
vars_eta (model-variable-selection), 11
vars_matches
  (model-variable-selection), 11
vars_nm_std (model-variable-selection), 11
vars_prms (model-variable-selection), 11
vars_starts_with
  (model-variable-selection), 11