

Package ‘glmmrBase’

December 8, 2022

Type Package

Title Specification of Generalised Linear Mixed Models

Version 0.2.3

Date 2022-12-08

Description Specification of generalised linear mixed models using the 'R6' object-orientated class system. The package provides classes 'Covariance', 'MeanFunction' and 'Model', which allow for flexible specification of generalised linear mixed models, as well as functionality to produce relevant matrices, values, and analyses. See <<https://github.com/samuel-watson/glmmrBase/blob/master/README.md>> for a detailed manual.

License GPL (>= 2)

Imports methods, digest, Rcpp (>= 1.0.7), R6

LinkingTo Rcpp (>= 0.12.0), RcppEigen

RoxygenNote 7.2.1

NeedsCompilation yes

Author Sam Watson [aut, cre],
Yi Pan [aut]

URL <https://github.com/samuel-watson/glmmrBase>

BugReports <https://github.com/samuel-watson/glmmrBase/issues>

Suggests testthat

Biarch true

Depends R (>= 3.4.0), Matrix (>= 1.3-1)

SystemRequirements GNU make

Encoding UTF-8

Maintainer Sam Watson <S.I.Watson@bham.ac.uk>

Repository CRAN

Date/Publication 2022-12-08 15:00:02 UTC

R topics documented:

glmmrBase-package	2
Beta	4
Covariance	4
cross_df	9
cycles	10
dfactor	10
dfexp	11
didentity	11
genCholD	11
genD	12
gen_dhdmu	13
match_rows	13
MeanFunction	14
Model	20
nelder	30
nest_df	32
parallel_crt	32
progress_bar	34
sample_re	35
staircase_crt	36
stepped_wedge	38
Index	40

glmmrBase-package	<i>Specification of Generalised Linear Mixed Models</i>
-------------------	---

Description

Specification of generalised linear mixed models using the 'R6' object-orientated class system. The package provides classes 'Covariance', 'MeanFunction' and 'Model', which allow for flexible specification of generalised linear mixed models, as well as functionality to produce relevant matrices, values, and analyses. See <<https://github.com/samuel-watson/glmmrBase/blob/master/README.md>> for a detailed manual.

Details

The DESCRIPTION file:

```

Package:      glmmrBase
Type:         Package
Title:        Specification of Generalised Linear Mixed Models
Version:      0.2.3
Date:         2022-12-08
Authors@R:    c(person("Sam", "Watson", email = "S.I.Watson@bham.ac.uk", role = c("aut", "cre")), person("Yi", "
Description:  Specification of generalised linear mixed models using the 'R6' object-orientated class system. The p

```

License: GPL (≥ 2)
 Imports: methods, digest, Rcpp ($\geq 1.0.7$), R6
 LinkingTo: Rcpp ($\geq 0.12.0$), RcppEigen
 RoxygenNote: 7.2.1
 NeedsCompilation: yes
 Author: Sam Watson [aut, cre], Yi Pan [aut]
 URL: <https://github.com/samuel-watson/glmmrBase>
 BugReports: <https://github.com/samuel-watson/glmmrBase/issues>
 Suggests: testthat
 Biarch: true
 Depends: R ($\geq 3.4.0$), Matrix ($\geq 1.3-1$)
 SystemRequirements: GNU make
 Encoding: UTF-8
 Maintainer: Sam Watson <S.I.Watson@bham.ac.uk>

Index of help topics:

Beta	Beta distribution declaration
Covariance	R6 Class representing a covariance function and data
MeanFunction	R6 Class representing a mean function function and data
Model	A GLMM Model
cross_df	Generate crossed block structure
cycles	Generates all the orderings of a
dfactor	Factor function
dfexp	Exponential function
didentity	Identity function
genCholD	Generates the Cholesky decomposition covariance matrix of the random effects
genD	Generates the covariance matrix of the random effects
gen_dhdmu	Generates the derivative of the link function with respect to the mean. Used internally in the Model function class.
glmmrBase-package	Specification of Generalised Linear Mixed Models
match_rows	Generate matrix mapping between data frames
nelder	Generates a block experimental structure using Nelder's formula
nest_df	Generate nested block structure
parallel_crt	Generate a parallel cluster design
progress_bar	Generates a progress bar
sample_re	Generates a sample of random effects
staircase_crt	Generate a staircase/diagonal trial design
stepped_wedge	Generate a stepped-wedge design

<https://github.com/samuel-watson/glmmrBase/blob/master/README.md>

Author(s)

Sam Watson [aut, cre], Yi Pan [aut]

Maintainer: NA

Beta

Beta distribution declaration

Description

Skeleton list to declare a Beta distribution in a 'Model' object

Usage

```
Beta(link = "logit")
```

Arguments

link Name of link function. Only accepts 'logit' currently.

Value

A list with two elements naming the family and link function

Covariance

R6 Class representing a covariance function and data

Description

R6 Class representing a covariance function and data

R6 Class representing a covariance function and data

Details

For the generalised linear mixed model

$$\begin{aligned}
 Y &\sim F(\mu, \sigma) \\
 \mu &= h^{-1}(X\beta + Z\gamma) \\
 \gamma &\sim MVN(0, D)
 \end{aligned}$$

where h is the link function, this class defines Z and D . The covariance is defined by a covariance function, data, and parameters. A new instance can be generated with `$new()`. The class will generate the relevant matrices Z and D automatically. See [glmmrBase](#) for a detailed guide on model specification.

****Initialisation**** A covariance function is specified as an additive formula made up of components with structure $(1|f(j))$. The left side of the vertical bar specifies the covariates in the model that have a random effects structure. The right side of the vertical bar specify the covariance function 'f' for that term using variable named in the data 'j'. Covariance functions on the right side of the vertical bar are multiplied together, i.e. $(1|f(j)*g(t))$.

There are several common functions included for a named variable in data x. A non-exhaustive list (see `glmmrBase` for a full list): `gr(x)`: Indicator function (1 parameter) * `fexp(x)`: Exponential function (2 parameters) * `ar1(x)`: AR1 function (1 parameter) * `sqexp(x)`: Squared exponential (1 parameter) * `matern(x)`: Matern function (2 parameters) * `bessel(x)`: Modified Bessel function of the 2nd kind (1 parameter)

Parameters are provided to the covariance function as a vector. The parameters in the vector for each function should be provided in the order the covariance functions are written are written. For example, * Formula: `'~(1|gr(j))+(1|gr(j*t))'`; parameters: `'c(0.25,0.1)'` * Formula: `'~(1|gr(j)*fexp(t))'`; parameters: `'c(0.25,1,0.5)'` Note that it is also possible to specify a group membership with two variable alternatively as `'(1|gr(j)*gr(t))'`, for example, but this will require two parameters to be specified, so it is recommended against.

If not all of 'formula', 'data', and 'parameters' are not specified then the linked matrices are not calculated. These options can be later specified, or updated via a `Model` object. If these arguments are updated or changed then call `'self$check()'` to update linked matrices. Updating of parameters is automatic if using the `'update_parameters()'` member function.

Using `'update_parameters()'` is the preferred way of updating the parameters of the mean or covariance objects as opposed to direct assignment, e.g. `'self$parameters <- c(...)'`. The function calls check functions to automatically update linked matrices with the new parameters. If using direct assignment, call `'self$check()'` afterwards.

Public fields

`data` Data frame with data required to build covariance

`formula` Covariance function formula.

`parameters` Model parameters specified in order of the functions in the formula.

`eff_range` The effective range of covariance functions, specified in order of the functions in the formula. Only the functions with compact support require effective range parameters.

`Z` Design matrix

`D` Covariance matrix of the random effects

Methods

Public methods:

- `Covariance$n()`
- `Covariance$new()`
- `Covariance$check()`
- `Covariance$update_parameters()`
- `Covariance$print()`
- `Covariance$subset()`
- `Covariance$sampleD()`

- `Covariance$get_D_data()`
- `Covariance$get_chol_D()`
- `Covariance$clone()`

Method `n()`: Return the size of the design

Usage:

```
Covariance$n()
```

Returns: Scalar

Method `new()`: Create a new Covariance object

Usage:

```
Covariance$new(
  formula,
  data = NULL,
  parameters = NULL,
  eff_range = NULL,
  verbose = FALSE
)
```

Arguments:

`formula` Formula describing the covariance function. See Details

`data` (Optional) Data frame with data required for constructing the covariance.

`parameters` (Optional) Vector with parameter values for the functions in the model formula. See Details.

`eff_range` (Optional) Vector with the effective range parameter for covariance functions that require it, i.e. those with compact support.

`verbose` Logical whether to provide detailed output.

Returns: A Covariance object

Examples:

```
df <- nelder(~(c1(5)*t(5)) > ind(5))
cov <- Covariance$new(formula = ~(1|gr(c1)*ar1(t)),
  parameters = c(0.25,0.7),
  data= df)
```

Method `check()`: Check if anything has changed and update matrices if so.

Usage:

```
Covariance$check(verbose = TRUE)
```

Arguments:

`verbose` Logical whether to report if any changes detected.

Returns: NULL

Examples:

```
df <- nelder(~(c1(5)*t(5)) > ind(5))
cov <- Covariance$new(formula = ~(1|gr(c1)*ar1(t)),
  parameters = c(0.15,0.8),
  data= df)
cov$parameters <- c(0.25,0.1)
cov$check(verbose=FALSE)
```


Method `get_D_data()`: Returns the list specifying the covariance matrix D

Usage:

```
Covariance$get_D_data()
```

Returns: A list

Method `get_chol_D()`: Returns the Cholesky decomposition of the covariance matrix D

Usage:

```
Covariance$get_chol_D(parameters = NULL)
```

Arguments:

`parameters` (Optional) Vector of parameters, if specified then the Cholesky factor is calculated with these parameter values rather than the ones stored in the object.

Returns: A list of matrices

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Covariance$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## -----
## Method `Covariance$new`
## -----

df <- nelder(~(cl(5)*t(5)) > ind(5))
cov <- Covariance$new(formula = ~(1|gr(cl)*ar1(t)),
                      parameters = c(0.25,0.7),
                      data= df)

## -----
## Method `Covariance$check`
## -----

df <- nelder(~(cl(5)*t(5)) > ind(5))
cov <- Covariance$new(formula = ~(1|gr(cl)*ar1(t)),
                      parameters = c(0.15,0.8),
                      data= df)
cov$parameters <- c(0.25,0.1)
cov$check(verbose=FALSE)

## -----
## Method `Covariance$print`
## -----

df <- nelder(~(cl(5)*t(5)) > ind(5))
Covariance$new(formula = ~(1|gr(cl)*ar1(t)),
```



```

                                parameters = c(0.05,0.8),
                                data= df)

## -----
## Method `Covariance$subset`
## -----

df <- nelder(~(cl(10)*t(5)) > ind(10))
cov <- Covariance$new(formula = ~(1|gr(cl)*ar1(t)),
                      parameters = c(0.05,0.8),
                      data= df)
cov$subset(1:100)

## -----
## Method `Covariance$sampleD`
## -----

df <- nelder(~(cl(10)*t(5)) > ind(10))
cov <- Covariance$new(formula = ~(1|gr(cl)*ar1(t)),
                      parameters = c(0.05,0.8),
                      data= df)
cov$sampleD(c(0.01,0.1))

```

cross_df	<i>Generate crossed block structure</i>
----------	---

Description

Generate a data frame with crossed rows from two other data frames

Usage

```
cross_df(df1, df2)
```

Arguments

df1	data frame
df2	data frame

Details

For two data frames ‘df1’ and ‘df2’, the function will return another data frame that crosses them, which has rows with every unique combination of the input data frames

Value

data frame

Examples

```
cross_df(data.frame(t=1:4), data.frame(cl=1:3))
```

cycles	<i>Generates all the orderings of a</i>
--------	---

Description

Given input a, returns a $\text{length}(a)^2$ vector by cycling through the values of a

Usage

```
cycles(a)
```

Arguments

a vector

Value

vector

dfactor	<i>Factor function</i>
---------	------------------------

Description

Factor function

Usage

```
dfactor(x)
```

Arguments

x List with named elements 'data', a vector of covariate values, and 'pars', a vector of two parameters

dfexp	<i>Exponential function</i>
-------	-----------------------------

Description

Exponential function

Usage

dfexp(x)

Arguments

x	List with named elements 'data', a vector of covariate values, and 'pars', a vector of two parameters
---	---

didentity	<i>Identity function</i>
-----------	--------------------------

Description

Identity function

Usage

didentity(x)

Arguments

x	List with named vector data
---	-----------------------------

genCholD	<i>Generates the Cholesky decomposition covariance matrix of the random effects</i>
----------	---

Description

Generates the Cholesky Decomposition of the covariance matrix of the random effects. Used internally in the Covariance class.

Usage

genCholD(cov, data, eff_range, gamma)

Arguments

cov	An integer matrix with columns of block identifier, dimension of block, function definition, number of variables in the argument to the function, and index of the parameters, respectively. Rows are specific functions of each block.
data	Vector of data. Created by flattening the matrices in column-major order of the data used in each block.
eff_range	Vector of values with the effective range parameters of the covariance functions, where required.
gamma	Vector of parameters used to generate the matrix D.

Value

A lower triangular matrix

genD	<i>Generates the covariance matrix of the random effects</i>
------	--

Description

Generates the covariance matrix of the random effects from a sparse representation. Used internally in the Covariance class.

Usage

```
genD(cov, data, eff_range, gamma)
```

Arguments

cov	An integer matrix with columns of block identifier, dimension of block, function definition, number of variables in the argument to the function, and index of the parameters, respectively. Rows are specific functions of each block.
data	Vector of data. Created by flattening the matrices in column-major order of the data used in each block.
eff_range	Vector of values with the effective range parameters of the covariance functions, where required.
gamma	Vector of parameters used to generate the matrix D.

Value

A symmetric positive definite covariance matrix

gen_dhdmu	<i>Generates the derivative of the link function with respect to the mean. Used internally in the Model function class.</i>
-----------	---

Description

Generates the derivative of the link function with respect to the mean. Used internally in the Model function class.

Usage

```
gen_dhdmu(xb, family, link)
```

Arguments

xb	Vector with mean function value evaluated at fitted model parameters
family	String declaring model family
link	String declaring model link function

Value

Vector of derivative values

match_rows	<i>Generate matrix mapping between data frames</i>
------------	--

Description

For a data frames 'x' and 'target', the function will return a matrix mapping the rows of 'x' to those of 'target'.

Usage

```
match_rows(x, target, by)
```

Arguments

x	data.frame
target	data.frame to map to
by	vector of strings naming columns in 'x' and 'target'

Details

'x' is a data frame with n rows and 'target' a data frame with m rows. This function will return a n times m matrix that maps the rows of 'x' to those of 'target' based on the values in the columns specified by the argument 'by'

Value

A matrix with `nrow(x)` rows and `nrow(target)` columns

Examples

```
df <- nelder(~(cl(10)*t(5)) > ind(10))
df_unique <- df[!duplicated(df[,c('cl', 't')]),]
match_rows(df, df_unique, c('cl', 't'))
```

MeanFunction

R6 Class representing a mean function function and data

Description

R6 Class representing a mean function function and data

R6 Class representing a mean function function and data

Details

For the generalised linear mixed model

$$\begin{aligned}
 Y &\sim F(\mu, \sigma) \\
 \mu &= h^{-1}(X\beta + Z\gamma) \\
 \gamma &\sim MVN(0, D)
 \end{aligned}$$

this class defines the family F , link function h , and fixed effects design matrix X . The mean function is defined by a model formula, data, and parameters. A new instance can be generated with `$new()`. The class will generate the relevant matrix X automatically. See [glmmrBase](#) for a detailed guide on model specification.

Specification of the mean function follows standard model formulae in R. For example for a stepped-wedge cluster trial model, a typical mean model is $E(y_{ijt}|\delta) = \beta_0 + \tau_t + \beta_1 d_{jt} + z_{ijt}\delta$ where τ_t are fixed effects for each time period. The formula specification for this would be `'~ factor(t) + int'` where `'int'` is the name of the variable indicating the treatment.

One can also include non-linear functions of variables in the mean function. These are handled in the analyses by first-order approximation.

If not all of `'formula'`, `'data'`, `'family'`, and `'parameters'` are not specified then the linked matrices are not calculated. These options can be later specified, or updated via a [Model](#) object. If these arguments are updated or changed then call `'self$check()'` to update linked matrices. Updating of parameters is automatic if using the `'update_parameters()'` member function.

Using `'update_parameters()'` is the preferred way of updating the parameters of the mean or covariance objects as opposed to direct assignment, e.g. `'self$parameters <- c(...)'`. The function calls check functions to automatically update linked matrices with the new parameters. If using direct assignment, call `'self$check()'` afterwards.

Public fields

- formula model formula for the fixed effects
- data Data frame with data required to build X
- family One of the family function used in R's glm functions. See [family](#) for details
- parameters A vector of parameter values for β used for simulating data and calculating covariance matrix of observations for non-linear models.
- randomise A function that generates a new set of values representing the treatment allocation in an experimental study
- treat_var A string naming the column in data that represents the treatment variable in data. Used to identify where to replace allocation when randomiser is used.
- X the fixed effects design matrix

Methods**Public methods:**

- [MeanFunction\\$n\(\)](#)
- [MeanFunction\\$check\(\)](#)
- [MeanFunction\\$new\(\)](#)
- [MeanFunction\\$print\(\)](#)
- [MeanFunction\\$update_parameters\(\)](#)
- [MeanFunction\\$colnames\(\)](#)
- [MeanFunction\\$subset_rows\(\)](#)
- [MeanFunction\\$subset_cols\(\)](#)
- [MeanFunction\\$clone\(\)](#)

Method `n()`: Returns the number of observations

Usage:

```
MeanFunction$n()
```

Arguments:

... ignored

Returns: The number of observations in the model

Examples:

```
df <- nelder(~(cl(4)*t(5)) > ind(5))
df$int <- 0
df[df$cl <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
                        parameters = c(-1,1),
                        family = stats::binomial()
                        )
mf1$n()
```

Method `check()`: Checks if any changes have been made and updates
Checks if any changes have been made and updates, usually called automatically.

Usage:

```
MeanFunction$check(verbose = TRUE)
```

Arguments:

verbose Logical whether to report if any changes detected.

Returns: NULL

Examples:

```
df <- nelder(~(c1(4)*t(5)) > ind(5))
df$int <- 0
df[df$c1 <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
                        parameters = c(-1,1),
                        family = stats::binomial()
                        )
mf1$parameters <- c(0,0)
mf1$check()
```

Method new(): Create a new MeanFunction object

Usage:

```
MeanFunction$new(
  formula,
  data = NULL,
  family = NULL,
  parameters = NULL,
  verbose = FALSE
)
```

Arguments:

formula A [formula](#) object that describes the mean function, see [Details](#)

data (Optional) A data frame containing the covariates in the model, named in the model formula

family (Optional) A family object expressing the distribution and link function of the model, see [family](#)

parameters (Optional) A vector with the values of the parameters β to use in data simulation and covariance calculations. If the parameters are not specified then they are initialised to 0.

verbose Logical indicating whether to report detailed output

Returns: A MeanFunction object

Examples:

```
df <- nelder(~(c1(4)*t(5)) > ind(5))
df$int <- 0
df[df$c1 <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
                        parameters = c(-1,1),
                        family = stats::binomial()
                        )
```


Method print(): Prints details about the object

Usage:

```
MeanFunction$print()
```

Arguments:

... ignored

Method update_parameters(): Updates the model parameters

Usage:

```
MeanFunction$update_parameters(parameters, verbose = FALSE)
```

Arguments:

parameters A vector of parameters for the mean function.

verbose Logical indicating whether to provide more detailed feedback

Method colnames(): Returns or replaces the column names of the data in the object

Usage:

```
MeanFunction$colnames(names = NULL)
```

Arguments:

names If NULL then the function prints the column names, if a vector of names, then it attempts to replace the current column names of the data

Examples:

```
df <- nelder(~(cl(4)*t(5)) > ind(5))
df$int <- 0
df[df$cl <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
                        parameters = c(-1,1),
                        family = stats::binomial()
                        )
mf1$colnames(c("cluster","time","individual","treatment"))
mf1$colnames()
```

Method subset_rows(): Keeps a subset of the data and removes the rest

All indices not in the provided vector of row numbers will be removed from both the data and fixed effects design matrix X.

Usage:

```
MeanFunction$subset_rows(index)
```

Arguments:

index Rows of the data to keep

Returns: NULL

Examples:

```
df <- nelder(~(cl(4)*t(5)) > ind(5))
df$int <- 0
df[df$cl <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
                        parameters = c(-1,1),
                        family = stats::binomial()
                        )
mf1$subset_rows(1:20)
```

Method subset_cols(): Keeps a subset of the columns of X

All indices not in the provided vector of column numbers will be removed from the fixed effects design matrix X.

Usage:

```
MeanFunction$subset_cols(index)
```

Arguments:

index Columns of X to keep

Returns: NULL

Examples:

```
df <- nelder(~(cl(4)*t(5)) > ind(5))
df$int <- 0
df[df$cl <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
                        parameters = c(-1,1),
                        family = stats::binomial()
                        )
mf1$subset_cols(1:2)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
MeanFunction$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## -----
## Method `MeanFunction$new`
## -----

df <- nelder(~(cl(4)*t(5)) > ind(5))
df$int <- 0
df[df$cl <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
```

```

                                parameters = c(-1,1),
                                family = stats::binomial()
                                )
mf1$n()

## -----
## Method `MeanFunction$check`
## -----

df <- nelder(~(cl(4)*t(5)) > ind(5))
df$int <- 0
df[df$cl <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
                        parameters = c(-1,1),
                        family = stats::binomial()
                        )
mf1$parameters <- c(0,0)
mf1$check()

## -----
## Method `MeanFunction$new`
## -----

df <- nelder(~(cl(4)*t(5)) > ind(5))
df$int <- 0
df[df$cl <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
                        parameters = c(-1,1),
                        family = stats::binomial()
                        )

## -----
## Method `MeanFunction$colnames`
## -----

df <- nelder(~(cl(4)*t(5)) > ind(5))
df$int <- 0
df[df$cl <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
                        parameters = c(-1,1),
                        family = stats::binomial()
                        )
mf1$colnames(c("cluster", "time", "individual", "treatment"))
mf1$colnames()

## -----
## Method `MeanFunction$subset_rows`
## -----

df <- nelder(~(cl(4)*t(5)) > ind(5))

```

```

df$int <- 0
df[df$c1 <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
                        parameters = c(-1,1),
                        family = stats::binomial()
                        )
mf1$subset_rows(1:20)

## -----
## Method `MeanFunction$subset_cols`
## -----

df <- nelder(~(c1(4)*t(5)) > ind(5))
df$int <- 0
df[df$c1 <= 5, 'int'] <- 1
mf1 <- MeanFunction$new(formula = ~ int ,
                        data=df,
                        parameters = c(-1,1),
                        family = stats::binomial()
                        )
mf1$subset_cols(1:2)

```

Model

A GLMM Model

Description

A GLMM Model

A GLMM Model

Details

An R6 class representing a GLMM model

For the generalised linear mixed model

$$\begin{aligned}
 Y &\sim F(\mu, \sigma) \\
 \mu &= h^{-1}(X\beta + Z\gamma) \\
 \gamma &\sim MVN(0, D)
 \end{aligned}$$

where h is the link function. A Model is comprised of a [MeanFunction](#) object, which defines the family F , link function h , and fixed effects design matrix X , and a [Covariance](#) object, which defines Z and D . The class provides methods for analysis and simulation with these models.

This class provides methods for generating the matrices described above and data simulation, and serves as a base for extended functionality in related packages.

The class by default calculates the covariance matrix of the observations as:

$$\Sigma = W^{-1} + ZDZ^T$$

where W is a diagonal matrix with the WLS iterated weights for each observation equal to, for individual i , $\phi_{a_i} v(\mu_i) [h'(\mu_i)]^2$ (see Table 2.1 in McCullagh and Nelder (1989) <ISBN:9780412317606>). For very large designs, this can be disabled as the memory requirements can be prohibitive.

See [glmmrBase](#) for a detailed guide on model specification.

Calls the respective print methods of the linked covariance and mean function objects.

The matrices X and Z both have n rows, where n is the number of observations in the model/design.

****Number of clusters**** Returns a data frame describing the number of independent clusters or groups at each level in the design. For example, if there were cluster-periods nested in clusters, then the top level would be clusters, and the second level would be cluster periods.

Using `update_parameters()` is the preferred way of updating the parameters of the mean or covariance objects as opposed to direct assignment, e.g. `self$covariance$parameters <- c(...)`. The function calls check functions to automatically update linked matrices with the new parameters. If using direct assignment, call `self$check()` afterwards.

Public fields

`covariance` A [Covariance](#) object defining the random effects covariance.

`mean_function` A [MeanFunction](#) object, defining the mean function for the model, including the data and covariate design matrix X .

`exp_condition` A vector indicating the unique experimental conditions for each observation, see [Details](#).

`Sigma` The overall covariance matrix for the observations. Calculated and updated automatically as $W^{-1} + ZDZ^T$ where W is an $n \times n$ diagonal matrix with elements on the diagonal equal to the GLM iterated weights. See [Details](#).

`var_par` Scale parameter required for some distributions (Gaussian, Gamma, Beta).

Methods

Public methods:

- `Model$fitted()`
- `Model$new()`
- `Model$print()`
- `Model$n()`
- `Model$n_cluster()`
- `Model$subset_rows()`
- `Model$subset_cols()`
- `Model$sim_data()`
- `Model$check()`
- `Model$update_parameters()`
- `Model$information_matrix()`
- `Model$power()`

- [Model\\$clone\(\)](#)

Method `fitted()`: Return predicted values based on the currently stored parameter values in `'mean_function'`

Usage:

```
Model$fitted(type = "link")
```

Arguments:

type One of either "link" for values on the scale of the link function, or "response" for values on the scale of the response

Returns: A [Matrix](#) class object containing the predicted values

Method `new()`: Create a new Model object

Usage:

```
Model$new(
  covariance,
  mean,
  data = NULL,
  family = NULL,
  var_par = NULL,
  verbose = TRUE,
  skip.sigma = FALSE
)
```

Arguments:

`covariance` Either a [Covariance](#) object, or an equivalent list of arguments that can be passed to `'Covariance'` to create a new object. At a minimum the list must specify a formula. If parameters are not included then they are initialised to 0.5.

`mean` Either a [MeanFunction](#) object, or an equivalent list of arguments that can be passed to `'MeanFunction'` to create a new object. At a minimum the list must specify a formula. If parameters are not included then they are initialised to 0.

`data` A data frame with the data required for the mean function and covariance objects. This argument can be ignored if data are provided to the covariance or mean arguments either via `'Covariance'` and `'MeanFunction'` object, or as a member of the list of arguments to both `'covariance'` and `'mean'`.

`family` A family object expressing the distribution and link function of the model, see [family](#). This argument is optional if the family is provided either via a `'MeanFunction'` or `'MeanFunction'` objects, or as members of the list of arguments to `'mean'`. Current accepts [binomial](#), [gaussian](#), [Gamma](#), [poisson](#), and [Beta](#).

`var_par` Scale parameter required for some distributions, including Gaussian. Default is NULL.

`verbose` Logical indicating whether to provide detailed output

`skip.sigma` Logical indicating whether to skip the creating of the covariance matrix Sigma. For very large designs with thousands of observations or more, the covariance matrix will be too big to fit in memory, so this option will prevent sigma being created.

Returns: A new Model class object

Examples:

```

#create a data frame describing a cross-sectional parallel cluster
#randomised trial
df <- nelder(~(cl(10)*t(5)) > ind(10))
df$int <- 0
df[df$cl > 5, 'int'] <- 1

mf1 <- MeanFunction$new(
  formula = ~ factor(t) + int - 1,
  data=df
)
cov1 <- Covariance$new(
  data = df,
  formula = ~ (1|gr(cl)) + (1|gr(cl*t))
)
des <- Model$new(
  covariance = cov1,
  mean = mf1,
  family = stats::gaussian(),
  var_par = 1
)

#alternatively we can pass the data directly to Model
#here we will specify a cohort study and provide parameter values
df <- nelder(~ind(20) * t(6))
df$int <- 0
df[df$t > 3, 'int'] <- 1

des <- Model$new(
  covariance = list(
    formula = ~ (1|gr(ind)),
    parameters = c(0.25)),
  mean = list(
    formula = ~ int,
    parameters = c(1,0.5)),
  data = df,
  family = stats::poisson())

#an example of a spatial grid with two time points
df <- nelder(~ (x(10)*y(10))*t(2))
spt_design <- Model$new(covariance = list( formula = ~(1|fexp(x,y)*ar1(t))),
  mean = list(formula = ~ 1),
  data = df,
  family = stats::gaussian())

```

Method print(): Print method for 'Model' class

Usage:

```
Model$print()
```

Arguments:

... ignored

Method `n()`: Returns the number of observations in the model

Usage:

```
Model$n(...)
```

Arguments:

... ignored

Method `n_cluster()`: Returns the number of clusters at each level

Usage:

```
Model$n_cluster()
```

Arguments:

... ignored

Returns: A data frame with the level, number of clusters, and variables describing each level.

Examples:

```
df <- nelder(~(cl(10)*t(5)) > ind(10))
df$int <- 0
df[df$cl > 5, 'int'] <- 1
```

```
des <- Model$new(
  covariance = list(formula = ~ (1|gr(cl)) + (1|gr(cl*t))),
  mean = list(formula = ~ factor(t) + int - 1),
  data = df,
  family = stats::gaussian(),
  var_par = 1
)
des$n_cluster() ## returns two levels of 10 and 50
```

Method `subset_rows()`: Subsets the design keeping specified observations only

Given a vector of row indices, the corresponding rows will be kept and the other rows will be removed from the mean function and covariance

Usage:

```
Model$subset_rows(index)
```

Arguments:

`index` Integer or vector integers listing the rows to keep

Returns: The function updates the object and nothing is returned

Examples:

```
#generate a stepped wedge design and remove the first sequence
des <- stepped_wedge(8,10,icc=0.05)
ids_to_keep <- which(des$mean_function$data$J!=1)
des$subset_rows(ids_to_keep)
```

Method `subset_cols()`: Subsets the columns of the design

Removes the specified columns from the linked mean function object's X matrix.

Usage:

```
Model$subset_cols(index)
```

Arguments:

index Integer or vector of integers specifying the indexes of the columns to keep

Returns: The function updates the object and nothing is returned

Examples:

```
#generate a stepped wedge design and remove first and last time periods
des <- stepped_wedge(8,10,icc=0.05)
des$subset_cols(c(2:8,10))
```

Method `sim_data()`: Generates a realisation of the design

Generates a single vector of outcome data based upon the specified GLMM design

Usage:

```
Model$sim_data(type = "y")
```

Arguments:

type Either 'y' to return just the outcome data, or 'data' to return a data frame with the simulated outcome data alongside the model data

Returns: Either a vector or a data frame

Examples:

```
df <- nelder(~(c1(10)*t(5)) > ind(10))
df$int <- 0
df[df$c1 > 5, 'int'] <- 1
des <- Model$new(
  covariance = list(
    formula = ~ (1|gr(c1))*ar1(t),
    parameters = c(0.25,0.8)),
  mean = list(
    formula = ~ factor(t) + int - 1,
    parameters = c(rep(0,5),0.6)),
  data = df,
  family = stats::binomial()
)
ysim <- des$sim_data()
```

Method `check()`: Checks for any changes in linked objects and updates.

Checks for any changes in any object and updates all linked objects if any are detected. Generally called automatically.

Usage:

```
Model$check(verbose = TRUE)
```

Arguments:

verbose Logical indicating whether to report if any updates are made, defaults to TRUE

Returns: Linked objects are updated by nothing is returned

Examples:

```

df <- nelder(~(c1(10)*t(5)) > ind(10))
df$int <- 0
df[df$c1 > 5, 'int'] <- 1
des <- Model$new(
  covariance = list(
    formula = ~ (1|gr(c1)*ar1(t)),
    parameters = c(0.25,0.8)),
  mean = list(
    formula = ~ factor(t) + int - 1,
    parameters = c(rep(0,5),0.6)),
  data = df,
  family = stats::binomial()
)
des$check() #does nothing
des$covariance$parameters <- c(0.1,0.9)
des$check() #updates
des$mean_function$parameters <- c(rnorm(5),0.1)
des$check() #updates

```

Method `update_parameters()`: Updates the parameters of the mean function and/or the covariance function

Usage:

```
Model$update_parameters(mean.pars = NULL, cov.pars = NULL, verbose = FALSE)
```

Arguments:

`mean.pars` (Optional) Vector of new mean function parameters
`cov.pars` (Optional) Vector of new covariance function(s) parameters
`verbose` Logical indicating whether to provide more detailed feedback

Examples:

```

df <- nelder(~(c1(10)*t(5)) > ind(10))
df$int <- 0
df[df$c1 > 5, 'int'] <- 1
des <- Model$new(
  covariance = list(
    formula = ~ (1|gr(c1)*ar1(t)),
  mean = list(
    formula = ~ factor(t) + int - 1),
  data = df,
  family = stats::binomial()
)
des$update_parameters(cov.pars = c(0.1,0.9))

```

Method `information_matrix()`: Generates the information matrix

Usage:

```
Model$information_matrix()
```

Returns: A PxP matrix

Method `power()`: Estimates the power of the design described by the model using the square root of the relevant element of the GLS variance matrix:

$$(X^T \Sigma^{-1} X)^{-1}$$

Note that this is equivalent to using the "design effect" for many models.

Usage:

```
Model$power(alpha = 0.05)
```

Arguments:

`alpha` Numeric between zero and one indicating the type I error rate. Default of 0.05.

Returns: A data frame describing the parameters, their values, expected standard errors and estimated power.

Examples:

```
df <- nelder(~(cl(10)*t(5)) > ind(10))
df$int <- 0
df[df$cl > 5, 'int'] <- 1
des <- Model$new(
  covariance = list(
    formula = ~ (1|gr(cl)) + (1|gr(cl*t)),
    parameters = c(0.25,0.1)),
  mean = list(
    formula = ~ factor(t) + int - 1,
    parameters = c(rep(0,5),0.6)),
  data = df,
  family = stats::gaussian(),
  var_par = 1
)
des$power() #power of 0.90 for the int parameter
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Model$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

[nelder](#), [MeanFunction](#), [Covariance](#)

Examples

```
## -----
## Method `Model$new`
## -----

#create a data frame describing a cross-sectional parallel cluster
```

```

#randomised trial
df <- nelder(~(cl(10)*t(5)) > ind(10))
df$int <- 0
df[df$cl > 5, 'int'] <- 1

mf1 <- MeanFunction$new(
  formula = ~ factor(t) + int - 1,
  data=df
)
cov1 <- Covariance$new(
  data = df,
  formula = ~ (1|gr(cl)) + (1|gr(cl*t))
)
des <- Model$new(
  covariance = cov1,
  mean = mf1,
  family = stats::gaussian(),
  var_par = 1
)

#alternatively we can pass the data directly to Model
#here we will specify a cohort study and provide parameter values
df <- nelder(~ind(20) * t(6))
df$int <- 0
df[df$t > 3, 'int'] <- 1

des <- Model$new(
  covariance = list(
    formula = ~ (1|gr(ind)),
    parameters = c(0.25)),
  mean = list(
    formula = ~ int,
    parameters = c(1,0.5)),
  data = df,
  family = stats::poisson())

#an example of a spatial grid with two time points
df <- nelder(~ (x(10)*y(10))*t(2))
spt_design <- Model$new(covariance = list( formula = ~(1|fexp(x,y)*ar1(t))),
  mean = list(formula = ~ 1),
  data = df,
  family = stats::gaussian())

## -----
## Method `Model$n_cluster`
## -----

df <- nelder(~(cl(10)*t(5)) > ind(10))
df$int <- 0
df[df$cl > 5, 'int'] <- 1

des <- Model$new(
  covariance = list(formula = ~ (1|gr(cl)) + (1|gr(cl*t))),

```

```

    mean = list(formula = ~ factor(t) + int - 1),
    data = df,
    family = stats::gaussian(),
    var_par = 1
  )
des$cluster() ## returns two levels of 10 and 50

## -----
## Method `Model$subset_rows`
## -----

#generate a stepped wedge design and remove the first sequence
des <- stepped_wedge(8,10,icc=0.05)
ids_to_keep <- which(des$mean_function$data$J!=1)
des$subset_rows(ids_to_keep)

## -----
## Method `Model$subset_cols`
## -----

#generate a stepped wedge design and remove first and last time periods
des <- stepped_wedge(8,10,icc=0.05)
des$subset_cols(c(2:8,10))

## -----
## Method `Model$sim_data`
## -----

df <- nelder(~(cl(10)*t(5)) > ind(10))
df$int <- 0
df[df$cl > 5, 'int'] <- 1
des <- Model$new(
  covariance = list(
    formula = ~ (1|gr(cl))*ar1(t),
    parameters = c(0.25,0.8)),
  mean = list(
    formula = ~ factor(t) + int - 1,
    parameters = c(rep(0,5),0.6)),
  data = df,
  family = stats::binomial()
)
ysim <- des$sim_data()

## -----
## Method `Model$check`
## -----

df <- nelder(~(cl(10)*t(5)) > ind(10))
df$int <- 0
df[df$cl > 5, 'int'] <- 1
des <- Model$new(
  covariance = list(
    formula = ~ (1|gr(cl))*ar1(t)),

```

```

    parameters = c(0.25,0.8)),
  mean = list(
    formula = ~ factor(t) + int - 1,
    parameters = c(rep(0,5),0.6)),
  data = df,
  family = stats::binomial()
)
des$check() #does nothing
des$covariance$parameters <- c(0.1,0.9)
des$check() #updates
des$mean_function$parameters <- c(rnorm(5),0.1)
des$check() #updates

## -----
## Method `Model$update_parameters`
## -----

df <- nelder(~(cl(10)*t(5)) > ind(10))
df$int <- 0
df[df$cl > 5, 'int'] <- 1
des <- Model$new(
  covariance = list(
    formula = ~ (1|gr(cl)*ar1(t))),
  mean = list(
    formula = ~ factor(t) + int - 1,
    data = df,
    family = stats::binomial()
  )
)
des$update_parameters(cov.pars = c(0.1,0.9))

## -----
## Method `Model$power`
## -----

df <- nelder(~(cl(10)*t(5)) > ind(10))
df$int <- 0
df[df$cl > 5, 'int'] <- 1
des <- Model$new(
  covariance = list(
    formula = ~ (1|gr(cl)) + (1|gr(cl*t)),
    parameters = c(0.25,0.1)),
  mean = list(
    formula = ~ factor(t) + int - 1,
    parameters = c(rep(0,5),0.6)),
  data = df,
  family = stats::gaussian(),
  var_par = 1
)
des$power() #power of 0.90 for the int parameter

```

Description

Generates a data frame expressing a block experimental structure using Nelder's formula

Usage

```
nelder(formula)
```

Arguments

formula A model formula. See details

Details

Nelder (1965) suggested a simple notation that could express a large variety of different blocked designs. The function 'nelder()' that generates a data frame of a design using the notation. There are two operations:

'>' (or \rightarrow in Nelder's notation) indicates "clustered in".

'*' (or \times in Nelder's notation) indicates a crossing that generates all combinations of two factors.

The implementation of this notation includes a string indicating the name of the variable and a number for the number of levels, such as 'abc(12)'. So for example '~cl(4) > ind(5)' means in each of five levels of 'cl' there are five levels of 'ind', and the individuals are different between clusters. The formula '~cl(4) * t(3)' indicates that each of the four levels of 'cl' are observed for each of the three levels of 't'. Brackets are used to indicate the order of evaluation. Some specific examples:

'~person(5) * time(10)': A cohort study with five people, all observed in each of ten periods 'time'

'~(cl(4) * t(3)) > ind(5)': A repeated-measures cluster study with four clusters (labelled 'cl'), each observed in each time period 't' with cross-sectional sampling and five individuals (labelled 'ind') in each cluster-period.

'~(cl(4) > ind(5)) * t(3)': A repeated-measures cluster cohort study with four clusters (labelled 'cl') with five individuals per cluster, and each cluster-individual combination is observed in each time period 't'.

'~((x(100) * y(100)) > hh(4)) * t(2)': A spatio-temporal grid of 100x100 and two time points, with 4 households per spatial grid cell.

Value

A list with the first member being the data frame

Examples

```
nelder(~(j(4) * t(5)) > i(5))
nelder(~person(5) * time(10))
```

nest_df	<i>Generate nested block structure</i>
---------	--

Description

Generate a data frame that nests one data frame in another

Usage

```
nest_df(df1, df2)
```

Arguments

df1	data frame
df2	data frame

Details

For two data frames 'df1' and 'df2', the function will return another data frame that nests 'df2' in 'df1'. So each row of 'df1' will be duplicated 'nrow(df2)' times and matched with 'df2'. The values of each 'df2' will be unique for each row of 'df1'

Value

data frame

Examples

```
nest_df(data.frame(t=1:4), data.frame(c1=1:3))
```

parallel_crt	<i>Generate a parallel cluster design</i>
--------------	---

Description

Generate a parallel cluster randomised trial design in glmmr

Usage

```
parallel_crt(  
  J,  
  M,  
  t,  
  ratio = 0.5,  
  beta = c(rep(0, t), 0),  
  icc,
```



```

cac = NULL,
iac = NULL,
var = 1,
family = stats::gaussian()
)

```

Arguments

J	Integer indicating the number of sequences such that there are J+1 time periods
M	Integer. The number of individual observations per cluster-period, assumed equal across all clusters
t	Integer. The number of time periods.
ratio	Numeric value indicating the proportion of clusters assigned to treatment. Default is 0.5.
beta	Vector of beta parameters to initialise the design, defaults to all zeros.
icc	Intraclass correlation coefficient. User may specify more than one value, see details.
cac	Cluster autocorrelation coefficient, optional and user may specify more than one value, see details
iac	Individual autocorrelation coefficient, optional and user may specify more than one value, see details
var	Assumed overall variance of the model, used to calculate the other covariance, see details
family	a family object

Details

The complete parallel cluster randomised trial design has J clusters observed over T time periods. A proportion ('ratio') of the clusters are assigned to treatment condition for the duration of the trial and the rest are control.

The assumed generalised linear mixed model for the parallel cluster trial is, for individual i, in cluster j, at time t:

$$\begin{aligned}
 y_{ijt} &\sim F(\mu_{ijt}, \sigma) \\
 \mu_{ijt} &= h^{-1}(x_{ijt}\beta + \alpha_{1j} + \alpha_{2jt} + \alpha_{3i}) \\
 \alpha_p &\sim N(0, \sigma_p^2), p = 1, 2, 3
 \end{aligned}$$

Defining τ as the total model variance, then the intraclass correlation coefficient (ICC) is

$$ICC = \frac{\sigma_1 + \sigma_2}{\tau}$$

the cluster autocorrelation coefficient (CAC) is :

$$CAC = \frac{\sigma_1}{\sigma_1 + \sigma_2}$$

and the individual autocorrelation coefficient as:

$$IAC = \frac{\sigma_3}{\tau(1 - ICC)}$$

When CAC and/or IAC are not specified in the call, then the respective random effects terms are assumed to be zero. For example, if IAC is not specified then α_{3i} does not appear in the model, and we have a cross-sectional sampling design; if IAC were specified then we would have a cohort.

For non-linear models, such as Poisson or Binomial models, there is no single obvious choice for ‘var_par’ (τ in the above formulae), as the models are heteroskedastic. Choices might include the variance at the mean values of the parameters or a reasonable choice based on the variance of the respective distribution.

If the user specifies more than one value for icc, cac, or iac, then a ModelSpace is returned with Models with every combination of parameters. This can be used in particular to generate a design space for optimal design analyses.

Value

A Model object with MeanFunction and Covariance objects, or a ModelSpace holding several such Model objects.

See Also

[Model](#)

Examples

```
#generate a simple design with only cluster random effects and 6 clusters in 3 time periods
# with 10 individuals in each cluster-period
des <- parallel_crt(J=6,M=10,t=3,icc=0.05)
# same design but with a cohort of individuals
des <- parallel_crt(J=6,M=10,t=3,icc=0.05, iac = 0.1)
# same design, but with two clusters per sequence and specifying the initial parameters
des <- parallel_crt(J=6,M=10,t=3,beta = c(rnorm(3,0,0.1),-0.1),icc=0.05, iac = 0.1)
# specifying multiple values of the variance parameters will return a design space
# with all designs with all the combinations of the variance parameter
des <- parallel_crt(J=6,M=10,t=3,icc=c(0.01,0.05), cac = c(0.5,0.7,0.9), iac = 0.1)
```

progress_bar

Generates a progress bar

Description

Prints a progress bar

Usage

```
progress_bar(i, n, len = 30)
```

Arguments

i	integer. The current iteration.
n	integer. The total number of iterations
len	integer. Length of the progress a number of characters

Value

A character string

Examples

```
progress_bar(10, 100)
```

sample_re	<i>Generates a sample of random effects</i>
-----------	---

Description

Generates a sample of random effects from the specified covariance matrix.

Usage

```
sample_re(cov, data, eff_range, gamma)
```

Arguments

cov	An integer matrix with columns of block identifier, dimension of block, function definition, number of variables in the argument to the function, and index of the parameters, respectively. Rows are specific functions of each block.
data	Vector of data. Created by flattening the matrices in column-major order of the data used in each block.
eff_range	Vector of values with the effective range parameters of the covariance functions, where required.
gamma	Vector of parameters used to generate the matrix D.

Value

A lower triangular matrix

staircase_crt	<i>Generate a staircase/diagonal trial design</i>
---------------	---

Description

Generate a staircase/diagonal cluster randomised trial design in glmmr

Usage

```
staircase_crt(
  J,
  M,
  beta = c(rep(0, J + 1), 0),
  icc,
  cac = NULL,
  iac = NULL,
  var = 1,
  family = stats::gaussian()
)
```

Arguments

J	Integer indicating the number of sequences such that there are J time periods
M	Integer. The number of individual observations per cluster-period, assumed equal across all clusters
beta	Vector of beta parameters to initialise the design, defaults to all zeros.
icc	Intraclass correlation coefficient. User may specify more than one value, see details.
cac	Cluster autocorrelation coefficient, optional and user may specify more than one value, see details
iac	Individual autocorrelation coefficient, optional and user may specify more than one value, see details
var	Assumed overall variance of the model, used to calculate the other covariance, see details
family	a family object

Details

The staircase/diagonal cluster randomised trial design has J sequences of clusters observed over J time periods, each sequence has 'nper' clusters. The first cluster is observed only once, in the first period in the treatment state. All the remaining clusters are observed twice, once in the control and once in the treatment state staggered over the course of the trial. In each time period there is therefore one cluster/sequence in the control state and one in the treatment state, so that the design produces a "staircase".

The assumed generalised linear mixed model for the staircase/diagonal cluster trial is, for individual i , in cluster j , at time t :

$$\begin{aligned} y_{ijt} &\sim F(\mu_{ijt}, \sigma) \\ \mu_{ijt} &= h^{-1}(x_{ijt}\beta + \alpha_{1j} + \alpha_{2jt} + \alpha_{3i}) \\ \alpha_p &\sim N(0, \sigma_p^2), p = 1, 2, 3 \end{aligned}$$

Defining τ as the total model variance, then the intraclass correlation coefficient (ICC) is

$$ICC = \frac{\sigma_1 + \sigma_2}{\tau}$$

the cluster autocorrelation coefficient (CAC) is :

$$CAC = \frac{\sigma_1}{\sigma_1 + \sigma_2}$$

and the individual autocorrelation coefficient as:

$$IAC = \frac{\sigma_3}{\tau(1 - ICC)}$$

When CAC and/or IAC are not specified in the call, then the respective random effects terms are assumed to be zero. For example, if IAC is not specified then α_{3i} does not appear in the model, and we have a cross-sectional sampling design; if IAC were specified then we would have a cohort.

For non-linear models, such as Poisson or Binomial models, there is no single obvious choice for ‘var_par’ (τ in the above formulae), as the models are heteroskedastic. Choices might include the variance at the mean values of the parameters or a reasonable choice based on the variance of the respective distribution.

If the user specifies more than one value for `icc`, `cac`, or `iac`, then a `ModelSpace` is returned with `Models` with every combination of parameters. This can be used in particular to generate a design space for optimal design analyses.

Value

A `Model` object with `MeanFunction` and `Covariance` objects, or a `ModelSpace` holding several such `Model` objects.

See Also

[Model](#)

Examples

```
#generate a simple design with only cluster random effects and 6 clusters with 10
#individuals in each cluster-period
des <- staircase_crt(6,10,icc=0.05)
# same design but with a cohort of individuals
des <- staircase_crt(6,10,icc=0.05, iac = 0.1)
# same design, but with two clusters per sequence and specifying the initial parameters
des <- staircase_crt(6,10,beta = c(rnorm(7,0,0.1),-0.1),icc=0.05, iac = 0.1)
# specifying multiple values of the variance parameters will return a design space
# with all designs with all the combinations of the variance parameter
des <- staircase_crt(6,10,icc=c(0.01,0.05), cac = c(0.5,0.7,0.9), iac = 0.1)
```

stepped_wedge	<i>Generate a stepped-wedge design</i>
---------------	--

Description

Generate a stepped-wedge cluster randomised trial design in glmmr

Usage

```
stepped_wedge(
  J,
  M,
  nper = 1,
  beta = c(rep(0, J + 1), 0),
  icc,
  cac = NULL,
  iac = NULL,
  var = 1,
  family = stats::gaussian()
)
```

Arguments

J	Integer indicating the number of sequences such that there are J+1 time periods
M	Integer. The number of individual observations per cluster-period, assumed equal across all clusters
nper	Integer. The number of clusters per sequence, default is one.
beta	Vector of beta parameters to initialise the design, defaults to all zeros.
icc	Intraclass correlation coefficient. User may specify more than one value, see details.
cac	Cluster autocorrelation coefficient, optional and user may specify more than one value, see details
iac	Individual autocorrelation coefficient, optional and user may specify more than one value, see details
var	Assumed overall variance of the model, used to calculate the other covariance, see details
family	a family object

Details

The complete stepped-wedge cluster randomised trial design has J sequences of clusters observed over J+1 time periods, each sequence has 'nper' clusters. The first time period has all clusters in the control state, and the final time period has all clusters in the treatment state, with one sequence switching between the two each period to create the "step".

The assumed generalised linear mixed model for the stepped-wedge cluster trial is, for individual i , in cluster j , at time t :

$$y_{ijt} \sim F(\mu_{ijt}, \sigma)$$

$$\mu_{ijt} = h^{-1}(x_{ijt}\beta + \alpha_{1j} + \alpha_{2jt} + \alpha_{3i})$$

$$\alpha_p. \sim N(0, \sigma_p^2), p = 1, 2, 3$$

Defining τ as the total model variance, then the intraclass correlation coefficient (ICC) is

$$ICC = \frac{\sigma_1 + \sigma_2}{\tau}$$

the cluster autocorrelation coefficient (CAC) is :

$$CAC = \frac{\sigma_1}{\sigma_1 + \sigma_2}$$

and the individual autocorrelation coefficient as:

$$IAC = \frac{\sigma_3}{\tau(1 - ICC)}$$

When CAC and/or IAC are not specified in the call, then the respective random effects terms are assumed to be zero. For example, if IAC is not specified then α_{3i} does not appear in the model, and we have a cross-sectional sampling design; if IAC were specified then we would have a cohort.

For non-linear models, such as Poisson or Binomial models, there is no single obvious choice for ‘var_par’ (τ in the above formulae), as the models are heteroskedastic. Choices might include the variance at the mean values of the parameters or a reasonable choice based on the variance of the respective distribution.

If the user specifies more than one value for icc, cac, or iac, then a ModelSpace is returned with Models with every combination of parameters. This can be used in particular to generate a design space for optimal design analyses.

Value

A Model object with MeanFunction and Covariance objects, or a ModelSpace holding several such Model objects.

See Also

[Model](#)

Examples

```
#generate a simple design with only cluster random effects and 6 clusters with 10
#individuals in each cluster-period
des <- stepped_wedge(6,10,icc=0.05)
# same design but with a cohort of individuals
des <- stepped_wedge(6,10,icc=0.05, iac = 0.1)
# same design, but with two clusters per sequence and specifying the initial parameters
des <- stepped_wedge(6,10,beta = c(rnorm(7,0,0.1),-0.1),icc=0.05, iac = 0.1)
# specifying multiple values of the variance parameters will return a design space
# with all designs with all the combinations of the variance parameter
des <- stepped_wedge(6,10,icc=c(0.01,0.05), cac = c(0.5,0.7,0.9), iac = 0.1)
```

Index

* package

glmmrBase-package, 2

Beta, 4, 22

binomial, 22

Covariance, 4, 20–22, 27

cross_df, 9

cycles, 10

dfactor, 10

dfexp, 11

didentity, 11

family, 15, 16, 22, 33, 36, 38

formula, 16

Gamma, 22

gaussian, 22

gen_dhdmu, 13

genCholD, 11

genD, 12

glmmrBase (glmmrBase-package), 2

glmmrBase-package, 2

match_rows, 13

Matrix, 22

MeanFunction, 14, 20–22, 27

Model, 5, 14, 20, 34, 37, 39

nelder, 27, 30

nest_df, 32

parallel_crt, 32

poisson, 22

progress_bar, 34

sample_re, 35

staircase_crt, 36

stepped_wedge, 38