# Package: smimodel (via r-universe)

June 8, 2026

**Title** Sparse Multiple Index Models for Nonparametric Forecasting

**Version** 0.1.3

**Description** Implements a general algorithm for estimating Sparse Multiple Index (SMI) models for nonparametric forecasting and prediction. Estimation of SMI models requires the Gurobi mixed integer programming (MIP) solver via the gurobi R package. To use this functionality, the Gurobi Optimizer must be installed, and a valid license obtained and activated from <https://www.gurobi.com>. The gurobi R package must then be installed and configured following the instructions at <https://support.gurobi.com/hc/en-us/articles/14462206790033-How-do-I-install-Gurobi-for-R>. The package also includes functions for fitting nonparametric additive models with backward elimination, group-wise additive index models, and projection pursuit regression models as benchmark comparison methods. In addition, it provides tools for generating prediction intervals to quantify uncertainty in point forecasts produced by the SMI model and benchmark models, using the classical block bootstrap and a new method called conformal bootstrap, which integrates block bootstrap with split conformal prediction.

**License** GPL (>= 3)

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 4.1.0)

**Imports** cgaim, conformalForecast, dplyr, furrr, future, generics, ggplot2, gratia, gtools, Matrix, methods, mgcv, purrr, ROI, tibble, tidyselect, tidyr, tsibble, utils

**Suggests** gurobi, knitr, lubridate, rmarkdown, testthat (>= 3.0.0)

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**URL** https://github.com/nuwani-palihawadana/smimodel, https://nuwani-palihawadana.github.io/smimodel/

**BugReports** https://github.com/nuwani-palihawadana/smimodel/issues

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**Config/pak/sysreqs** cmake make libicu-dev

**Repository** https://nuwani-palihawadana.r-universe.dev

**Date/Publication** 2026-04-02 12:45:55 UTC

**RemoteUrl** https://github.com/nuwani-palihawadana/smimodel

**RemoteRef** HEAD

**RemoteSha** e599f7f821d0ddb6f89ab8770428ff1d5f170652

# Contents

---

`smimodel-package` *smimodel: Sparse Multiple Index Models for Nonparametric Forecasting*

---

## Description

Implements a general algorithm for estimating Sparse Multiple Index (SMI) models for nonparametric forecasting and prediction. Estimation of SMI models requires the Gurobi mixed integer programming (MIP) solver via the gurobi R package. To use this functionality, the Gurobi Optimizer must be installed, and a valid license obtained and activated from https://www.gurobi.com. The gurobi R package must then be installed and configured following the instructions at https://support.gurobi.com/hc/en-us/articles/14462206790033-How-do-I-install-Gurobi-for-R.

The package also includes functions for fitting nonparametric additive models with backward elimination, group-wise additive index models, and projection pursuit regression models as benchmark comparison methods. In addition, it provides tools for generating prediction intervals to quantify uncertainty in point forecasts produced by the SMI model and benchmark models, using the classical block bootstrap and a new method called conformal bootstrap, which integrates block bootstrap with split conformal prediction.

## Author(s)

**Maintainer**: Nuwani Palihawadana <nuwanipalihawadana@gmail.com> (ORCID) [copyright holder]

Other contributors:

- Xiaoqian Wang <Xiaoqian.Wang@amss.ac.cn> (ORCID) [contributor]

## See Also

Useful links:

- https://github.com/nuwani-palihawadana/smimodel
- https://nuwani-palihawadana.github.io/smimodel/
- Report bugs at https://github.com/nuwani-palihawadana/smimodel/issues

---

`allpred_index` *Constructing index coefficient vectors with all predictors in each index*

---

## Description

Constructs vectors of coefficients for each index including a coefficient for all the predictors that are entering indices. i.e. if a coefficient is not provided for a particular predictor in a particular index, the function will replace the missing coefficient with a zero.

## Usage

```
allpred_index(num_pred, num_ind, ind_pos, alpha)
```

## Arguments

| | |
|---|---|
| `num_pred` | Number of predictors. |
| `num_ind` | Number of indices. |
| `ind_pos` | A list of length = `num_ind` that indicates which predictors belong to which index. |
| `alpha` | A vector of index coefficients. |

## Value

A list containing the following components:

| | |
|---|---|
| alpha_init_new | A numeric vector of index coefficients. |
| index | An integer vector that assigns group indices for each predictor. |
| index_positions | A list of length = num_ind that indicates which predictors belong to which index. |

---

augment.backward *Augment function for class* backward

---

## Description

Generates residuals and fitted values of a fitted backward object.

## Usage

```
## S3 method for class 'backward'
augment(x, ...)
```

## Arguments

| | |
|---|---|
| x | A backward object. |
| ... | Other arguments not currently used. |

## Value

A tibble.

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1205
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
```

```
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Validation set
sim_val <- sim_data[1001:1200, ]

# Predictors taken as non-linear variables
s.vars <- colnames(sim_data)[3:8]

# Model fitting
backwardModel <- model_backward(data = sim_train,
                                val.data = sim_val,
                                yvar = "y",
                                s.vars = s.vars)
# Obtain residuals and fitted values
augment(backwardModel)
```

---

`augment.gaimFit` *Augment function for class* `gaimFit`

---

## Description

Generates residuals and fitted values of a fitted `gaimFit` object.

## Usage

```
## S3 method for class 'gaimFit'
augment(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `gaimFit` object. |
| `...` | Other arguments not currently used. |

## Value

A `tibble`.

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1005
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Predictors taken as index variables
index.vars <- colnames(sim_data)[3:7]

# Assign group indices for each predictor
index.ind = c(rep(1, 3), rep(2, 2))

# Predictors taken as non-linear variables not entering indices
s.vars = "x_lag_005"

# Model fitting
gaimModel <- model_gaim(data = sim_data,
                        yvar = "y",
                        index.vars = index.vars,
                        index.ind = index.ind,
                        s.vars = s.vars)
# Obtain residuals and fitted values
augment(gaimModel)
```

---

`augment.gamFit` *Augment function for class* `gamFit`

---

## Description

Generates residuals and fitted values of a fitted `gamFit` object.

## Usage

```
## S3 method for class 'gamFit'
augment(x, ...)
```

## Arguments

| | |
|---|---|
| x | A gamFit object. |
| ... | Other arguments not currently used. |

## Value

A tibble.

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1005
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Predictors taken as non-linear variables
s.vars <- colnames(sim_data)[3:6]

# Predictors taken as linear variables
linear.vars <- colnames(sim_data)[7:8]

# Model fitting
gamModel <- model_gam(data = sim_data,
                      yvar = "y",
                      s.vars = s.vars,
                      linear.vars = linear.vars)

# Obtain residuals and fitted values
```

```
augment(gamModel)
```

---

`augment.lmFit` *Augment function for class* `lmFit`

---

## Description

Generates residuals and fitted values of a fitted `lmFit` object.

## Usage

```
## S3 method for class 'lmFit'
augment(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `lmFit` object. |
| `...` | Other arguments not currently used. |

## Value

A `tibble`.

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1005
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)
```

```
# Predictor variables
linear.vars <- colnames(sim_data)[3:8]

# Model fitting
lmModel <- model_lm(data = sim_data,
                    yvar = "y",
                    linear.vars = linear.vars)
# Obtain residuals and fitted values
augment(lmModel)
```

---

`augment.pprFit` *Augment function for class* `pprFit`

---

## Description

Generates residuals and fitted values of a fitted `pprFit` object.

## Usage

```
## S3 method for class 'pprFit'
augment(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `pprFit` object. |
| `...` | Other arguments not currently used. |

## Value

A `tibble`.

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1005
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
```

```
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Index variables
index.vars <- colnames(sim_data)[3:8]

# Model fitting
pprModel <- model_ppr(data = sim_data,
                      yvar = "y",
                      index.vars = index.vars)

# Obtain residuals and fitted values
augment(pprModel)
```

---

`augment.smimodel` *Augment function for class* `smimodel`

---

## Description

Generates residuals and fitted values of a fitted `smimodel` object.

## Usage

```
## S3 method for class 'smimodel'
augment(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `smimodel` object. |
| `...` | Other arguments not currently used. |

## Value

A `tibble`.

## Examples

```
if(requireNamespace("gurobi", quietly = TRUE)){
  library(dplyr)
  library(ROI)
  library(tibble)
  library(tidyr)
  library(tsibble)
```

```
  # Simulate data
  n = 1005
  set.seed(123)
  sim_data <- tibble(x_lag_000 = runif(n)) |>
    mutate(
      # Add x_lags
      x_lag = lag_matrix(x_lag_000, 5)) |>
    unpack(x_lag, names_sep = "_") |>
    mutate(
      # Response variable
      y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
      # Add an index to the data set
      inddd = seq(1, n)) |>
    drop_na() |>
    select(inddd, y, starts_with("x_lag")) |>
    # Make the data set a `tsibble`
    as_tsibble(index = inddd)

  # Index variables
  index.vars <- colnames(sim_data)[3:8]

  # Model fitting
  smimodel_ppr <- model_smimodel(data = sim_data,
                                 yvar = "y",
                                 index.vars = index.vars,
                                 initialise = "ppr")

  # Obtain residuals and fitted values
  augment(smimodel_ppr)
}
```

---

`augment.smimodelFit` *Augment function for class* `smimodelFit`

---

## Description

Generates residuals and fitted values of a fitted `smimodelFit` object.

## Usage

```
## S3 method for class 'smimodelFit'
augment(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `smimodelFit` object. |
| `...` | Other arguments not currently used. |

### Value

A tibble.

---

autoplot.smimodel *Plot estimated smooths from a fitted* smimodel

---

### Description

Plots the graphs of fitted spline(s). If a set of multiple models are fitted, plots graphs of fitted spline(s) of a specified model (in argument model) out of the set of multiple models fitted.

### Usage

```
## S3 method for class 'smimodel'
autoplot(object, model = 1, ...)
```

### Arguments

| | |
|---|---|
| object | A smimodel object. |
| model | An integer to indicate the smooths of which model (out of the set of multiple models fitted) to be plotted. |
| ... | Other arguments not currently used. |

### Value

Plot(s) of fitted spline(s).

### Examples

```
if(requireNamespace("gurobi", quietly = TRUE)){
  library(dplyr)
  library(ROI)
  library(tibble)
  library(tidyr)
  library(tsibble)

  # Simulate data
  n = 1005
  set.seed(123)
  sim_data <- tibble(x_lag_000 = runif(n)) |>
    mutate(
      # Add x_lags
      x_lag = lag_matrix(x_lag_000, 5)) |>
    unpack(x_lag, names_sep = "_") |>
    mutate(
      # Response variable
      y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
      # Add an index to the data set
```

```
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Index variables
index.vars <- colnames(sim_data)[3:8]

# Model fitting
smimodel_ppr <- model_smimodel(data = sim_data,
                               yvar = "y",
                               index.vars = index.vars,
                               initialise = "ppr")

autoplot(smimodel_ppr)
}
```

---

avgCoverage *Calculate interval forecast coverage*

---

## Description

This is a wrapper for the function `conformalForecast::coverage`. Calculates the mean coverage and the ifinn matrix for prediction intervals on validation set. If `window` is not `NULL`, a matrix of the rolling means of interval forecast coverage is also returned.

## Usage

```
avgCoverage(object, level = 95, window = NULL, na.rm = FALSE)
```

## Arguments

| | |
|---|---|
| `object` | An object of class `bb_cvforecast` or `cb_cvforecast`. |
| `level` | Target confidence level for prediction intervals. |
| `window` | If not `NULL`, the rolling mean matrix for coverage is also returned. |
| `na.rm` | A `logical` indicating whether `NA` values should be stripped before the rolling mean computation proceeds. |

## Value

A list of class `coverage` with the following components:

| | |
|---|---|
| `mean` | Mean coverage across the validation set. |
| `ifinn` | A indicator matrix as a multivariate time series, where the $h$th column holds the coverage for forecast horizon $h$. The time index corresponds to the period for which the forecast is produced. |
| `rollmean` | If `window` is not `NULL`, a matrix of the rolling means of interval forecast coverage will be returned. |

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1055
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Test set
sim_test <- sim_data[1001:1050, ]

# Index variables
index.vars <- colnames(sim_data)[3:8]

# Model fitting
pprModel <- model_ppr(data = sim_train,
                      yvar = "y",
                      index.vars = index.vars)

# Conformal bootstrap prediction intervals (2-steps-ahead interval forecasts)
set.seed(12345)
pprModel_cb <- cb_cvforecast(object = pprModel,
                             data = sim_data,
                             yvar = "y",
                             predictor.vars = index.vars,
                             h = 2,
                             ncal = 30,
                             num.futures = 100,
                             window = 1000)

# Mean coverage of generated 95% conformal bootstrap prediction intervals
cov_data <- avgCoverage(object = pprModel_cb)
cov_data$mean
```

avgWidth *Calculate interval forecast width*

## Description

This is a wrapper for the function `conformalForecast::width`. Calculates the mean width of prediction intervals on the validation set. If `window` is not `NULL`, a matrix of the rolling means of interval width is also returned. If `includemedian` is `TRUE`, the information of the median interval width will be returned.

## Usage

```
avgWidth(
  object,
  level = 95,
  includemedian = FALSE,
  window = NULL,
  na.rm = FALSE
)
```

## Arguments

| | |
|---|---|
| `object` | An object of class `bb_cvforecast` or `cb_cvforecast`. |
| `level` | Target confidence level for prediction intervals. |
| `includemedian` | If `TRUE`, the median interval width will also be returned. |
| `window` | If not `NULL`, the rolling mean (and rolling median if applicable) matrix for interval width will also be returned. |
| `na.rm` | A logical indicating whether `NA` values should be stripped before the rolling mean and rolling median computation proceeds. |

## Value

A list of class `width` with the following components:

| | |
|---|---|
| `width` | Forecast interval width as a multivariate time series, where the $h$th column holds the interval width for the forecast horizon $h$. The time index corresponds to the period for which the forecast is produced. |
| `mean` | Mean interval width across the validation set. |
| `rollmean` | If `window` is not `NULL`, a matrix of the rolling means of interval width will be returned. |
| `median` | Median interval width across the validation set. |
| `rollmedian` | If `window` is not `NULL`, a matrix of the rolling medians of interval width will be returned. |

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1055
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Test set
sim_test <- sim_data[1001:1050, ]

# Index variables
index.vars <- colnames(sim_data)[3:8]

# Model fitting
pprModel <- model_ppr(data = sim_train,
                      yvar = "y",
                      index.vars = index.vars)

# Conformal bootstrap prediction intervals (2-steps-ahead interval forecasts)
set.seed(12345)
pprModel_cb <- cb_cvforecast(object = pprModel,
                             data = sim_data,
                             yvar = "y",
                             predictor.vars = index.vars,
                             h = 2,
                             ncal = 30,
                             num.futures = 100,
                             window = 1000)

# Mean width of generated 95% conformal bootstrap prediction intervals
width_data <- avgWidth(object = pprModel_cb)
width_data$mean
```

bb_cvforecast *Single season block bootstrap prediction intervals through time series cross-validation forecasting*

## Description

Compute prediction intervals by applying the single season block bootstrap method to subsets of time series data using a rolling forecast origin.

## Usage

```
bb_cvforecast(
  object,
  data,
  yvar,
  neighbour = 0,
  predictor.vars,
  h = 1,
  season.period = 1,
  m = 1,
  num.futures = 1000,
  level = c(80, 95),
  forward = TRUE,
  initial = 1,
  window = NULL,
  roll.length = 1,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colNames = NULL,
  na.rm = TRUE,
  verbose = list(solver = FALSE, progress = FALSE),
  ...
)
```

## Arguments

| | |
|---|---|
| object | Fitted model object of class smimodel, backward, gaimFit or pprFit. |
| data | Data set. Must be a data set of class tsibble.(Make sure there are no additional date or time related variables except for the index of the tsibble). If multiple models are fitted, the grouping variable should be the key of the tsibble. If a key is not specified, a dummy key with only one level will be created. |
| yvar | Name of the response variable as a character string. |

`neighbour` If multiple models are fitted: Number of neighbours of each key (i.e. grouping variable) to be considered in model fitting to handle smoothing over the key. Should be an `integer`. If `neighbour = x`, `x` number of keys before the key of interest and `x` number of keys after the key of interest are grouped together for model fitting. The default is `neighbour = 0` (i.e. no neighbours are considered for model fitting).

`predictor.vars` A character vector of names of the predictor variables.

`h` Forecast horizon.

`season.period` Length of the seasonal period.

`m` Multiplier. (Block size = `NULLseason.period * m`)

`num.futures` Number of possible future sample paths to be generated.

`level` Confidence level for prediction intervals.

`forward` If `TRUE`, the final forecast origin for forecasting is $y_T$. Otherwise, the final forecast origin is $y_{T-1}$.

`initial` Initial period of the time series where no cross-validation forecasting is performed.

`window` Length of the rolling window. If `NULL`, a rolling window will not be used.

`roll.length` Number of observations by which each rolling/expanding window should be rolled forward.

`exclude.trunc` The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.)

`recursive` Whether to obtain recursive forecasts or not (default - `FALSE`).

`recursive_colNames`
If `recursive = TRUE`, a character vector giving the names of the columns in test data to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. `lag_1, lag_2, ..., lag_m`, `lag_m` = maximum lag used) in `data`, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors.

`na.rm` logical; if `TRUE` (default), any `NA` and `NaN`'s are removed from the sample before the quantiles are computed.

`verbose` A named list controlling verbosity options. Defaults to `list(solver = FALSE, progress = FALSE)`.

**solver** Logical. If TRUE, prints detailed solver output when the SMI model is used.

**progress** Logical. If TRUE, prints cross-validation progress messages (all models) and optimisation algorithm progress messages (SMI model only).

`...` Other arguments not currently used.

## Value

An object of class `bb_cvforecast`, which is a list that contains following elements:

| | |
|---|---|
| `x` | The original time series. |
| `method` | A character string "bb_cvforecast". |
| `fit_times` | The number of times the model is fitted in cross-validation. |
| `mean` | Point forecasts as a multivariate time series, where the $h^{th}$ column holds the point forecasts for forecast horizon $h$. The time index corresponds to the period for which the forecast is produced. |
| `res` | The matrix of in-sample residuals produced in cross-validation. The number of rows corresponds to `fit_times`, and the row names corresponds the time index of the forecast origin of the corresponding cross-validation iteration. |
| `model_fit` | Models fitted in cross-validation. |
| `level` | The confidence values associated with the prediction intervals. |
| `lower` | A list containing lower bounds for prediction intervals for each level. Each element within the list will be a multivariate time series with the same dimensional characteristics as `mean`. |
| `upper` | A list containing upper bounds for prediction intervals for each level. Each element within the list will be a multivariate time series with the same dimensional characteristics as `mean`. |
| `possible_futures` | A list of matrices containing future sample paths generated at each cross-validation step. |

## See Also

`cb_cvforecast`

## Examples

```
if(requireNamespace("gurobi", quietly = TRUE)){
  library(dplyr)
  library(ROI)
  library(tibble)
  library(tidyr)
  library(tsibble)

  # Simulate data
  n = 1105
  set.seed(123)
  sim_data <- tibble(x_lag_000 = runif(n)) |>
    mutate(
      # Add x_lags
      x_lag = lag_matrix(x_lag_000, 5)) |>
    unpack(x_lag, names_sep = "_") |>
    mutate(
      # Response variable
      y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 +
```

```
    (0.35*x_lag_002 + 0.7*x_lag_005)^2 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Index variables
index.vars <- colnames(sim_data)[3:8]

# Training set
sim_train <- sim_data[1:1000, ]
# Test set
sim_test <- sim_data[1001:1100, ]

# Model fitting
smimodel_ppr <- model_smimodel(data = sim_train,
                               yvar = "y",
                               index.vars = index.vars,
                               initialise = "ppr")

# Block bootstrap prediction intervals (3-steps-ahead interval forecasts)
set.seed(12345)
smimodel_ppr_bb <- bb_cvforecast(object = smimodel_ppr,
                                 data = sim_data,
                                 yvar = "y",
                                 predictor.vars = index.vars,
                                 h = 3,
                                 num.futures = 50,
                                 window = 1000)
}
```

---

`blockBootstrap` *Futures through single season block bootstrapping*

---

## Description

Generates possible future sample paths by applying the single season block bootstrap method.

## Usage

```
blockBootstrap(
  object,
  newdata,
  resids,
  preds,
  season.period = 1,
```

```
  m = 1,
  num.futures = 1000,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL
)
```

## Arguments

| | |
|---|---|
| object | Fitted model object. |
| newdata | Test data set. Must be a data set of class tsibble. |
| resids | In-sample residuals from the fitted model. |
| preds | Predictions for the test set (i.e. data for the forecast horizon). |
| season.period | Length of the seasonal period. |
| m | Multiplier. (Block size = season.period * m) |
| num.futures | Number of possible future sample paths to be generated. |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, The range of column numbers in test data to be filled with forecasts. |

## Value

A matrix of simulated future sample paths.

---

# cb_cvforecast *Conformal bootstrap prediction intervals through time series cross-validation forecasting*

---

## Description

Compute prediction intervals by applying the conformal bootstrap method to subsets of time series data using a rolling forecast origin.

## Usage

```
cb_cvforecast(
  object,
  data,
  yvar,
  neighbour = 0,
  predictor.vars,
  h = 1,
```

```
  ncal = 100,
  num.futures = 1000,
  level = c(80, 95),
  forward = TRUE,
  initial = 1,
  window = NULL,
  roll.length = 1,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colNames = NULL,
  na.rm = TRUE,
  nacheck_frac_numerator = 2,
  nacheck_frac_denominator = 3,
  verbose = list(solver = FALSE, progress = FALSE),
  ...
)
```

## Arguments

| | |
|---|---|
| object | Fitted model object of class smimodel, backward, gaimFit or pprFit. |
| data | Data set. Must be a data set of class tsibble.(Make sure there are no additional date or time related variables except for the index of the tsibble). If multiple models are fitted, the grouping variable should be the key of the tsibble. If a key is not specified, a dummy key with only one level will be created. |
| yvar | Name of the response variable as a character string. |
| neighbour | If multiple models are fitted: Number of neighbours of each key (i.e. grouping variable) to be considered in model fitting to handle smoothing over the key. Should be an integer. If neighbour = x, x number of keys before the key of interest and x number of keys after the key of interest are grouped together for model fitting. The default is neighbour = 0 (i.e. no neighbours are considered for model fitting). |
| predictor.vars | A character vector of names of the predictor variables. |
| h | Forecast horizon. |
| ncal | Length of a calibration window. |
| num.futures | Number of possible future sample paths to be generated in bootstrap. |
| level | Confidence level for prediction intervals. |
| forward | If TRUE, the final forecast origin for forecasting is $y_T$. Otherwise, the final forecast origin is $y_{T-1}$. |
| initial | Initial period of the time series where no cross-validation forecasting is performed. |
| window | Length of the rolling window. If NULL, a rolling window will not be used. |
| roll.length | Number of observations by which each rolling/expanding window should be rolled forward. |

`exclude.trunc` The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.)

`recursive` Whether to obtain recursive forecasts or not (default - `FALSE`).

`recursive_colNames`
If `recursive = TRUE`, a character vector giving the names of the columns in test data to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. `lag_1, lag_2, ..., lag_m`, `lag_m` = maximum lag used) in `data`, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors.

`na.rm` logical; if `TRUE` (default), any `NA` and `NaN`'s are removed from the sample before the quantiles are computed.

`nacheck_frac_numerator`
Numerator of the fraction of non-missing values that is required in a test set.

`nacheck_frac_denominator`
Denominator of the fraction of non-missing values that is required in a test set.

`verbose` A named list controlling verbosity options. Defaults to `list(solver = FALSE, progress = FALSE)`.

**solver** Logical. If TRUE, prints detailed solver output when the SMI model is used.

**progress** Logical. If TRUE, prints cross-validation progress messages (all models) and optimisation algorithm progress messages (SMI model only).

`...` Other arguments not currently used.

## Value

An object of class `cb_cvforecast`, which is a list that contains following elements:

`x` The original time series.

`method` A character string "cb_cvforecast".

`fit_times` The number of times the model is fitted in cross-validation.

`mean` Point forecasts as a multivariate time series, where the $h^{th}$ column holds the point forecasts for forecast horizon $h$. The time index corresponds to the period for which the forecast is produced.

`error` Forecast errors given by $e_{t+h|t} = y_{t+h} - \hat{y}_{t+h|t}$.

`res` The matrix of in-sample residuals produced in cross-validation.

`level` The confidence levels associated with the prediction intervals.

`cal_times` The number of calibration windows considered in cross-validation.

`num_cal` The number of non-missing multi-step forecast errors in each calibration window.

skip_cal — An indicator vector indicating whether a calibration window is skipped without constructing prediction intervals due to missing model or missing data in the test set.

lower — A list containing lower bounds for prediction intervals for each level. Each element within the list will be a multivariate time series with the same dimensional characteristics as mean.

upper — A list containing upper bounds for prediction intervals for each level. Each element within the list will be a multivariate time series with the same dimensional characteristics as mean.

possible_futures — A list of matrices containing future sample paths generated at each calibration step.

## See Also

bb_cvforecast

## Examples

```
if(requireNamespace("gurobi", quietly = TRUE)){
  library(dplyr)
  library(ROI)
  library(tibble)
  library(tidyr)
  library(tsibble)

  # Simulate data
  n = 1105
  set.seed(123)
  sim_data <- tibble(x_lag_000 = runif(n)) |>
    mutate(
      # Add x_lags
      x_lag = lag_matrix(x_lag_000, 5)) |>
    unpack(x_lag, names_sep = "_") |>
    mutate(
      # Response variable
      y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 +
      (0.35*x_lag_002 + 0.7*x_lag_005)^2 + rnorm(n, sd = 0.1),
      # Add an index to the data set
      inddd = seq(1, n)) |>
    drop_na() |>
    select(inddd, y, starts_with("x_lag")) |>
    # Make the data set a `tsibble`
    as_tsibble(index = inddd)

  # Index variables
  index.vars <- colnames(sim_data)[3:8]

  # Training set
  sim_train <- sim_data[1:1000, ]
  # Test set
```

```
sim_test <- sim_data[1001:1100, ]

# Model fitting
smimodel_ppr <- model_smimodel(data = sim_train,
                               yvar = "y",
                               index.vars = index.vars,
                               initialise = "ppr")

# Conformal bootstrap prediction intervals (3-steps-ahead interval forecasts)
set.seed(12345)
smimodel_ppr_cb <- cb_cvforecast(object = smimodel_ppr,
                                 data = sim_data,
                                 yvar = "y",
                                 predictor.vars = index.vars,
                                 h = 3,
                                 ncal = 30,
                                 num.futures = 100,
                                 window = 1000)
}
```

---

`eliminate` *Eliminate a variable and fit a nonparametric additive model*

---

## Description

Eliminates a specified variable and fits a nonparametric additive model with remaining variables, and returns validation set MSE. This is an internal function of the package, and designed to be called from `model_backward`.

## Usage

```
eliminate(
  ind,
  train,
  val,
  yvar,
  family = gaussian(),
  s.vars = NULL,
  s.basedim = NULL,
  linear.vars = NULL,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL
)
```

## Arguments

| | |
|---|---|
| ind | An integer corresponding to the position of the predictor variable to be eliminated when fitting the model. (i.e. the function will combine s.vars and linear.vars in a single vector and eliminate the element corresponding to ind.) |
| train | The data set on which the model(s) will be trained. Must be a data set of class tsibble. |
| val | Validation data set. (The data set on which the model selection will be performed.) Must be a data set of class tsibble. |
| yvar | Name of the response variable as a character string. |
| family | A description of the error distribution and link function to be used in the model (see glm and family). |
| s.vars | A character vector of names of the predictor variables for which splines should be fitted (i.e. non-linear predictors). |
| s.basedim | Dimension of the bases used to represent the smooth terms corresponding to s.vars. (For more information refer mgcv::s().) |
| linear.vars | A character vector of names of the predictor variables that should be included linearly into the model (i.e. linear predictors). |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in val that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.) |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in val.data to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in val.data, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors. |

## Value

A numeric.

---

forecast.backward *Forecasting using nonparametric additive models with backward elimination*

---

## Description

Returns forecasts and other information for nonparametric additive models with backward elimination.

## Usage

```
## S3 method for class 'backward'
forecast(
  object,
  h = 1,
  level = c(80, 95),
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | An object of class backward. Usually the result of a call to model_backward. |
| h | Forecast horizon. |
| level | Confidence level for prediction intervals. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. |
| ... | Other arguments not currently used. |

## Value

An object of class forecast. Here, it is a list containing the following elements:

| | |
|---|---|
| method | The name of the forecasting method as a character string. |
| model | The fitted model. |
| mean | Point forecasts as a time series. |
| residuals | Residuals from the fitted model. |
| fitted | Fitted values (one-step forecasts). |

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1215
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Validation set
sim_val <- sim_data[1001:1200, ]
# Test set
sim_test <- sim_data[1201:1210, ]

# Predictors taken as non-linear variables
s.vars <- colnames(sim_data)[3:8]

# Model fitting
backwardModel <- model_backward(data = sim_train,
                                val.data = sim_val,
                                yvar = "y",
                                s.vars = s.vars)
forecast(backwardModel, newdata = sim_test)
```

---

`forecast.gaimFit` *Forecasting using GAIMs*

---

## Description

Returns forecasts and other information for GAIMs.

## Usage

```
## S3 method for class 'gaimFit'
forecast(
  object,
  h = 1,
  level = c(80, 95),
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | An object of class gaimFit. Usually the result of a call to model_gaim. |
| h | Forecast horizon. |
| level | Confidence level for prediction intervals. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. |
| ... | Other arguments not currently used. |

## Value

An object of class forecast. Here, it is a list containing the following elements:

| | |
|---|---|
| method | The name of the forecasting method as a character string. |
| model | The fitted model. |
| mean | Point forecasts as a time series. |
| residuals | Residuals from the fitted model. |
| fitted | Fitted values (one-step forecasts). |

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
```

```
n = 1015
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Test set
sim_test <- sim_data[1001:1010, ]

# Predictors taken as index variables
index.vars <- colnames(sim_data)[3:7]

# Assign group indices for each predictor
index.ind = c(rep(1, 3), rep(2, 2))

# Predictors taken as non-linear variables not entering indices
s.vars = "x_lag_005"

# Model fitting
gaimModel <- model_gaim(data = sim_train,
                        yvar = "y",
                        index.vars = index.vars,
                        index.ind = index.ind,
                        s.vars = s.vars)

forecast(gaimModel, newdata = sim_test)
```

---

`forecast.gamFit` *Forecasting using GAMs*

---

## Description

Returns forecasts and other information for GAMs.

## Usage

```
## S3 method for class 'gamFit'
forecast(
  object,
  h = 1,
  level = c(80, 95),
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | An object of class gamFit. Usually the result of a call to model_gam. |
| h | Forecast horizon. |
| level | Confidence level for prediction intervals. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. |
| ... | Other arguments not currently used. |

## Value

An object of class forecast. Here, it is a list containing the following elements:

| | |
|---|---|
| method | The name of the forecasting method as a character string. |
| model | The fitted model. |
| mean | Point forecasts as a time series. |
| residuals | Residuals from the fitted model. |
| fitted | Fitted values (one-step forecasts). |

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
```

```
n = 1015
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Test set
sim_test <- sim_data[1001:1010, ]

# Predictors taken as non-linear variables
s.vars <- colnames(sim_data)[3:6]

# Predictors taken as linear variables
linear.vars <- colnames(sim_data)[7:8]

# Model fitting
gamModel <- model_gam(data = sim_train,
                      yvar = "y",
                      s.vars = s.vars,
                      linear.vars = linear.vars)

forecast(gamModel, newdata = sim_test)
```

---

`forecast.pprFit` *Forecasting using PPR models*

---

## Description

Returns forecasts and other information for PPR models.

## Usage

```
## S3 method for class 'pprFit'
forecast(
  object,
  h = 1,
```

```
  level = c(80, 95),
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | An object of class pprFit. Usually the result of a call to model_ppr. |
| h | Forecast horizon. |
| level | Confidence level for prediction intervals. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. |
| ... | Other arguments not currently used. |

## Value

An object of class forecast. Here, it is a list containing the following elements:

| | |
|---|---|
| method | The name of the forecasting method as a character string. |
| model | The fitted model. |
| mean | Point forecasts as a time series. |
| residuals | Residuals from the fitted model. |
| fitted | Fitted values (one-step forecasts). |

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1015
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
```

```
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Test set
sim_test <- sim_data[1001:1010, ]

# Index variables
index.vars <- colnames(sim_data)[3:8]

# Model fitting
pprModel <- model_ppr(data = sim_train,
                      yvar = "y",
                      index.vars = index.vars)

forecast(pprModel, newdata = sim_test)
```

---

`forecast.smimodel` *Forecasting using SMI models*

---

## Description

Returns forecasts and other information for SMI models.

## Usage

```
## S3 method for class 'smimodel'
forecast(
  object,
  h = 1,
  level = c(80, 95),
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | An object of class smimodel. Usually the result of a call to model_smimodel. |
| h | Forecast horizon. |
| level | Confidence level for prediction intervals. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. |
| ... | Other arguments not currently used. |

## Value

An object of class forecast. Here, it is a list containing the following elements:

| | |
|---|---|
| method | The name of the forecasting method as a character string. |
| model | The fitted model. |
| mean | Point forecasts as a time series. |
| residuals | Residuals from the fitted model. |
| fitted | Fitted values (one-step forecasts). |

## Examples

```
if(requireNamespace("gurobi", quietly = TRUE)){
  library(dplyr)
  library(ROI)
  library(tibble)
  library(tidyr)
  library(tsibble)

  # Simulate data
  n = 1015
  set.seed(123)
  sim_data <- tibble(x_lag_000 = runif(n)) |>
    mutate(
      # Add x_lags
      x_lag = lag_matrix(x_lag_000, 5)) |>
    unpack(x_lag, names_sep = "_") |>
    mutate(
      # Response variable
      y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
      # Add an index to the data set
      inddd = seq(1, n)) |>
    drop_na() |>
```

```
    select(inddd, y, starts_with("x_lag")) |>
    # Make the data set a `tsibble`
    as_tsibble(index = inddd)

  # Training set
  sim_train <- sim_data[1:1000, ]
  # Test set
  sim_test <- sim_data[1001:1010, ]

  # Index variables
  index.vars <- colnames(sim_data)[3:8]

  # Model fitting
  smimodel_ppr <- model_smimodel(data = sim_train,
                                 yvar = "y",
                                 index.vars = index.vars,
                                 initialise = "ppr")

  forecast(smimodel_ppr, newdata = sim_test)
}
```

---

`greedy.fit` *Greedy search for tuning penalty parameters*

---

## Description

Function to perform a greedy search over a given grid of penalty parameter combinations (lambda0, lambda2), and fits a single SMI model with the best (lowest validation set MSE) penalty parameter combination. If the optimal combination lies on the edge of the grid, the penalty parameters are adjusted by ±10%, and a second round of grid search is performed.This is a helper function designed to be called from `greedy_smimodel`.

## Usage

```
greedy.fit(
  data,
  val.data,
  yvar,
  neighbour = 0,
  family = gaussian(),
  index.vars,
  initialise = c("ppr", "additive", "linear", "multiple", "userInput"),
  num_ind = 5,
  num_models = 5,
  seed = 123,
  index.ind = NULL,
  index.coefs = NULL,
  s.vars = NULL,
```

```
  linear.vars = NULL,
  nlambda = 100,
  lambda.min.ratio = 1e-04,
  refit = TRUE,
  M = 10,
  max.iter = 50,
  tol = 0.001,
  tolCoefs = 0.001,
  TimeLimit = Inf,
  MIPGap = 1e-04,
  NonConvex = -1,
  verbose = list(solver = FALSE, progress = FALSE),
  parallel = FALSE,
  workers = NULL,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | Training data set on which models will be trained. Must be a data set of class `tsibble`.(Make sure there are no additional date or time related variables except for the `index` of the `tsibble`). |
| `val.data` | Validation data set. (The data set on which the penalty parameter selection will be performed.) Must be a data set of class `tsibble`. (Once the penalty parameter selection is completed, the best model will be re-fitted for the combined data set `data` + `val.data`.) |
| `yvar` | Name of the response variable as a character string. |
| `neighbour` | `neighbour` argument passed from the outer function. |
| `family` | A description of the error distribution and link function to be used in the model (see `glm` and `family`). |
| `index.vars` | A `character` vector of names of the predictor variables for which indices should be estimated. |
| `initialise` | The model structure with which the estimation process should be initialised. The default is `"ppr"`, where the initial model is derived from projection pursuit regression. The other options are `"additive"` - nonparametric additive model, `"linear"` - linear regression model (i.e. a special case single-index model, where the initial values of the index coefficients are obtained through a linear regression), `"multiple"` - multiple models are fitted starting with different initial models (number of indices = `num_ind`; `num_models` random instances of the model (i.e. the predictor assignment to indices and initial index coefficients are generated randomly) are considered), and the final optimal model with the lowest loss is returned, and `"userInput"` - user specifies the initial model structure (i.e. the number of indices and the placement of index variables among indices) and the initial index coefficients through `index.ind` and `index.coefs` arguments respectively. |

num_ind — If initialise = "ppr" or "multiple": an integer that specifies the number of indices to be used in the model(s). The default is num_ind = 5.

num_models — If initialise = "multiple": an integer that specifies the number of starting models to be checked. The default is num_models = 5.

seed — If initialise = "multiple": the seed to be set when generating random starting points.

index.ind — If initialise = "userInput": an integer vector that assigns group index for each predictor in index.vars.

index.coefs — If initialise = "userInput": a numeric vector of index coefficients.

s.vars — A character vector of names of the predictor variables for which splines should be fitted individually (rather than considering as part of an index).

linear.vars — A character vector of names of the predictor variables that should be included linearly into the model.

nlambda — The number of values for lambda0 (penalty parameter for L0 penalty) - default is 100.

lambda.min.ratio — Smallest value for lambda0, as a fraction of lambda0.max (data derived).

refit — Whether to refit the model combining training and validation sets after parameter tuning. If FALSE, the final model will be estimated only on the training set.

M — Big-M value used in MIP.

max.iter — Maximum number of MIP iterations performed to update index coefficients for a given model.

tol — Tolerance for the objective function value (loss) of MIP.

tolCoefs — Tolerance for coefficients.

TimeLimit — A limit for the total time (in seconds) expended in a single MIP iteration.

MIPGap — Relative MIP optimality gap.

NonConvex — The strategy for handling non-convex quadratic objectives or non-convex quadratic constraints in Gurobi solver.

verbose — A named list controlling verbosity options. Defaults to list(solver = FALSE, progress = FALSE).

**solver** Logical. If TRUE, print detailed solver output.

**progress** Logical. If TRUE, print optimisation algorithm progress messages.

parallel — The option to use parallel processing in fitting SMI models for different penalty parameter combinations.

workers — If parallel = TRUE: Number of cores to use.

exclude.trunc — The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in val.data that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.)

recursive — Whether to obtain recursive forecasts or not (default - FALSE).

recursive_colRange
: If recursive = TRUE, the range of column numbers in val.data to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in val.data, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors.

## Value

A list that contains six elements:

| | |
|---|---|
| initial | A list of information of the model initialisation. (For descriptions of the list elements see make_smimodelFit). |
| best | A list of information of the final optimised model. (For descriptions of the list elements see make_smimodelFit). |
| best_lambdas | Selected penalty parameter combination. |
| lambda0_seq | Sequence of values for lambda0 used to construct the initial grid. |
| lambda2_seq | Sequence of values for lambda2 used to construct the initial grid. |
| searched | A tibble containing the penalty parameter combinations searched during the two-step greedy search and the corresponding validation set MSEs. |

---

# greedy_smimodel *SMI model estimation through a greedy search for penalty parameters*

---

## Description

Performs a greedy search over a given grid of penalty parameter combinations (lambda0, lambda2), and fits SMI model(s) with best (lowest validation set MSE) penalty parameter combination(s). If the optimal combination lies on the edge of the grid, the penalty parameters are adjusted by ±10%, and a second round of grid search is performed. If a grouping variable is used, penalty parameters are tuned separately for each individual model.

## Usage

```
greedy_smimodel(
  data,
  val.data,
  yvar,
  neighbour = 0,
  family = gaussian(),
  index.vars,
  initialise = c("ppr", "additive", "linear", "multiple", "userInput"),
  num_ind = 5,
  num_models = 5,
```

```
  seed = 123,
  index.ind = NULL,
  index.coefs = NULL,
  s.vars = NULL,
  linear.vars = NULL,
  nlambda = 100,
  lambda.min.ratio = 1e-04,
  refit = TRUE,
  M = 10,
  max.iter = 50,
  tol = 0.001,
  tolCoefs = 0.001,
  TimeLimit = Inf,
  MIPGap = 1e-04,
  NonConvex = -1,
  verbose = list(solver = FALSE, progress = FALSE),
  parallel = FALSE,
  workers = NULL,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL
)
```

## Arguments

| | |
|---|---|
| data | Training data set on which models will be trained. Must be a data set of class tsibble.(Make sure there are no additional date or time related variables except for the index of the tsibble). If multiple models are fitted, the grouping variable should be the key of the tsibble. If a key is not specified, a dummy key with only one level will be created. |
| val.data | Validation data set. (The data set on which the penalty parameter selection will be performed.) Must be a data set of class tsibble. (Once the penalty parameter selection is completed, the best model will be re-fitted for the combined data set data + val.data.) |
| yvar | Name of the response variable as a character string. |
| neighbour | If multiple models are fitted: Number of neighbours of each key (i.e. grouping variable) to be considered in model fitting to handle smoothing over the key. Should be an integer. If neighbour = x, x number of keys before the key of interest and x number of keys after the key of interest are grouped together for model fitting. The default is neighbour = 0 (i.e. no neighbours are considered for model fitting). |
| family | A description of the error distribution and link function to be used in the model (see glm and family). |
| index.vars | A character vector of names of the predictor variables for which indices should be estimated. |
| initialise | The model structure with which the estimation process should be initialised. The default is "ppr", where the initial model is derived from projection pursuit regression. The other options are "additive" - nonparametric additive model, |

"linear" - linear regression model (i.e. a special case single-index model, where the initial values of the index coefficients are obtained through a linear regression), "multiple" - multiple models are fitted starting with different initial models (number of indices = num_ind; num_models random instances of the model (i.e. the predictor assignment to indices and initial index coefficients are generated randomly) are considered), and the final optimal model with the lowest loss is returned, and "userInput" - user specifies the initial model structure (i.e. the number of indices and the placement of index variables among indices) and the initial index coefficients through index.ind and index.coefs arguments respectively.

num_ind — If initialise = "ppr" or "multiple": an integer that specifies the number of indices to be used in the model(s). The default is num_ind = 5.

num_models — If initialise = "multiple": an integer that specifies the number of starting models to be checked. The default is num_models = 5.

seed — If initialise = "multiple": the seed to be set when generating random starting points.

index.ind — If initialise = "userInput": an integer vector that assigns group index for each predictor in index.vars.

index.coefs — If initialise = "userInput": a numeric vector of index coefficients.

s.vars — A character vector of names of the predictor variables for which splines should be fitted individually (rather than considering as part of an index).

linear.vars — A character vector of names of the predictor variables that should be included linearly into the model.

nlambda — The number of values for lambda0 (penalty parameter for L0 penalty) - default is 100.

lambda.min.ratio — Smallest value for lambda0, as a fraction of lambda0.max (data derived).

refit — Whether to refit the model combining training and validation sets after parameter tuning. If FALSE, the final model will be estimated only on the training set.

M — Big-M value used in MIP.

max.iter — Maximum number of MIP iterations performed to update index coefficients for a given model.

tol — Tolerance for the objective function value (loss) of MIP.

tolCoefs — Tolerance for coefficients.

TimeLimit — A limit for the total time (in seconds) expended in a single MIP iteration.

MIPGap — Relative MIP optimality gap.

NonConvex — The strategy for handling non-convex quadratic objectives or non-convex quadratic constraints in Gurobi solver.

verbose — A named list controlling verbosity options. Defaults to list(solver = FALSE, progress = FALSE).

**solver** Logical. If TRUE, print detailed solver output.

**progress** Logical. If TRUE, print optimisation algorithm progress messages.

`parallel` The option to use parallel processing in fitting SMI models for different penalty parameter combinations.

`workers` If `parallel` = `TRUE`: Number of cores to use.

`exclude.trunc` The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in `val.data` that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.)

`recursive` Whether to obtain recursive forecasts or not (default - `FALSE`).

`recursive_colRange`
If `recursive` = `TRUE`, the range of column numbers in `val.data` to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. `lag_1`, `lag_2`, ..., `lag_m`, `lag_m` = maximum lag used) in `val.data`, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors.

## Value

An object of class `smimodel`. This is a `tibble` with two columns:

`key` The level of the grouping variable (i.e. key of the training data set).

`fit` Information of the fitted model corresponding to the `key`.

Each row of the column `fit` contains a list with six elements:

`initial` A list of information of the model initialisation. (For descriptions of the list elements see `make_smimodelFit`).

`best` A list of information of the final optimised model. (For descriptions of the list elements see `make_smimodelFit`).

`best_lambdas` Selected penalty parameter combination.

`lambda0_seq` Sequence of values for lambda0 used to construct the initial grid.

`lambda2_seq` Sequence of values for lambda2 used to construct the initial grid.

`searched` A `tibble` containing the penalty parameter combinations searched during the two-step greedy search and the corresponding validation set MSEs.

The number of rows of the `tibble` equals to the number of levels in the grouping variable.

## See Also

`model_smimodel`

## Examples

```
if(requireNamespace("gurobi", quietly = TRUE)){
  library(dplyr)
  library(ROI)
  library(tibble)
  library(tidyr)
  library(tsibble)

  # Simulate data
  n = 1205
  set.seed(123)
  sim_data <- tibble(x_lag_000 = runif(n)) |>
    mutate(
      # Add x_lags
      x_lag = lag_matrix(x_lag_000, 5)) |>
    unpack(x_lag, names_sep = "_") |>
    mutate(
      # Response variable
      y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
      # Add an index to the data set
      inddd = seq(1, n)) |>
    drop_na() |>
    select(inddd, y, starts_with("x_lag")) |>
    # Make the data set a `tsibble`
    as_tsibble(index = inddd)

  # Training set
  sim_train <- sim_data[1:1000, ]
  # Validation set
  sim_val <- sim_data[1001:1200, ]

  # Index variables
  index.vars <- colnames(sim_data)[3:8]

  # Model fitting
  smi_greedy <- greedy_smimodel(data = sim_train,
                                val.data = sim_val,
                                yvar = "y",
                                index.vars = index.vars,
                                initialise = "ppr",
                                lambda.min.ratio = 0.1)

  # Best (optimised) fitted model
  smi_greedy$fit[[1]]$best

  # Selected penalty parameter combination
  smi_greedy$fit[[1]]$best_lambdas
 }
```

`init_alpha` *Initialising index coefficients*

## Description

Initialises index coefficient vector through linear regression or penalised linear regression.

## Usage

```
init_alpha(
  Y,
  X,
  index.ind,
  init.type = "penalisedReg",
  lambda0 = 1,
  lambda2 = 1,
  M = 10
)
```

## Arguments

| | |
|---|---|
| `Y` | Column matrix of response. |
| `X` | Matrix of predictors entering indices. |
| `index.ind` | An `integer` vector that assigns group index for each predictor. |
| `init.type` | Type of initialisation for index coefficients. (`"penalisedReg"` - Penalised linear regression; `"reg"` - Linear regression) |
| `lambda0` | If `init.type` = `"penalisedReg"`, penalty parameter for L0 penalty. |
| `lambda2` | If `init.type` = `"penalisedReg"`, penalty parameter for L2 penalty. |
| `M` | If `init.type` = `"penalisedReg"`, the big-M value to be used in the MIP. |

## Value

A list containing the following components:

| | |
|---|---|
| `alpha_init` | Normalised vector of index coefficients. |
| `alpha_nonNormalised` | Non-normalised (i.e. prior to normalising) vector of index coefficients. |

inner_update *Updating index coefficients and non-linear functions iteratively*

## Description

Iteratively updates index coefficients and non-linear functions using mixed integer programming. (A helper function used within update_smimodelFit; users are not expected to directly call this function.)

## Usage

```
inner_update(
  x,
  data,
  yvar,
  family = gaussian(),
  index.vars,
  s.vars,
  linear.vars,
  num_ind,
  dgz,
  alpha_old,
  lambda0 = 1,
  lambda2 = 1,
  M = 10,
  max.iter = 50,
  tol = 0.001,
  TimeLimit = Inf,
  MIPGap = 1e-04,
  NonConvex = -1,
  verbose = list(solver = FALSE, progress = FALSE)
)
```

## Arguments

| | |
|---|---|
| x | Fitted gam. |
| data | Training data set on which models will be trained. Should be a tsibble. |
| yvar | Name of the response variable as a character string. |
| family | A description of the error distribution and link function to be used in the model (see glm and family). |
| index.vars | A character vector of names of the predictor variables for which indices should be estimated. |
| s.vars | A character vector of names of the predictor variables for which splines should be fitted individually (rather than considering as part of an index). |
| linear.vars | A character vector of names of the predictor variables that should be included linearly into the model. |

| | |
|---|---|
| num_ind | Number of indices. |
| dgz | The tibble of derivatives of the estimated smooths. |
| alpha_old | Current vector of index coefficients. |
| lambda0 | Penalty parameter for L0 penalty. |
| lambda2 | Penalty parameter for L2 penalty. |
| M | Big-M value to be used in MIP. |
| max.iter | Maximum number of MIP iterations performed to update index coefficients for a given model. |
| tol | Tolerance for loss. |
| TimeLimit | A limit for the total time (in seconds) expended in a single MIP iteration. |
| MIPGap | Relative MIP optimality gap. |
| NonConvex | The strategy for handling non-convex quadratic objectives or non-convex quadratic constraints in Gurobi solver. |
| verbose | A named list controlling verbosity options. Defaults to list(solver = FALSE, progress = FALSE).<br>**solver** Logical. If TRUE, print detailed solver output.<br>**progress** Logical. If TRUE, print optimisation algorithm progress messages. |

## Value

A list containing following elements:

| | |
|---|---|
| best_alpha | The vector of best index coefficient estimates. |
| min_loss | Minimum value of the objective function(loss). |
| index.ind | An integer vector that assigns group index for each predictor, corresponding to best_alpha. |
| ind_pos | A list that indicates which predictors belong to which index, corresponding to best_alpha. |
| X_new | A matrix of selected predictor variables, corresponding to best_alpha. |

---

# lag_matrix *Function for adding lags of time series variables*

---

## Description

Generates specified number of lagged variables of the given variable in the form of a tibble.

## Usage

```
lag_matrix(variable, n = 10)
```

### Arguments

| | |
|---|---|
| variable | Variable to be lagged. |
| n | Number of lags. The default value is n = 10. |

### Value

A tibble.

### Examples

```
library(dplyr)
library(tibble)
library(tidyr)
# Adding lagged variables to an existing tibble
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(100)) |>
  mutate(x_lag = lag_matrix(x_lag_000, 3)) |>
  unpack(x_lag, names_sep = "_")
```

---

## loss *Calculating the loss of the MIP used to estimate a SMI model*

---

### Description

Calculates the value of the objective function (loss function) of the mixed integer program used to estimate a SMI model.

### Usage

```
loss(Y, Yhat, alpha, lambda0, lambda2)
```

### Arguments

| | |
|---|---|
| Y | Column matrix of response. |
| Yhat | Predicted value of the response. |
| alpha | Vector of index coefficients. |
| lambda0 | Penalty parameter for L0 penalty. |
| lambda2 | Penalty parameter for L2 penalty. |

### Value

A numeric.

MAE *Point estimate accuracy measures*

## Description

Point estimate accuracy measures

## Usage

```
MAE(residuals, na.rm = TRUE, ...)

MSE(residuals, na.rm = TRUE, ...)

point_measures
```

## Arguments

| | |
|---|---|
| residuals | A vector of residuals from either the validation or test data. |
| na.rm | If TRUE, remove the missing values before calculating the accuracy measure. |
| ... | Additional arguments for each measure. |

## Format

An object of class list of length 2.

## Value

For the individual functions (MAE, MSE), returns a single numeric scalar giving the requested accuracy measure.

For the exported object point_measures, returns a named list of functions that can be supplied to higher-level accuracy routines.

## Examples

```
set.seed(123)
ytrain <- rnorm(100)
ytest  <- rnorm(30)
yhat   <- ytest + rnorm(30, sd = 0.3)
resid   <- ytest - yhat

MAE(resid)
MSE(resid)
```

make_smimodelFit *Converting a fitted* gam *object to a* smimodelFit *object*

## Description

Converts a given object of class gam to an object of class smimodelFit.

## Usage

```
make_smimodelFit(
  x,
  data,
  yvar,
  neighbour,
  index.vars,
  index.ind,
  index.data,
  index.names,
  alpha,
  s.vars = NULL,
  linear.vars = NULL,
  lambda0 = NULL,
  lambda2 = NULL,
  M = NULL,
  max.iter = NULL,
  tol = NULL,
  tolCoefs = NULL,
  TimeLimit = NULL,
  MIPGap = NULL,
  NonConvex = NULL
)
```

## Arguments

| | |
|---|---|
| x | A fitted gam object. |
| data | The original training data set. |
| yvar | Name of the response variable as a character string. |
| neighbour | neighbour argument passed from the outer function. |
| index.vars | A character vector of names of the predictor variables for which indices are estimated. |
| index.ind | An integer vector that assigns group index for each predictor in index.vars. |
| index.data | A tibble including columns for the constructed indices. |
| index.names | A character vector of names of the constructed indices. |
| alpha | A vector of index coefficients. |

| | |
|---|---|
| s.vars | A character vector of names of the predictor variables for which splines are fitted individually (rather than considering as part of an index). |
| linear.vars | A character vector of names of the predictor variables that are included linearly in the model. |
| lambda0 | Penalty parameter for L0 penalty. |
| lambda2 | Penalty parameter for L2 penalty. |
| M | Big-M value to be used in MIP. |
| max.iter | Maximum number of MIP iterations performed to update index coefficients for a given model. |
| tol | Tolerance for the objective function value (loss) of MIP. |
| tolCoefs | Tolerance for coefficients. |
| TimeLimit | A limit for the total time (in seconds) expended in a single MIP iteration. |
| MIPGap | Relative MIP optimality gap. |
| NonConvex | The strategy for handling non-convex quadratic objectives or non-convex quadratic constraints in Gurobi solver. |

## Value

An object of class smimodelFit, which is a list that contains following elements:

| | |
|---|---|
| alpha | A sparse matrix of index coefficients vectors. Each column of the matrix corresponds to the index coefficient vector of each index. |
| derivatives | A tibble of derivatives of the estimated smooths. |
| var_y | Name of the response variable. |
| vars_index | A character vector of names of the predictor variables for which indices are estimated. |
| vars_s | A character vector of names of the predictor variables for which splines are fitted individually. |
| vars_linear | A character vector of names of the predictor variables that are included linearly in the model. |
| neighbour | Number of neighbours of each key considered in model fitting. |
| gam | Fitted gam. |
| lambda0 | L0 penalty parameter used for model fitting. |
| lambda2 | L2 penalty parameter used for model fitting. |
| M | Big-M value used in MIP. |
| max.iter | Maximum number of MIP iterations for a single round of index coefficients update. |
| tol | Tolerance for the objective function value (loss) used in solving MIP. |
| tolCoefs | Tolerance for coefficients used in updating index coefficients. |
| TimeLimit | Limit for the total time (in seconds) expended in a single MIP iteration. |
| MIPGap | Relative MIP optimality gap used. |
| Nonconvex | The strategy used for handling non-convex quadratic objectives or non-convex quadratic constraints in Gurobi solver. |

model_backward *Nonparametric Additive Model with Backward Elimination*

## Description

Fits a nonparametric additive model, with simultaneous variable selection through a backward elimination procedure as proposed by Fan and Hyndman (2012).

## Usage

```
model_backward(
  data,
  val.data,
  yvar,
  neighbour = 0,
  family = gaussian(),
  s.vars = NULL,
  s.basedim = NULL,
  linear.vars = NULL,
  refit = TRUE,
  tol = 0.001,
  parallel = FALSE,
  workers = NULL,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| data | Training data set on which models will be trained. Must be a data set of class tsibble.(Make sure there are no additional date or time related variables except for the index of the tsibble). If multiple models are fitted, the grouping variable should be the key of the tsibble. If a key is not specified, a dummy key with only one level will be created. |
| val.data | Validation data set. (The data set on which the model selection will be performed.) Must be a data set of class tsibble. |
| yvar | Name of the response variable as a character string. |
| neighbour | If multiple models are fitted: Number of neighbours of each key (i.e. grouping variable) to be considered in model fitting to handle smoothing over the key. Should be an integer. If neighbour = x, x number of keys before the key of interest and x number of keys after the key of interest are grouped together for model fitting. The default is neighbour = 0 (i.e. no neighbours are considered for model fitting). |

| | |
|---|---|
| family | A description of the error distribution and link function to be used in the model (see glm and family). |
| s.vars | A character vector of names of the predictor variables for which splines should be fitted (i.e. non-linear predictors). |
| s.basedim | Dimension of the bases used to represent the smooth terms corresponding to s.vars. (For more information refer mgcv::s().) |
| linear.vars | A character vector of names of the predictor variables that should be included linearly into the model (i.e. linear predictors). |
| refit | Whether to refit the model combining training and validation sets after model selection. If FALSE, the final model will be estimated only on the training set. |
| tol | Tolerance for the ratio of relative change in validation set MSE, used in model selection. |
| parallel | Whether to use parallel computing in model selection or not. |
| workers | If parallel = TRUE, number of workers to use. |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in val.data that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.) |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in val.data to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in val.data, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors. |
| verbose | Logical; controls whether progress messages (model indices) are printed during fitting. Defaults to FALSE. |

## Details

This function fits a nonparametric additive model formulated through Backward Elimination, as proposed by Fan and Hyndman (2012). The process starts with all predictors included in an additive model, and predictors are progressively omitted until the best model is obtained based on the validation set. Once the best model is obtained, the final model is re-fitted for the data set combining training and validation sets. For more details see reference.

## Value

An object of class backward. This is a tibble with two columns:

| | |
|---|---|
| key | The level of the grouping variable (i.e. key of the training data set). |
| fit | Information of the fitted model corresponding to the key. |

Each row of the column fit is an object of class gam. For details refer mgcv::gamObject.

## References

Fan, S. & Hyndman, R.J. (2012). Short-Term Load Forecasting Based on a Semi-Parametric Additive Model. *IEEE Transactions on Power Systems*, 27(1), 134-141.doi:10.1109/TPWRS.2011.2162082.

## See Also

model_smimodel, model_gaim, model_ppr, model_gam, model_lm

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1205
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Validation set
sim_val <- sim_data[1001:1200, ]

# Predictors taken as non-linear variables
s.vars <- colnames(sim_data)[3:8]

# Model fitting
backwardModel <- model_backward(data = sim_train,
                                val.data = sim_val,
                                yvar = "y",
                                s.vars = s.vars)
# Fitted model
backwardModel$fit[[1]]
```

model_gaim *Groupwise Additive Index Models (GAIM)*

## Description

A wrapper for cgaim::cgaim() enabling multiple GAIM models based on a grouping variable. Currently does not support Constrained GAIM (CGAIM)s.

## Usage

```
model_gaim(
  data,
  yvar,
  neighbour = 0,
  index.vars,
  index.ind,
  s.vars = NULL,
  linear.vars = NULL,
  verbose = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| data | Training data set on which models will be trained. Must be a data set of class tsibble.(Make sure there are no additional date or time related variables except for the index of the tsibble). If multiple models are fitted, the grouping variable should be the key of the tsibble. If a key is not specified, a dummy key with only one level will be created. |
| yvar | Name of the response variable as a character string. |
| neighbour | If multiple models are fitted: Number of neighbours of each key (i.e. grouping variable) to be considered in model fitting to handle smoothing over the key. Should be an integer. If neighbour = x, x number of keys before the key of interest and x number of keys after the key of interest are grouped together for model fitting. The default is neighbour = 0 (i.e. no neighbours are considered for model fitting). |
| index.vars | A character vector of names of the predictor variables for which indices should be estimated. |
| index.ind | An integer vector that assigns group index for each predictor in index.vars. |
| s.vars | A character vector of names of the predictor variables for which splines should be fitted individually (rather than considering as part of an index). |
| linear.vars | A character vector of names of the predictor variables that should be included linearly into the model. |
| verbose | Logical; controls whether progress messages (model indices) are printed during fitting. Defaults to FALSE. |

... Other arguments not currently used. (Note that the arguments in `cgaim::cgaim()` related to constrained GAIMs are currently not supported. Furthermore, the argument `subset` is also not supported due to a bug in `cgaim::cgaim()`.)

## Details

Group-wise Additive Index Model (GAIM) can be written in the form

$$y_i = \sum_{j=1}^{p} g_j(\boldsymbol{\alpha}_j^T \boldsymbol{x}_{ij}) + \varepsilon_i, \quad i = 1, \ldots, n,$$

where $y_i$ is the univariate response, $\boldsymbol{x}_{ij} \in \mathbb{R}^{lj}$, $j = 1, \ldots, p$ are pre-specified non-overlapping subsets of $\boldsymbol{x}_i$, and $\boldsymbol{\alpha}_j$ are the corresponding index coefficients, $g_j$ is an unknown (possibly nonlinear) component function, and $\varepsilon_i$ is the random error, which is independent of $\boldsymbol{x}_i$.

## Value

An object of class `gaimFit`. This is a `tibble` with two columns:

| | |
|---|---|
| `key` | The level of the grouping variable (i.e. key of the training data set). |
| `fit` | Information of the fitted model corresponding to the `key`. |

Each row of the column `fit` is an object of class `cgaim`. For details refer `cgaim::cgaim()`.

## See Also

`model_smimodel`, `model_backward`, `model_ppr`, `model_gam`, `model_lm`

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1005
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
```

```
  as_tsibble(index = inddd)

# Predictors taken as index variables
index.vars <- colnames(sim_data)[3:7]

# Assign group indices for each predictor
index.ind = c(rep(1, 3), rep(2, 2))

# Predictors taken as non-linear variables not entering indices
s.vars = "x_lag_005"

# Model fitting
gaimModel <- model_gaim(data = sim_data,
                        yvar = "y",
                        index.vars = index.vars,
                        index.ind = index.ind,
                        s.vars = s.vars)
# Fitted model
gaimModel$fit[[1]]
```

---

`model_gam` *Generalised Additive Models*

---

## Description

A wrapper for `mgcv::gam()` enabling multiple GAMs based on a grouping variable.

## Usage

```
model_gam(
  data,
  yvar,
  family = gaussian(),
  neighbour = 0,
  s.vars,
  s.basedim = NULL,
  linear.vars = NULL,
  verbose = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | Training data set on which models will be trained. Must be a data set of class `tsibble`.(Make sure there are no additional date or time related variables except for the `index` of the `tsibble`). If multiple models are fitted, the grouping variable should be the `key` of the `tsibble`. If a `key` is not specified, a dummy key with only one level will be created. |

| | |
|---|---|
| yvar | Name of the response variable as a character string. |
| family | A description of the error distribution and link function to be used in the model (see glm and family). |
| neighbour | If multiple models are fitted: Number of neighbours of each key (i.e. grouping variable) to be considered in model fitting to handle smoothing over the key. Should be an integer. If neighbour = x, x number of keys before the key of interest and x number of keys after the key of interest are grouped together for model fitting. The default is neighbour = 0 (i.e. no neighbours are considered for model fitting). |
| s.vars | A character vector of names of the predictor variables for which splines should be fitted (i.e. non-linear predictors). |
| s.basedim | Dimension of the bases used to represent the smooth terms corresponding to s.vars. (For more information refer mgcv::s().) |
| linear.vars | A character vector of names of the predictor variables that should be included linearly into the model (i.e. linear predictors). |
| verbose | Logical; controls whether progress messages (model indices) are printed during fitting. Defaults to FALSE. |
| ... | Other arguments not currently used. |

## Value

An object of class gamFit. This is a tibble with two columns:

| | |
|---|---|
| key | The level of the grouping variable (i.e. key of the training data set). |
| fit | Information of the fitted model corresponding to the key. |

Each row of the column fit is an object of class gam. For details refer mgcv::gamObject.

## See Also

model_smimodel, model_backward, model_gaim, model_ppr, model_lm

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1005
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
```

```
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Predictors taken as non-linear variables
s.vars <- colnames(sim_data)[3:6]

# Predictors taken as linear variables
linear.vars <- colnames(sim_data)[7:8]

# Model fitting
gamModel <- model_gam(data = sim_data,
                      yvar = "y",
                      s.vars = s.vars,
                      linear.vars = linear.vars)

# Fitted model
gamModel$fit[[1]]
```

---

`model_lm` *Linear Regression models*

---

## Description

A wrapper for `lm` enabling multiple linear models based on a grouping variable.

## Usage

```
model_lm(data, yvar, neighbour = 0, linear.vars, verbose = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `data` | Training data set on which models will be trained. Must be a data set of class `tsibble`.(Make sure there are no additional date or time related variables except for the `index` of the `tsibble`). If multiple models are fitted, the grouping variable should be the `key` of the `tsibble`. If a `key` is not specified, a dummy key with only one level will be created. |
| `yvar` | Name of the response variable as a character string. |
| `neighbour` | If multiple models are fitted: Number of neighbours of each key (i.e. grouping variable) to be considered in model fitting to handle smoothing over the key. Should be an `integer`. If `neighbour = x`, `x` number of keys before the key of interest and `x` number of keys after the key of interest are grouped together for model fitting. The default is `neighbour = 0` (i.e. no neighbours are considered for model fitting). |

| | |
|---|---|
| linear.vars | A character vector of names of the predictor variables. |
| verbose | Logical; controls whether progress messages (model indices) are printed during fitting. Defaults to FALSE. |
| ... | Other arguments not currently used. |

## Value

An object of class lmFit. This is a tibble with two columns:

| | |
|---|---|
| key | The level of the grouping variable (i.e. key of the training data set). |
| fit | Information of the fitted model corresponding to the key. |

Each row of the column fit is an object of class lm. For details refer stats::lm.

## See Also

model_smimodel, model_backward, model_gaim, model_ppr, model_gam

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1005
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Predictor variables
linear.vars <- colnames(sim_data)[3:8]

# Model fitting
lmModel <- model_lm(data = sim_data,
                    yvar = "y",
                    linear.vars = linear.vars)
# Fitted model
lmModel$fit[[1]]
```

model_ppr *Projection Pursuit Regression (PPR) models*

## Description

A wrapper for `stats::ppr()` enabling multiple PPR models based on a grouping variable.

## Usage

```
model_ppr(
  data,
  yvar,
  neighbour = 0,
  index.vars,
  num_ind = 5,
  verbose = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | Training data set on which models will be trained. Must be a data set of class `tsibble`.(Make sure there are no additional date or time related variables except for the `index` of the `tsibble`). If multiple models are fitted, the grouping variable should be the `key` of the `tsibble`. If a `key` is not specified, a dummy key with only one level will be created. |
| `yvar` | Name of the response variable as a character string. |
| `neighbour` | If multiple models are fitted: Number of neighbours of each key (i.e. grouping variable) to be considered in model fitting to handle smoothing over the key. Should be an `integer`. If `neighbour = x`, `x` number of keys before the key of interest and `x` number of keys after the key of interest are grouped together for model fitting. The default is `neighbour = 0` (i.e. no neighbours are considered for model fitting). |
| `index.vars` | A `character` vector of names of the predictor variables for which indices should be estimated. |
| `num_ind` | An `integer` that specifies the number of indices to be used in the model(s). (Corresponds to `nterms` in `stats::ppr()`.) |
| `verbose` | Logical; controls whether progress messages (model indices) are printed during fitting. Defaults to FALSE. |
| `...` | Other arguments not currently used. (For more information on other arguments that can be passed, refer `stats::ppr()`.) |

## Details

A Projection Pursuit Regression (PPR) model (Friedman & Stuetzle (1981)) is given by

$$y_i = \sum_{j=1}^{p} g_j(\boldsymbol{\alpha}_j^T \boldsymbol{x}_i) + \varepsilon_i, \quad i = 1, \ldots, n,$$

where $y_i$ is the response, $\boldsymbol{x}_i$ is the $q$-dimensional predictor vector, $\boldsymbol{\alpha}_j = (\alpha_{j1}, \ldots, \alpha_{jp})^T$, $j = 1, \ldots, p$ are $q$-dimensional projection vectors (or vectors of "index coefficients"), $g_j$'s are unknown nonlinear functions, and $\varepsilon_i$ is the random error.

## Value

An object of class `pprFit`. This is a `tibble` with two columns:

| | |
|---|---|
| `key` | The level of the grouping variable (i.e. key of the training data set). |
| `fit` | Information of the fitted model corresponding to the `key`. |

Each row of the column `fit` is an object of class `c("ppr.form", "ppr")`. For details refer `stats::ppr()`.

## References

Friedman, J. H. & Stuetzle, W. (1981). Projection pursuit regression. *Journal of the American Statistical Association*, 76, 817–823. doi:10.2307/2287576.

## See Also

`model_smimodel`, `model_backward`, `model_gaim`, `model_gam`, `model_lm`

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1005
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
```

```
  as_tsibble(index = inddd)

# Index variables
index.vars <- colnames(sim_data)[3:8]

# Model fitting
pprModel <- model_ppr(data = sim_data,
                      yvar = "y",
                      index.vars = index.vars)

# Fitted model
pprModel$fit[[1]]
```

---

`model_smimodel` *Sparse Multiple Index (SMI) Models*

---

## Description

Fits nonparametric multiple index model(s), with simultaneous predictor selection (hence "sparse") and predictor grouping. Possible to fit multiple SMI models based on a grouping variable.

## Usage

```
model_smimodel(
  data,
  yvar,
  neighbour = 0,
  family = gaussian(),
  index.vars,
  initialise = c("ppr", "additive", "linear", "multiple", "userInput"),
  num_ind = 5,
  num_models = 5,
  seed = 123,
  index.ind = NULL,
  index.coefs = NULL,
  s.vars = NULL,
  linear.vars = NULL,
  lambda0 = 1,
  lambda2 = 1,
  M = 10,
  max.iter = 50,
  tol = 0.001,
  tolCoefs = 0.001,
  TimeLimit = Inf,
  MIPGap = 1e-04,
  NonConvex = -1,
  verbose = list(solver = FALSE, progress = FALSE)
)
```

## Arguments

| | |
|---|---|
| data | Training data set on which models will be trained. Must be a data set of class tsibble.(Make sure there are no additional date or time related variables except for the index of the tsibble). If multiple models are fitted, the grouping variable should be the key of the tsibble. If a key is not specified, a dummy key with only one level will be created. |
| yvar | Name of the response variable as a character string. |
| neighbour | If multiple models are fitted: Number of neighbours of each key (i.e. grouping variable) to be considered in model fitting to handle smoothing over the key. Should be an integer. If neighbour = x, x number of keys before the key of interest and x number of keys after the key of interest are grouped together for model fitting. The default is neighbour = 0 (i.e. no neighbours are considered for model fitting). |
| family | A description of the error distribution and link function to be used in the model (see glm and family). |
| index.vars | A character vector of names of the predictor variables for which indices should be estimated. |
| initialise | The model structure with which the estimation process should be initialised. The default is "ppr", where the initial model is derived from projection pursuit regression. The other options are "additive" - nonparametric additive model, "linear" - linear regression model (i.e. a special case single-index model, where the initial values of the index coefficients are obtained through a linear regression), "multiple" - multiple models are fitted starting with different initial models (number of indices = num_ind; num_models random instances of the model (i.e. the predictor assignment to indices and initial index coefficients are generated randomly) are considered), and the final optimal model with the lowest loss is returned, and "userInput" - user specifies the initial model structure (i.e. the number of indices and the placement of index variables among indices) and the initial index coefficients through index.ind and index.coefs arguments respectively. |
| num_ind | If initialise = "ppr" or "multiple": an integer that specifies the number of indices to be used in the model(s). The default is num_ind = 5. |
| num_models | If initialise = "multiple": an integer that specifies the number of starting models to be checked. The default is num_models = 5. |
| seed | If initialise = "multiple": the seed to be set when generating random starting points. |
| index.ind | If initialise = "userInput": an integer vector that assigns group index for each predictor in index.vars. |
| index.coefs | If initialise = "userInput": a numeric vector of index coefficients. |
| s.vars | A character vector of names of the predictor variables for which splines should be fitted individually (rather than considering as part of an index). |
| linear.vars | A character vector of names of the predictor variables that should be included linearly into the model. |
| lambda0 | Penalty parameter for L0 penalty. |

| | |
|---|---|
| `lambda2` | Penalty parameter for L2 penalty. |
| `M` | Big-M value to be used in MIP. |
| `max.iter` | Maximum number of MIP iterations performed to update index coefficients for a given model. |
| `tol` | Tolerance for the objective function value (loss) of MIP. |
| `tolCoefs` | Tolerance for coefficients. |
| `TimeLimit` | A limit for the total time (in seconds) expended in a single MIP iteration. |
| `MIPGap` | Relative MIP optimality gap. |
| `NonConvex` | The strategy for handling non-convex quadratic objectives or non-convex quadratic constraints in Gurobi solver. |
| `verbose` | A named list controlling verbosity options. Defaults to `list(solver = FALSE, progress = FALSE)`. |
| | **solver** Logical. If TRUE, print detailed solver output. |
| | **progress** Logical. If TRUE, print optimisation algorithm progress messages. |

## Details

Sparse Multiple Index (SMI) model is a semi-parametric model that can be written as

$$y_i = \beta_0 + \sum_{j=1}^{p} g_j(\boldsymbol{\alpha}_j^T \boldsymbol{x}_{ij}) + \sum_{k=1}^{d} f_k(w_{ik}) + \boldsymbol{\theta}^T \boldsymbol{u}_i + \varepsilon_i, \quad i = 1, \ldots, n,$$

where $y_i$ is the univariate response, $\beta_0$ is the model intercept, $\boldsymbol{x}_{ij} \in \mathbb{R}^{l_j}$, $j = 1, \ldots, p$ are $p$ subsets of predictors entering indices, $\boldsymbol{\alpha}_j$ is a vector of index coefficients corresponding to the index $h_{ij} = \boldsymbol{\alpha}_j^T \boldsymbol{x}_{ij}$, and $g_j$ is a smooth nonlinear function (estimated by a penalised cubic regression spline). The model also allows for predictors that do not enter any indices, including covariates $w_{ik}$ that relate to the response through nonlinear functions $f_k$, $k = 1, \ldots, d$, and linear covariates $\boldsymbol{u}_i$.

In the model formulation related to this implementation, both the number of indices $p$ and the predictor grouping among indices are assumed to be unknown prior to model estimation. Suppose we observe $y_1, \ldots, y_n$, along with a set of potential predictors, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, with each vector $\boldsymbol{x}_i$ containing $q$ predictors. This function implements algorithmic variable selection for index variables (i.e. predictors entering indices) of the SMI model by allowing for zero index coefficients for predictors. Non-overlapping predictors among indices are assumed (i.e. no predictor enters more than one index). For algorithmic details see reference.

## Value

An object of class `smimodel`. This is a `tibble` with two columns:

| | |
|---|---|
| `key` | The level of the grouping variable (i.e. key of the training data set). |
| `fit` | Information of the fitted model corresponding to the `key`. |

Each row of the column `fit` contains a list with two elements:

| | |
|---|---|
| `initial` | A list of information of the model initialisation. (For descriptions of the list elements see `make_smimodelFit`). |
| `best` | A list of information of the final optimised model. (For descriptions of the list elements see `make_smimodelFit`). |

## References

Palihawadana, N.K., Hyndman, R.J. & Wang, X. (2024). Sparse Multiple Index Models for High-Dimensional Nonparametric Forecasting. (Department of Econometrics and Business Statistics Working Paper Series 16/24).

## See Also

greedy_smimodel

## Examples

```
if(requireNamespace("gurobi", quietly = TRUE)){
  library(dplyr)
  library(ROI)
  library(tibble)
  library(tidyr)
  library(tsibble)

  # Simulate data
  n = 1005
  set.seed(123)
  sim_data <- tibble(x_lag_000 = runif(n)) |>
    mutate(
      # Add x_lags
      x_lag = lag_matrix(x_lag_000, 5)) |>
    unpack(x_lag, names_sep = "_") |>
    mutate(
      # Response variable
      y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
      # Add an index to the data set
      inddd = seq(1, n)) |>
    drop_na() |>
    select(inddd, y, starts_with("x_lag")) |>
    # Make the data set a `tsibble`
    as_tsibble(index = inddd)

  # Index variables
  index.vars <- colnames(sim_data)[3:8]

  # Model fitting
  smimodel_ppr <- model_smimodel(data = sim_data,
                                 yvar = "y",
                                 index.vars = index.vars,
                                 initialise = "ppr")

  # Best (optimised) fitted model
  smimodel_ppr$fit[[1]]$best
 }
```

new_smimodelFit *Constructor function for the class* smimodelFit

## Description

Constructs an object of class smimodelFit using the information passed to arguments.

## Usage

```
new_smimodelFit(
  data,
  yvar,
  neighbour = 0,
  family = gaussian(),
  index.vars,
  initialise = c("additive", "linear", "userInput"),
  index.ind = NULL,
  index.coefs = NULL,
  s.vars = NULL,
  linear.vars = NULL
)
```

## Arguments

| | |
|---|---|
| data | Training data set on which models will be trained. Must be a data set of class tsibble.(Make sure there are no additional date or time related variables except for the index of the tsibble). |
| yvar | Name of the response variable as a character string. |
| neighbour | neighbour argument passed from the outer function. |
| family | A description of the error distribution and link function to be used in the model (see glm and family). |
| index.vars | A character vector of names of the predictor variables for which indices should be estimated. |
| initialise | The model structure with which the estimation process should be initialised. The default is "additive", where the initial model will be a nonparametric additive model. The other options are "linear" - linear regression model (i.e. a special case single-index model, where the initial values of the index coefficients are obtained through a linear regression), and "userInput" - user specifies the initial model structure (i.e. the number of indices and the placement of index variables among indices) and the initial index coefficients through index.ind and index.coefs arguments respectively. |
| index.ind | If initialise = "userInput": an integer vector that assigns group index for each predictor in index.vars. |
| index.coefs | If initialise = "userInput": a numeric vector of index coefficients. |

| | |
|---|---|
| s.vars | A character vector of names of the predictor variables for which splines should be fitted individually (rather than considering as part of an index). |
| linear.vars | A character vector of names of the predictor variables that should be included linearly into the model. |

## Value

A list of initial model information. For descriptions of the list elements see make_smimodelFit).

---

# normalise_alpha *Scaling index coefficient vectors to have unit norm*

---

## Description

Scales a coefficient vector of a particular index to have unit norm.

## Usage

```
normalise_alpha(alpha)
```

## Arguments

| | |
|---|---|
| alpha | A vector of index coefficients. |

## Value

A numeric vector.

---

# possibleFutures_benchmark *Possible future sample paths (multi-step) from residuals of a fitted benchmark model*

---

## Description

Generates possible future sample paths (multi-step) using residuals of a fitted benchmark model through recursive forecasting.

## Usage

```
possibleFutures_benchmark(
  object,
  newdata,
  bootstraps,
  exclude.trunc = NULL,
  recursive_colRange
)
```

### Arguments

| | |
|---|---|
| object | A fitted model object of the class backward, pprFit, or gaimFit. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| bootstraps | Generated matrix of bootstrapped residual series. |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. |
| recursive_colRange | The range of column numbers in newdata to be filled with forecasts. |

### Value

A list containing the following components:

| | |
|---|---|
| firstFuture | A numeric vector of 1-step-ahead simulated futures. |
| future_cols | A list of multi-steps-ahead simulated futures, where each list element corresponds to each 1-step-ahead simulated future in firstFuture. |

---

## possibleFutures_smimodel

*Possible future sample paths (multi-step) from* smimodel *residuals*

---

### Description

Generates possible future sample paths (multi-step) using residuals of a fitted smimodel through recursive forecasting.

### Usage

```
possibleFutures_smimodel(
  object,
  newdata,
  bootstraps,
  exclude.trunc = NULL,
  recursive_colRange
)
```

### Arguments

| | |
|---|---|
| object | A smimodel object. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| bootstraps | Generated matrix of bootstrapped residual series. |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. |
| recursive_colRange | The range of column numbers in newdata to be filled with forecasts. |

## Value

A list containing the following components:

| | |
|---|---|
| `firstFuture` | A `numeric` vector of 1-step-ahead simulated futures. |
| `future_cols` | A list of multi-steps-ahead simulated futures, where each list element corresponds to each 1-step-ahead simulated future in `firstFuture`. |

---

`predict.backward` *Obtaining forecasts on a test set from a fitted* `backward`

---

## Description

Gives forecasts on a test set.

## Usage

```
## S3 method for class 'backward'
predict(
  object,
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A `backward` object. |
| `newdata` | The set of new data on for which the forecasts are required (i.e. test set; should be a `tsibble`). |
| `exclude.trunc` | The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in `newdata` that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.) |
| `recursive` | Whether to obtain recursive forecasts or not (default - `FALSE`). |
| `recursive_colRange` | If `recursive = TRUE`, the range of column numbers in `newdata` to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. `lag_1`, `lag_2`, ..., `lag_m`, `lag_m` = maximum lag used) in `newdata`, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors. |
| `...` | Other arguments not currently used. |

## Value

A tsibble with forecasts on test set.

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1215
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Validation set
sim_val <- sim_data[1001:1200, ]
# Test set
sim_test <- sim_data[1201:1210, ]

# Predictors taken as non-linear variables
s.vars <- colnames(sim_data)[3:8]

# Model fitting
backwardModel <- model_backward(data = sim_train,
                                val.data = sim_val,
                                yvar = "y",
                                s.vars = s.vars)
predict(object = backwardModel, newdata = sim_test)
```

---

predict.gaimFit *Obtaining forecasts on a test set from a fitted* gaimFit

---

## Description

Gives forecasts on a test set.

## Usage

```
## S3 method for class 'gaimFit'
predict(
  object,
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A gaimFit object. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in newdata that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.) |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in newdata, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors. |
| ... | Other arguments not currently used. |

## Value

A tsibble with forecasts on test set.

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)
```

```
# Simulate data
n = 1015
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Test set
sim_test <- sim_data[1001:1010, ]

# Predictors taken as index variables
index.vars <- colnames(sim_data)[3:7]

# Assign group indices for each predictor
index.ind = c(rep(1, 3), rep(2, 2))

# Predictors taken as non-linear variables not entering indices
s.vars = "x_lag_005"

# Model fitting
gaimModel <- model_gaim(data = sim_train,
                        yvar = "y",
                        index.vars = index.vars,
                        index.ind = index.ind,
                        s.vars = s.vars)

predict(object = gaimModel, newdata = sim_test)
```

---

`predict.gamFit` *Obtaining forecasts on a test set from a fitted* `gamFit`

---

## Description

Gives forecasts on a test set.

## Usage

```
## S3 method for class 'gamFit'
predict(
  object,
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A gamFit object. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in newdata that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.) |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in newdata, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors. |
| ... | Other arguments not currently used. |

## Value

A tsibble with forecasts on test set.

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1015
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
```

```
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Test set
sim_test <- sim_data[1001:1010, ]

# Predictors taken as non-linear variables
s.vars <- colnames(sim_data)[3:6]

# Predictors taken as linear variables
linear.vars <- colnames(sim_data)[7:8]

# Model fitting
gamModel <- model_gam(data = sim_train,
                      yvar = "y",
                      s.vars = s.vars,
                      linear.vars = linear.vars)

predict(object = gamModel, newdata = sim_test)
```

---

`predict.lmFit` *Obtaining forecasts on a test set from a fitted* `lmFit`

---

## Description

Gives forecasts on a test set.

## Usage

```
## S3 method for class 'lmFit'
predict(object, newdata, recursive = FALSE, recursive_colRange = NULL, ...)
```

## Arguments

| | |
|---|---|
| `object` | A `lmFit` object. |

newdata The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble).

recursive Whether to obtain recursive forecasts or not (default - FALSE).

recursive_colRange
If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in newdata, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors.

... Other arguments not currently used.

## Value

A tsibble with forecasts on test set.

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1015
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Test set
sim_test <- sim_data[1001:1010, ]

# Predictor variables
linear.vars <- colnames(sim_data)[3:8]

# Model fitting
```

```
lmModel <- model_lm(data = sim_train,
                    yvar = "y",
                    linear.vars = linear.vars)

predict(object = lmModel, newdata = sim_test)
```

---

predict.pprFit *Obtaining forecasts on a test set from a fitted* pprFit

---

## Description

Gives forecasts on a test set.

## Usage

```
## S3 method for class 'pprFit'
predict(
  object,
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A pprFit object. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in newdata that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.) |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in newdata, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors. |
| ... | Other arguments not currently used. |

## Value

A tsibble with forecasts on test set.

## Examples

```
library(dplyr)
library(tibble)
library(tidyr)
library(tsibble)

# Simulate data
n = 1015
set.seed(123)
sim_data <- tibble(x_lag_000 = runif(n)) |>
  mutate(
    # Add x_lags
    x_lag = lag_matrix(x_lag_000, 5)) |>
  unpack(x_lag, names_sep = "_") |>
  mutate(
    # Response variable
    y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Test set
sim_test <- sim_data[1001:1010, ]

# Index variables
index.vars <- colnames(sim_data)[3:8]

# Model fitting
pprModel <- model_ppr(data = sim_train,
                      yvar = "y",
                      index.vars = index.vars)

predict(object = pprModel, newdata = sim_test)
```

---

# predict.smimodel *Obtaining forecasts on a test set from a fitted* smimodel

---

## Description

Gives forecasts on a test set.

## Usage

```
## S3 method for class 'smimodel'
predict(
  object,
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | A smimodel object. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in newdata that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.) |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in newdata, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors. |
| ... | Other arguments not currently used. |

## Value

A tsibble with forecasts on test set.

## Examples

```
if(requireNamespace("gurobi", quietly = TRUE)){
  library(dplyr)
  library(ROI)
  library(tibble)
  library(tidyr)
  library(tsibble)

  # Simulate data
  n = 1015
```

```
  set.seed(123)
  sim_data <- tibble(x_lag_000 = runif(n)) |>
    mutate(
      # Add x_lags
      x_lag = lag_matrix(x_lag_000, 5)) |>
    unpack(x_lag, names_sep = "_") |>
    mutate(
      # Response variable
      y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
      # Add an index to the data set
      inddd = seq(1, n)) |>
    drop_na() |>
    select(inddd, y, starts_with("x_lag")) |>
    # Make the data set a `tsibble`
    as_tsibble(index = inddd)

  # Training set
  sim_train <- sim_data[1:1000, ]
  # Test set
  sim_test <- sim_data[1001:1010, ]

  # Index variables
  index.vars <- colnames(sim_data)[3:8]

  # Model fitting
  smimodel_ppr <- model_smimodel(data = sim_train,
                                 yvar = "y",
                                 index.vars = index.vars,
                                 initialise = "ppr")

  predict(object = smimodel_ppr, newdata = sim_test)
}
```

---

`predict.smimodelFit` *Obtaining forecasts on a test set from a* `smimodelFit`

---

## Description

Gives forecasts on a test set.

## Usage

```
## S3 method for class 'smimodelFit'
predict(
  object,
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
```

```
  ...
)
```

## Arguments

| | |
|---|---|
| object | A smimodelFit object. |
| newdata | The set of new data on for which the forecasts are required (i.e. test set; should be a tsibble). |
| exclude.trunc | The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in newdata that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.) |
| recursive | Whether to obtain recursive forecasts or not (default - FALSE). |
| recursive_colRange | If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in newdata, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors. |
| ... | Other arguments not currently used. |

## Value

A tibble with forecasts on test set.

## Examples

```
if(requireNamespace("gurobi", quietly = TRUE)){
  library(dplyr)
  library(ROI)
  library(tibble)
  library(tidyr)
  library(tsibble)

  # Simulate data
  n = 1015
  set.seed(123)
  sim_data <- tibble(x_lag_000 = runif(n)) |>
    mutate(
      # Add x_lags
      x_lag = lag_matrix(x_lag_000, 5)) |>
    unpack(x_lag, names_sep = "_") |>
    mutate(
      # Response variable
      y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
```

```
    # Add an index to the data set
    inddd = seq(1, n)) |>
  drop_na() |>
  select(inddd, y, starts_with("x_lag")) |>
  # Make the data set a `tsibble`
  as_tsibble(index = inddd)

# Training set
sim_train <- sim_data[1:1000, ]
# Test set
sim_test <- sim_data[1001:1010, ]

# Index variables
index.vars <- colnames(sim_data)[3:8]

# Model fitting
smimodel_ppr <- model_smimodel(data = sim_train,
                               yvar = "y",
                               index.vars = index.vars,
                               initialise = "ppr")

predict(object = smimodel_ppr$fit[[1]]$best, newdata = sim_test)
}
```

---

`predict_gam` *Obtaining recursive forecasts on a test set from a fitted* `mgcv::gam`

---

## Description

Gives recursive forecasts on a test set.

## Usage

```
predict_gam(
  object,
  newdata,
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A `gam` object. |
| `newdata` | The set of new data on for which the forecasts are required (i.e. test set; should be a `tsibble`). |

exclude.trunc The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in newdata that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.)

recursive Whether to obtain recursive forecasts or not (default - FALSE).

recursive_colRange
If recursive = TRUE, the range of column numbers in newdata to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in newdata, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors.

... Other arguments not currently used.

## Value

A tibble with forecasts on test set.

---

# prep_newdata *Prepare a data set for recursive forecasting*

---

## Description

Prepare a test data for recursive forecasting by appropriately removing existing (actual) values from a specified range of columns (lagged response columns) of the data set. Handles seasonal data with gaps.

## Usage

```
prep_newdata(newdata, recursive_colRange)
```

## Arguments

newdata Data set to be ared. Should be a tsibble.

recursive_colRange
The range of column numbers (lagged response columns) in newdata from which existing values should be removed. Make sure such columns are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in newdata, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors.

## Value

A tibble.

## print.backward *Printing a* backward *object*

### Description

The default print method for a backward object.

### Usage

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

### Arguments

| | |
|---|---|
| x | A model object of class backward as produced by model_backward() function. |
| ... | Other arguments not currently used. |

### Value

No return value; called for side effects. Prints a summary of the fitted model(s) to console.

## print.gaimFit *Printing a* gaimFit *object*

### Description

The default print method for a gaimFit object.

### Usage

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

### Arguments

| | |
|---|---|
| x | A model object of class gaimFit as produced by model_gaim() function. |
| ... | Other arguments not currently used. |

### Value

No return value; called for side effects. Prints a summary of the fitted model(s) to console.

## print.pprFit *Printing a* pprFit *object*

### Description

The default print method for a pprFit object.

### Usage

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

### Arguments

| | |
|---|---|
| x | A model object of class pprFit as produced by model_ppr() function. |
| ... | Other arguments not currently used. |

### Value

No return value; called for side effects. Prints a summary of the fitted model(s) to console.

## print.smimodel *Printing a* smimodel *object*

### Description

The default print method for a smimodel object.

### Usage

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

### Arguments

| | |
|---|---|
| x | An object of class smimodel. |
| ... | Other arguments not currently used. |

### Value

No return value; called for side effects. Prints a summary of the fitted model(s) to console.

## print.smimodelFit *Printing a* smimodelFit *object*

### Description

The default print method for a smimodelFit object.

### Usage

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

### Arguments

| | |
|---|---|
| x | An object of class smimodelFit. |
| ... | Other arguments not currently used. |

### Value

No return value; called for side effects. Prints a summary of the fitted model to console.

## randomBlock *Randomly sampling a block*

### Description

Samples a block of specified size from a given series starting form a random point in the series.

### Usage

```
randomBlock(series, block.size)
```

### Arguments

| | |
|---|---|
| series | A series from which a block should be sampled. |
| block.size | Size of the block to be sampled. |

### Value

A numeric vector.

## remove_lags *Remove actual values from a data set for recursive forecasting*

### Description

Appropriately removes existing (actual) values from the specified column range (lagged response columns) of a given data set (typically a test set for which recursive forecasting is required).

### Usage

```
remove_lags(data, recursive_colRange)
```

### Arguments

| | |
|---|---|
| data | Data set (a tibble) from which the actual lagged values should be removed. |
| recursive_colRange | The range of column numbers in data from which lagged values should be removed. |

### Value

A tibble.

## residBootstrap *Generate multiple single season block bootstrap series*

### Description

Generates multiple replications of single season block bootstrap series.

### Usage

```
residBootstrap(x, season.period = 1, m = 1, num.bootstrap = 1000)
```

### Arguments

| | |
|---|---|
| x | A series of residuals from which bootstrap series to be generated. |
| season.period | Length of the seasonal period. |
| m | Multiplier. (Block size = season.period * m) |
| num.bootstrap | Number of bootstrap series to be generated. |

### Value

A matrix of bootstrapped series.

`residuals.smimodel` *Extract residuals from a fitted* `smimodel`

## Description

Generates residuals from a fitted `smimodel` object.

## Usage

```
## S3 method for class 'smimodel'
residuals(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | A `smimodel` object. |
| `...` | Other arguments not currently used. |

## Value

A `numeric` vector of residuals.

## Examples

```
if(requireNamespace("gurobi", quietly = TRUE)){
  library(dplyr)
  library(ROI)
  library(tibble)
  library(tidyr)
  library(tsibble)

  # Simulate data
  n = 1005
  set.seed(123)
  sim_data <- tibble(x_lag_000 = runif(n)) |>
    mutate(
      # Add x_lags
      x_lag = lag_matrix(x_lag_000, 5)) |>
    unpack(x_lag, names_sep = "_") |>
    mutate(
      # Response variable
      y = (0.9*x_lag_000 + 0.6*x_lag_001 + 0.45*x_lag_003)^3 + rnorm(n, sd = 0.1),
      # Add an index to the data set
      inddd = seq(1, n)) |>
    drop_na() |>
    select(inddd, y, starts_with("x_lag")) |>
    # Make the data set a `tsibble`
    as_tsibble(index = inddd)

  # Index variables
```

```
  index.vars <- colnames(sim_data)[3:8]

  # Model fitting
  smimodel_ppr <- model_smimodel(data = sim_data,
                                 yvar = "y",
                                 index.vars = index.vars,
                                 initialise = "ppr")

  residuals(smimodel_ppr)
}
```

---

`scaling` *Scale data*

---

## Description

Scales the columns of the `data` corresponding to `index.vars`.

## Usage

```
scaling(data, index.vars)
```

## Arguments

| | |
|---|---|
| `data` | Training data set on which models will be trained. Should be a `tibble`. |
| `index.vars` | A character vector of names of the predictor variables for which indices should be estimated. |

## Value

A list containing the following components:

| | |
|---|---|
| `scaled_data` | The scaled data set of class `tibble`. |
| `scaled_info` | A named `numeric` vector of standard deviations of `index.vars` that were used to scale the corresponding columns of `data`. |

## seasonBootstrap *Single season block bootstrap*

### Description

Generates a single replication of single season block bootstrap series.

### Usage

```
seasonBootstrap(x, season.period = 1, m = 1)
```

### Arguments

| | |
|---|---|
| x | A series of residuals from which bootstrap series to be generated. |
| season.period | Length of the seasonal period. |
| m | Multiplier. (Block size = season.period * m) |

### Value

A numeric vector.

## smimodel.fit *SMI model estimation*

### Description

Fits a single nonparametric multiple index model to the data. This is a helper function designed to be called from user-facing wrapper functions, model_smimodel and greedy_smimodel.

### Usage

```
smimodel.fit(
  data,
  yvar,
  neighbour = 0,
  family = gaussian(),
  index.vars,
  initialise = c("ppr", "additive", "linear", "multiple", "userInput"),
  num_ind = 5,
  num_models = 5,
  seed = 123,
  index.ind = NULL,
  index.coefs = NULL,
  s.vars = NULL,
  linear.vars = NULL,
```

```
  lambda0 = 1,
  lambda2 = 1,
  M = 10,
  max.iter = 50,
  tol = 0.001,
  tolCoefs = 0.001,
  TimeLimit = Inf,
  MIPGap = 1e-04,
  NonConvex = -1,
  verbose = list(solver = FALSE, progress = FALSE)
)
```

## Arguments

| | |
|---|---|
| data | Training data set on which models will be trained. Must be a data set of class tsibble.(Make sure there are no additional date or time related variables except for the index of the tsibble). |
| yvar | Name of the response variable as a character string. |
| neighbour | neighbour argument passed from the outer function. |
| family | A description of the error distribution and link function to be used in the model (see glm and family). |
| index.vars | A character vector of names of the predictor variables for which indices should be estimated. |
| initialise | The model structure with which the estimation process should be initialised. The default is "ppr", where the initial model is derived from projection pursuit regression. The other options are "additive" - nonparametric additive model, "linear" - linear regression model (i.e. a special case single-index model, where the initial values of the index coefficients are obtained through a linear regression), "multiple" - multiple models are fitted starting with different initial models (number of indices = num_ind; num_models random instances of the model (i.e. the predictor assignment to indices and initial index coefficients are generated randomly) are considered), and the final optimal model with the lowest loss is returned, and "userInput" - user specifies the initial model structure (i.e. the number of indices and the placement of index variables among indices) and the initial index coefficients through index.ind and index.coefs arguments respectively. |
| num_ind | If initialise = "ppr" or "multiple": an integer that specifies the number of indices to be used in the model(s). The default is num_ind = 5. |
| num_models | If initialise = "multiple": an integer that specifies the number of starting models to be checked. The default is num_models = 5. |
| seed | If initialise = "multiple": the seed to be set when generating random starting points. |
| index.ind | If initialise = "userInput": an integer vector that assigns group index for each predictor in index.vars. |
| index.coefs | If initialise = "userInput": a numeric vector of index coefficients. |

| | |
|---|---|
| s.vars | A character vector of names of the predictor variables for which splines should be fitted individually (rather than considering as part of an index). |
| linear.vars | A character vector of names of the predictor variables that should be included linearly into the model. |
| lambda0 | Penalty parameter for L0 penalty. |
| lambda2 | Penalty parameter for L2 penalty. |
| M | Big-M value to be used in MIP. |
| max.iter | Maximum number of MIP iterations performed to update index coefficients for a given model. |
| tol | Tolerance for the objective function value (loss) of MIP. |
| tolCoefs | Tolerance for coefficients. |
| TimeLimit | A limit for the total time (in seconds) expended in a single MIP iteration. |
| MIPGap | Relative MIP optimality gap. |
| NonConvex | The strategy for handling non-convex quadratic objectives or non-convex quadratic constraints in Gurobi solver. |
| verbose | A named list controlling verbosity options. Defaults to list(solver = FALSE, progress = FALSE).<br>**solver** Logical. If TRUE, print detailed solver output.<br>**progress** Logical. If TRUE, print optimisation algorithm progress messages. |

## Value

A list with two elements:

| | |
|---|---|
| initial | A list of information of the model initialisation. (For descriptions of the list elements see make_smimodelFit). |
| best | A list of information of the final optimised model. (For descriptions of the list elements see make_smimodelFit). |

---

split_index *Splitting predictors into multiple indices*

---

## Description

Splits a given number of predictors into a given number of indices.

## Usage

```
split_index(num_pred, num_ind)
```

## Arguments

| | |
|---|---|
| num_pred | Number of predictors. |
| num_ind | Number of indices. |

### Value

A list containing the following components:

| | |
|---|---|
| index | An integer vector that assigns group indices for each predictor. |
| index_positions | A list of length = num_ind that indicates which predictors belong to which index. |

---

## truncate_vars *Truncating predictors to be in the in-sample range*

---

### Description

Truncates predictors to be in the in-sample range to avoid spline extrapolation.

### Usage

```
truncate_vars(range.object, data, cols.trunc)
```

### Arguments

| | |
|---|---|
| range.object | A matrix containing range of each predictor variable. Should be a matrix with two rows for min and max, and the columns should correspond to variables. |
| data | Out-of-sample data set of which variables should be truncated. |
| cols.trunc | Column names of the variables to be truncated. |

---

## tune_smimodel *SMI model with a given penalty parameter combination*

---

### Description

Fits a nonparametric multiple index model to the data for a given combination of the penalty parameters (lambda0, lambda2), and returns the validation set mean squared error (MSE). (Used within greedy.fit; users are not expected to use this function directly.)

### Usage

```
tune_smimodel(
  data,
  val.data,
  yvar,
  neighbour = 0,
  family = gaussian(),
  index.vars,
  initialise = c("ppr", "additive", "linear", "multiple", "userInput"),
  num_ind = 5,
```

```
  num_models = 5,
  seed = 123,
  index.ind = NULL,
  index.coefs = NULL,
  s.vars = NULL,
  linear.vars = NULL,
  lambda.comb = c(1, 1),
  M = 10,
  max.iter = 50,
  tol = 0.001,
  tolCoefs = 0.001,
  TimeLimit = Inf,
  MIPGap = 1e-04,
  NonConvex = -1,
  verbose = list(solver = FALSE, progress = FALSE),
  exclude.trunc = NULL,
  recursive = FALSE,
  recursive_colRange = NULL
)
```

## Arguments

| | |
|---|---|
| data | Training data set on which models will be trained. Must be a data set of class tsibble.(Make sure there are no additional date or time related variables except for the index of the tsibble). |
| val.data | Validation data set. (The data set on which the penalty parameter selection will be performed.) Must be a data set of class tsibble. (Once the penalty parameter selection is completed, the best model will be re-fitted for the combined data set data + val.data.) |
| yvar | Name of the response variable as a character string. |
| neighbour | neighbour argument passed from the outer function. |
| family | A description of the error distribution and link function to be used in the model (see glm and family). |
| index.vars | A character vector of names of the predictor variables for which indices should be estimated. |
| initialise | The model structure with which the estimation process should be initialised. The default is "ppr", where the initial model is derived from projection pursuit regression. The other options are "additive" - nonparametric additive model, "linear" - linear regression model (i.e. a special case single-index model, where the initial values of the index coefficients are obtained through a linear regression), "multiple" - multiple models are fitted starting with different initial models (number of indices = num_ind; num_models random instances of the model (i.e. the predictor assignment to indices and initial index coefficients are generated randomly) are considered), and the final optimal model with the lowest loss is returned, and "userInput" - user specifies the initial model structure (i.e. the number of indices and the placement of index variables among indices) and the initial index coefficients through index.ind and index.coefs arguments respectively. |

num_ind — If initialise = "ppr" or "multiple": an integer that specifies the number of indices to be used in the model(s). The default is num_ind = 5.

num_models — If initialise = "multiple": an integer that specifies the number of starting models to be checked. The default is num_models = 5.

seed — If initialise = "multiple": the seed to be set when generating random starting points.

index.ind — If initialise = "userInput": an integer vector that assigns group index for each predictor in index.vars.

index.coefs — If initialise = "userInput": a numeric vector of index coefficients.

s.vars — A character vector of names of the predictor variables for which splines should be fitted individually (rather than considering as part of an index).

linear.vars — A character vector of names of the predictor variables that should be included linearly into the model.

lambda.comb — A numeric vector (of length two) indicating the values for the two penalty parameters lambda0 and lambda2.

M — Big-M value used in MIP.

max.iter — Maximum number of MIP iterations performed to update index coefficients for a given model.

tol — Tolerance for the objective function value (loss) of MIP.

tolCoefs — Tolerance for coefficients.

TimeLimit — A limit for the total time (in seconds) expended in a single MIP iteration.

MIPGap — Relative MIP optimality gap.

NonConvex — The strategy for handling non-convex quadratic objectives or non-convex quadratic constraints in Gurobi solver.

verbose — A named list controlling verbosity options. Defaults to list(solver = FALSE, progress = FALSE).

**solver** Logical. If TRUE, print detailed solver output.

**progress** Logical. If TRUE, print optimisation algorithm progress messages.

exclude.trunc — The names of the predictor variables that should not be truncated for stable predictions as a character string. (Since the nonlinear functions are estimated using splines, extrapolation is not desirable. Hence, if any predictor variable in val.data that is treated non-linearly in the estimated model, will be truncated to be in the in-sample range before obtaining predictions. If any variables are listed here will be excluded from such truncation.)

recursive — Whether to obtain recursive forecasts or not (default - FALSE).

recursive_colRange — If recursive = TRUE, the range of column numbers in val.data to be filled with forecasts. Recursive/autoregressive forecasting is required when the lags of the response variable itself are used as predictor variables into the model. Make sure such lagged variables are positioned together in increasing lag order (i.e. lag_1, lag_2, ..., lag_m, lag_m = maximum lag used) in val.data, with no break in the lagged variable sequence even if some of the intermediate lags are not used as predictors.

### Value

A numeric.

---

## unscaling *Unscale a fitted* smimodel

---

### Description

Transforms back the index coefficients to suit original-scale index variables if the same were scaled when estimating the smimodel (happens in initialise = "ppr" in model_smimodel or greedy_smimodel). Users are not expected to directly use this function; usually called within smimodel.fit.

### Usage

```
unscaling(object, scaledInfo)
```

### Arguments

| | |
|---|---|
| object | A smimodel object. |
| scaledInfo | The list returned from a call of the function scaling. |

### Value

A smimodel object.

---

## update_alpha *Updating index coefficients using MIP*

---

### Description

Updates index coefficients by solving a mixed integer program.

### Usage

```
update_alpha(
  Y,
  X,
  num_pred,
  num_ind,
  index.ind,
  dgz,
  alpha_old,
  lambda0 = 1,
  lambda2 = 1,
  M = 10,
```

```
  TimeLimit = Inf,
  MIPGap = 1e-04,
  NonConvex = -1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| Y | Column matrix of response. |
| X | Matrix of predictors (size adjusted to number of indices). |
| num_pred | Number of predictors. |
| num_ind | Number of indices. |
| index.ind | An integer vector that assigns group index for each predictor. |
| dgz | The tibble of derivatives of the estimated smooths from previous iteration. |
| alpha_old | Vector of index coefficients from previous iteration. |
| lambda0 | Penalty parameter for L0 penalty. |
| lambda2 | Penalty parameter for L2 penalty. |
| M | Big-M value to be used in MIP. |
| TimeLimit | A limit for the total time (in seconds) expended in a single MIP iteration. |
| MIPGap | Relative MIP optimality gap. |
| NonConvex | The strategy for handling non-convex quadratic objectives or non-convex quadratic constraints in Gurobi solver. |
| verbose | The option to print detailed solver output. |

## Value

A vector of normalised index coefficients.

---

# update_smimodelFit *Updating a* smimodelFit

---

## Description

Optimises and updates a given smimodelFit.

## Usage

```
update_smimodelFit(
  object,
  data,
  lambda0 = 1,
  lambda2 = 1,
  M = 10,
```

```
  max.iter = 50,
  tol = 0.001,
  tolCoefs = 0.001,
  TimeLimit = Inf,
  MIPGap = 1e-04,
  NonConvex = -1,
  verbose = list(solver = FALSE, progress = FALSE),
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A `smimodelFit` object. |
| `data` | Training data set on which models will be trained. Must be a data set of class `tsibble`.(Make sure there are no additional date or time related variables except for the `index` of the `tsibble`). |
| `lambda0` | Penalty parameter for L0 penalty. |
| `lambda2` | Penalty parameter for L2 penalty. |
| `M` | Big-M value to be used in MIP. |
| `max.iter` | Maximum number of MIP iterations performed to update index coefficients for a given model. |
| `tol` | Tolerance for the objective function value (loss) of MIP. |
| `tolCoefs` | Tolerance for coefficients. |
| `TimeLimit` | A limit for the total time (in seconds) expended in a single MIP iteration. |
| `MIPGap` | Relative MIP optimality gap. |
| `NonConvex` | The strategy for handling non-convex quadratic objectives or non-convex quadratic constraints in Gurobi solver. |
| `verbose` | A named list controlling verbosity options. Defaults to `list(solver = FALSE, progress = FALSE)`.<br>**solver** Logical. If TRUE, print detailed solver output.<br>**progress** Logical. If TRUE, print optimisation algorithm progress messages. |
| `...` | Other arguments not currently used. |

## Value

A list of optimised model information. For descriptions of the list elements see `make_smimodelFit`).

# Index