

Package: hhh4addon (via r-universe)

October 8, 2026

Type Package

Title Extensions to endemic-epidemic timeseries modeling from package surveillance

Version 0.0.0.9014

Author Johannes Bracher [aut, cre], Maria Bekker-Nielsen Dunbar [ctb],
the authors and contributors of the surveillance package,
<https://cran.r-project.org/package=surveillance> [ctb]
(substantial parts of the package consist of modified code from
the surveillance package), R Core Team [ctb] (a few code
segments are modified versions of code from base R)

Maintainer Johannes Bracher <johannes.bracher@kit.edu>

Description Extending surveillance::hhh4 to allow for distributed
lags, solutions for longterm prediction and (periodically)
stationary moments.

License GPL-2

Encoding UTF-8

LazyData true

Depends R (>= 3.2.0), methods, grDevices, graphics, stats, utils, sp
(>= 1.0-15), surveillance

Imports fanplot, MASS, nlme, numDeriv

RoxygenNote 7.3.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

URL <https://github.com/jbracher/hhh4addon>

Repository <https://ee-lib.r-universe.dev>

Date/Publication 2026-07-24 14:05:38 UTC

RemoteUrl <https://github.com/jbracher/hhh4addon>

RemoteRef HEAD

RemoteSha e247993c6f7d3becf7f173f99a09e0307e215da2

Contents

aggregate_moments	3
ar2_lag	4
confint.oneStepAhead	5
decompose.hhh4	5
decompose.hhh4lag	5
dengueSJ	6
distr_lag	6
ds_score_hhh4	7
fanplot_prediction	8
fanplot_stationary	10
fit_par_lag	12
fixef.hhh4lag	14
geometric_lag	14
get_diags_of_array	15
get_weighted_lags	15
hhh4_lag	16
interpolate_qnbinom	18
is_complex_neighbourhood	19
is_fitted_par_lag	19
lambda_tilde	19
lambda_tilde_complex_neighbourhood	20
linear_lag	20
log_normal_lag	21
logLik.hhh4lag	22
matrix_is_cyclic	22
neOffsetArray.hhh4lag	23
norobL	23
numeric_fisher_hhh4lag	24
oneStepAhead_hhh4lag	24
plot.oneStepAhead	25
plot_moments_by_unit	25
poisson_lag	26
predictive_moments	27
print.hhh4lag	29
print.summary.hhh4lag	29
profile_par_lag	30
psi2size.hhh4lag	32
quantile.oneStepAhead	33
ranef.hhh4lag	33
residuals.hhh4lag	33
simulate.hhh4lag	34
stationary_moments	34
summary.hhh4lag	36
terms.hhh4lag	37
unrestricted_lag	37
update.hhh4lag	38

aggregate_moments	<i>Aggregation of stationary or predictive moments</i>
-------------------	--

Description

Aggregation of stationary or predictive moments as calculated using stationary_moments or predictive_moments.

Usage

```
aggregate_moments(momentsObj, aggregation_matrix, by_timepoint = FALSE)
```

Arguments

momentsObj	an object of class moments_hhh4 containing stationary or predictive moments, as returned by stationary_moments or predictive_moments
aggregation_matrix	an aggregation matrix with either momentsObj\$n_units columns (for aggregation across units while keeping the temporal structure; set option by_timepoint = TRUE in this case) or length(momentsObj\$mu_vector) (for aggregation that does not preserve the temporal structure; set option by_timepoint = FALSE).
by_timepoint	logical: is aggregation only across units while preserving the temporal structure? Note that the new moments_hhh4 object cannot have the condition, mu_matrix, var_matrix and cov_array elements if the temporal structure is given up.

Value

An object of class moments_hhh4 representing the new prediction.

Examples

```
# load data:
data("norobl")

#####
# fit a bivariate model:
controlBL <- list(end = list(f = addSeason2formula(~ -1 + fe(1, unitSpecific = TRUE))),
  ar = list(f = ~ -1 + fe(1, unitSpecific = TRUE)),
  ne = list(f = ~ -1 + fe(1, unitSpecific = TRUE)),
  family = "NegBinM", subset = 2:260) # not a very parsimonious parametrization, but feasible
fitBL <- hhh4(norobl, control = controlBL)
pred_mom <- predictive_moments(fitBL, t_condition = 260, lgt = 52, return_Sigma = TRUE)
# Sigma is required in order to aggregate predictions.

#####
# plot predictions for two regions:
par(mfrow = 1:2)
```

```

fanplot_prediction(pred_mom, unit = 1, main = "Bremen")
fanplot_prediction(pred_mom, unit = 2, main = "Lower Saxony")

#####
# aggregation 1: combine the two regions
aggr_matr_pool <- matrix(1, ncol = 2)
# specify by_timepoint = TRUE to keep the temporal structure and aggregate only
# counts from the same week:
pred_mom_pooled <- aggregate_moments(pred_mom, aggr_matr_pool, by_timepoint = TRUE)
fanplot_prediction(pred_mom_pooled, unit = 1, ylim = c(0, 500), main = "Aggregation over regions")

#####
# aggregation 2: total burden in the two regions
aggr_matr_total_burden <- matrix(rep(c(1, 0, 0, 1), 52), nrow = 2,
                                dimnames = list(c("Bremen", "Lower Saxony"),
                                                NULL))
pred_mom_total_burden <- aggregate_moments(pred_mom, aggr_matr_total_burden)
plot_moments_by_unit(pred_mom_total_burden, main = "Total burdens")

#####
# works also with stationary moments:
stat_mom <- stationary_moments(fitBL, return_Sigma = TRUE)
stat_mom_pooled <- aggregate_moments(stat_mom, aggr_matr_pool, by_timepoint = TRUE)
stat_mom_total_burden <- aggregate_moments(stat_mom, aggr_matr_total_burden, by_timepoint = FALSE)
fanplot_stationary(stat_mom_pooled)
plot_moments_by_unit(stat_mom_total_burden, main = "Total burdens")

```

ar2_lag

Function to obtain AR2 weights This function generates AR2 weights which are subsequently used inside of get_weighted_lags. To be passed to hhh4_lag or profile_par_lag as the control\$funct_lag argument.

Description

Function to obtain AR2 weights This function generates AR2 weights which are subsequently used inside of get_weighted_lags. To be passed to hhh4_lag or profile_par_lag as the control\$funct_lag argument.

Usage

```
ar2_lag(par_lag, min_lag, max_lag)
```

Arguments

par_lag a parameter to steer the lag structure, here $\text{logit}(p)$ where p is the weight of the first lag; see details of hhh4_lag or profile_par_lag.

min_lag	smallest lag to include; the support of the Poisson form starts only at min_lag. Defaults to 1.
max_lag	highest lag to include; higher lags are cut off and he remaining weights standardized. Defaults to 5.

confint.oneStepAhead	<i>confidence intervals for one-step-ahead predictions</i>
----------------------	--

Description

confidence intervals for one-step-ahead predictions

Usage

```
## S3 method for class 'oneStepAhead'
confint(object, parm, level = 0.95, ...)
```

decompose.hhh4	<i>A wrapper around</i>	decompose.hhh4lag	<i>and</i>
	surveillance::decompose.hhh4		

Description

A wrapper around decompose.hhh4lag and surveillance::decompose.hhh4 to handle ordinary hhh4 objects and objects of the new hhh4lag class.

Usage

```
decompose.hhh4(x, coefs = x$coefficients, ...)
```

decompose.hhh4lag	<i>A modified version of</i> decompose.hhh4
-------------------	---

Description

A modified version of decompose.hhh4 to deal with the added features of the hhh4lag class.

Usage

```
## S3 method for class 'hhh4lag'
decompose(x, coefs = x$coefficients, ...)
```

dengueSJ	<i>Data set on dengue in San Juan, Puerto Rico</i>
----------	--

Description

Case counts of dengue in San Juan, Puerto Rico, 1990-2013; stored as an sts object

Author(s)

Johannes Bracher

Source

Counts retrieved from the supplement of Ray et al (2017): Infectious disease prediction with kernel conditional density estimation, *Statistics in Medicine* 36(30):4908-4929. These data originally stem from a forecasting competition organized by the US federal government: <http://dengueforecasting.noaa.gov/>

distr_lag	<i>Display the function and parameters used for distributed lags</i>
-----------	--

Description

Display the function and parameters used for distributed lags

Usage

```
distr_lag(hhh4obj)
```

Arguments

hhh4obj an object of class hhh4

Value

A list containing the function and parameters characterizing the lag weights (for both the ar and ne components)

Examples

```
data("salmonella.agona")
## convert old "disProg" to new "sts" data class
salmonella <- disProg2sts(salmonella.agona)
# specify and fit model: fixed geometric lag structure
# with weight 0.8 for first lag
control_salmonella <- list(end = list(f = addSeason2formula(~ 1)),
                           ar = list(f = addSeason2formula(~ 1),
                                     par_lag = 0.8),
```

```

                                family = "NegBinM", subset = 6:312)
fit_salmonella <- hhh4_lag(salmonella, control_salmonella)
distr_lag(fit_salmonella)

```

ds_score_hhh4	<i>Calculate Dawid-Sebastiani score</i>
---------------	---

Description

Calculate Dawid-Sebastiani score for a prediction returned by `predictive_moments`.

Usage

```
ds_score_hhh4(pred, detailed = FALSE, scaled = TRUE)
```

Arguments

<code>pred</code>	the prediction as returned by <code>longterm_prediction_hhh4</code> (and potentially aggregated using <code>aggregate_prediction</code>)
<code>detailed</code>	detailed or less detailed output?
<code>scaled</code>	if <code>detailed == FALSE</code> : scale DSS with 2d?

Details

The Dawid-Sebastiani score is defined as

$$DSS = \log(|\Sigma|) + t(Y_{obs} - \mu)\Sigma^{-1}(Y_{obs} - \mu)$$

where μ and Σ are the predictive mean and variance, respectively. `Y_obs` represents the observation that has materialized.

Value

If `detailed == FALSE`: the (potentially scaled) Dawid-Sebastiani score. If `detailed == TRUE`: a vector containing the following elements:

- `dawid_sebastiani` the un-scaled Dawid-Sebastiani score
- `term1` value of the log-determinant entering into the un-scaled Dawid-Sebastiani score
- `term2` value of the quadratic form entering into the un-scaled Dawid-Sebastiani score
- `scaled_dawid_sebastiani` the scaled Dawid-Sebastiani score
- `determinant_sharpness` the determinant sharpness (scaled version of `term1`)

Examples

```
## a simple univariate example:
data("salmonella.agona")
## convert old "disProg" to new "sts" data class
salmonella <- disProg2sts(salmonella.agona)
# specify and fit model: fixed geometric lag structure
# with weight 0.8 for first lag
control_salmonella <- list(end = list(f = addSeason2formula(~ 1)),
                           ar = list(f = addSeason2formula(~ 1),
                                       par_lag = 0.8, use_distr_lag = TRUE),
                           family = "NegBinM", subset = 6:312)
fit_salmonella <- hhh4_lag(salmonella, control_salmonella)
pred_salmonella <- predictive_moments(fit_salmonella, t_condition = 260,
                                     52, return_Sigma = TRUE)
ds_score_hhh4(pred_salmonella, detailed = TRUE)
```

fanplot_prediction	<i>Display prediction as a fan plot</i>
--------------------	---

Description

Plots a fanplot to display quantiles of (negative binomial approximations) of the week-wise predictive distributions

Usage

```
fanplot_prediction(
  pred,
  unit = 1,
  probs = 1:99/100,
  interpolate_probs = TRUE,
  add_observed = TRUE,
  add_pred_means = TRUE,
  fan.col = colorRampPalette(c("darkgreen", "gray90")),
  pt.col = "red",
  pt.cex = 0.6,
  l.col = "black",
  mean.col = "black",
  mean.lty = "dashed",
  ln = NULL,
  rlab = NULL,
  add = FALSE,
  add_legend = FALSE,
  width_legend = 0.1 * (max(pred$timepoints) - min(pred$timepoints))/pred$freq,
  probs_legend = c(1, 25, 50, 75, 99)/100,
  ylim = NULL,
  xlab = "t",
```

```

    ylab = "No. infected",
    return_matrix = FALSE,
    ...
  )

```

Arguments

pred	the prediction as returned by <code>longterm_prediction_hhh4</code> (and potentially aggregated using <code>aggregate_prediction</code>)
unit	numeric denoting the unit to display
probs	vector of probabilities: which quantiles shall be displayed in the fan plot?
interpolate_probs	logical: smooth curves by simple interpolation of quantiles
add_observed	logical: shall observed values be added?
fan.col, ln, rlab	graphical parameters passed on to <code>fanplot::fan</code>
pt.col, pt.cex, l.col	graphical parameters for display of observed values
add	logical: add to existing plot?
add_legend	logical: shall a color key legend be added?
width_legend	width of box for color key legend in user coordinates
probs_legend	vector of probabilities to display in the legend
ylim	limit for the y-axis, passed to <code>plot()</code>
xlab, ylab	axis labels
return_matrix	logical: return matrix passed to <code>fanplot::fan</code> ; useful to make more sophisticated plots.
...	other arguments passed on to <code>plot()</code>

Value

Only if `return_matrix` set to `TRUE`: the matrix passed to `fanplot::fan`

Examples

```

data("salmonella.agona")
# convert old "disProg" to new "sts" data class:
salmonella <- disProg2sts(salmonella.agona)
control_salmonella <- list(end = list(f = addSeason2formula(~ 1), lag = 1),
                           ar = list(f = addSeason2formula(~ 1), lag = 1),
                           family = "NegBinM", subset = 6:250)
fit_salmonella <- hhh4_lag(salmonella, control_salmonella) # fit model
# obtain prediction:
pred_mom <- predictive_moments(fit_salmonella, t_condition = 250, lgt = 52)
# plot the prediction only:
fanplot_prediction(pred_mom, add_legend = TRUE)
# or plot it along with the fit:

```

```
plot(fit_salmonella)
fanplot_prediction(pred_mom, add = TRUE) # add fan plot
```

fanplot_stationary	<i>Display stationary distribution as a fanplot</i>
--------------------	---

Description

Plots a fanplot to display quantiles of (negative binomial approximations) of the week-wise stationary distributions

Usage

```
fanplot_stationary(
  stat_mom,
  unit = 1,
  probs = 1:99/100,
  interpolate_probs = TRUE,
  add_pred_means = TRUE,
  fan.col = colorRampPalette(c("darkgreen", "gray90")),
  pt.col = "red",
  pt.cex = 0.3,
  l.col = "black",
  mean_col = "black",
  mean_lty = "dashed",
  ln = NULL,
  ln.col = "red",
  rlab = NULL,
  style = "fan",
  add = FALSE,
  timepoints = 1:nrow(stat_mom$mu_matrix)/stat_mom$freq,
  add_legend = FALSE,
  width_legend = 0.1 * (max(timepoints) - min(timepoints)),
  probs_legend = c(1, 25, 50, 75, 99)/100,
  hlines = NULL,
  vlines = NULL,
  ylim = NULL,
  xlab = "t",
  ylab = "No. infected",
  return_matrix = FALSE,
  ...
)
```

Arguments

stat_mom	the stationary moments as returned by stationary_moments_hhh4 (and potentially aggregated using aggregate_prediction)
----------	---

unit	numeric denoting the unit to display
probs	vector of probabilities: which quantiles shall be displayed in the fan plot?
interpolate_probs	logical: smooth curves by simple interpolation of quantiles
add_pred_means	logical: add line showing the the predictive means
fan.col, ln, ln.col, rlab, style	graphical parameters passed on to fanplot::fan
pt.col, pt.cex, l.col	graphical parameters for display of observed values
add	logical: add to existing plot?
timepoints	vector giving the x-coordinates for the fanplot (generates start and frequency for fanplot::fan)
add_legend	logical: shall a color key legend be added?
width_legend	width of box for color key legend in user coordinates
probs_legend	vecor of probabilities to display in the legend
hlines, vlines	coordinates for horizontal and vertical grid lines
ylim	limit for the y-axis, passed to plot()
xlab, ylab	axis labels
return_matrix	logical: return matrix passed to fanplot::fan; useful to make more sophisticated plots.
...	other arguments passed on to plot()
means_col, mean_lty	graphical parameters for display of predictive means

Value

Only if return_matrix set to TRUE: the matrix passed to fanplot::fan

Examples

```
data("salmonella.agona")
## convert old "disProg" to new "sts" data class
salmonella <- disProg2sts(salmonella.agona)
# specify and fit model
control_salmonella <- list(end = list(f = addSeason2formula(~ 1), lag = 1),
                           ar = list(f = addSeason2formula(~ 1), lag = 1),
                           family = "NegBinM")
fit_salmonella <- hhh4(salmonella, control_salmonella)
# obtain periodically stationary moments:
stat_mom <- stationary_moments(fit_salmonella)
# plot periodically stationary means:
fanplot_stationary(stat_mom, add_legend = TRUE)
# add paths of the six seasons in the data set:
for(i in 0:5){
  lines(1:52/52, salmonella@observed[(i*52 + 1):((i + 1)*52)], col = "blue")
}
legend("topleft", col = "blue", lty = 1, legend = "observed seasons")
```

fit_par_lag	<i>Estimating the lag decay parameter of an hhh4_lag model using profile likelihood</i>
-------------	---

Description

Wrapper around hhh4_lag to allow for profile likelihood estimation of the scalar parameter governing the lag structure. hhh4_lag can fit models with fixed lag decay parameter; fit_par_lag loops around it and tries a set of possible parameters provided in the argument range_par. NOTE: this will soon be replaced by profile_par_lag which does the same, but using optim..., method = "Brent", ...).

Usage

```
fit_par_lag(
  stsObj,
  control,
  check.analyticals = FALSE,
  range_par,
  use_update = TRUE
)
```

Arguments

range_par	a vector of values to try for the par_lag argument of funct_lag
use_update	should results from previous values in range_par be used as starting value for next iteration (via update)?

Details

In this modified version of surveillance::hhh4, distributed lags can be specified by additional elements control argument:

- funct_lag Function to compute the lag weights u_q (see details) depending on a scalar parameter par_lag. The function has to take the following arguments:
 - par_lag A scalar parameter to steer u_q . It should be specified in a way which allows it to take any value in the real numbers
 - min_lag, max_lag Minimum and maximum lags; e.g. min_lag = 3, max_lag = 6 will assign all weights to lags 3 through 6. Usually min_lag is set to 1, higher values can be useful for direct forecasting at higher horizons. max_lag defaults to 5, which is often reasonable for weekly data, but should likely be increased when using daily data.
- min_lag, max_lag Specification of the arguments passed to funct_lag to compute the distributed lags. Unlike in hhh4_lag, par_lag is not to be specified as it is estimated from the data. Important: the first element of the subset argument in control needs to be larger than max_lag (as for the first max_lag observations the fitted values cannot be computed)

Unlike in `hhh4_lag` the `par_lag` argument for `funct_lag` is not specified directly by the user; instead the model is re-fit for each parameter value provided in `range_par`.

#' @paramstsObj,control,check.analyticals As in `surveillance::hhh4`, but `control` allows for some additional elements in order to specify a distributed lag structure:

- `funct_lag` Function to compute the lag weights u_q (see details) depending on a scalar parameter `par_lag`. The function has to take the following arguments:
 - `par_lag` A scalar parameter to steer u_q . It should be specified in a way which allows it to take any value in the real numbers
 - `min_lag`, `max_lag` Minimum and maximum lags; e.g. `min_lag = 3`, `max_lag = 6` will assign all weights to lags 3 through 6. Usually `min_lag` is set to 1, higher values can be useful for direct forecasting at higher horizons.
- `min_lag`, `max_lag` Specification of the arguments passed to `funct_lag` to compute the distributed lags. Unlike in `hhh4_lag`, `par_lag` is not to be specified as it is estimated from the data.

Value

A list including the best model among all fitted ones (`best_mod`) and a vector of the AIC values obtained for the different values provided in `range_par` (AICs)

See Also

`hhh4_lag` for fitting models with fixed `par_lag`; `profile_par_lag` for optimization using `optim` rather than a vector `range_par` of potential values.

Examples

```
## a simple univariate example:
data("salmonella.agona")
## convert old "disProg" to new "sts" data class
salmonella <- disProg2sts(salmonella.agona)
# specify and fit model: fixed geometric lag structure
control_salmonella <- list(end = list(f = addSeason2formula(~ 1)),
                           ar = list(f = addSeason2formula(~ 1)),
                           family = "NegBinM", subset = 6:312)
# get a reasonable range of values for par_lag. par_lag is logit(p) in teh
# geometric lag function
grid_p <- seq(from = 0.01, to = 0.99, by = 0.02)
grid_par_lag <- log(grid_p/(1 - grid_p))
fit_salmonella <- fit_par_lag(salmonella, control_salmonella, range_par = grid_par_lag)
summary(fit_salmonella$best_mod)
plot(fit_salmonella$AICs, xlab = "p", ylab = "AIC")
# 0.56 on first lag
#
# re-fit with Poisson lags:
control_salmonella2 <- control_salmonella
control_salmonella2$funct_lag = poisson_lag
grid_p2 <- seq(from = 0.01, to = 2, by = 0.02)
grid_par_lag2 <- log(grid_p2)
```

```
fit_salmonella2 <- fit_par_lag(salmonella, control_salmonella2, range_par = grid_par_lag2)
summary(fit_salmonella2$best_mod)
# leads to somewhat different decay and very slightly better AIC
```

fixef.hhh4lag	<i>A modified version of fixef.hhh4</i>
---------------	---

Description

A modified version of fixef.hhh4

Usage

```
## S3 method for class 'hhh4lag'
fixef(object, ...)
```

geometric_lag	<i>Function to obtain geometric weights This function generates geometric weights which are subsequently used inside of get_weighted_lags. To be passed to hhh4_lag or profile_par_lag as the control\$funct_lag argument.</i>
---------------	--

Description

Function to obtain geometric weights This function generates geometric weights which are subsequently used inside of get_weighted_lags. To be passed to hhh4_lag or profile_par_lag as the control\$funct_lag argument.

Usage

```
geometric_lag(par_lag, min_lag, max_lag)
```

Arguments

par_lag	a parameter to steer the lag structure, here $\text{logit}(p)$ where p is the parameter of the geometric distribution characterizing the lag structure; see details of hhh4lag or profile_par_lag.
min_lag	smallest lag to include; the support of the Poisson form starts only at min_lag. Defaults to 1.
max_lag	highest lag to include; higher lags are cut off and the remaining weights standardized. Defaults to 5.

get_diags_of_array	<i>Get diagonal elements of all slices of an array</i>
--------------------	--

Description

Extracts diagonals of all slices of an array (i.e. of `arr[:,1]`, `arr[:,2]`, ... and stacks them in one vector.)

Usage

```
get_diags_of_array(arr)
```

Arguments

arr	An array.
-----	-----------

get_weighted_lags	<i>Transform matrix of first-order lagged observations to matrix of weighted sums of past observation</i>
-------------------	---

Description

This function modifies the design matrices from first-order lags to weighted lags with weighted structure. Used within `weightedSumNE` and `weightedSumAR`.

Usage

```
get_weighted_lags(lag1, lag_weights, sum_up = FALSE)
```

Arguments

lag1	a matrix of first lags as usually used in <code>hhh4</code> .
sum_up	<code>sum_up = FALSE</code> returns a more detailed output; for debugging only.
lag_weights	a vector of weights; the length of this vector determines the number of lags.

hhh4_lag

*Fitting hhh4 models with distributed lags***Description**

A modified version of `surveillance::hhh4` to allow for distributed lags. Usually used from inside of the wrappers `profile_par_lag` or `fit_par_lag`.

Usage

```
hhh4_lag(
  stsObj,
  control = list(ar = list(f = ~-1, offset = 1, lag = NA), ne = list(f = ~-1, offset = 1,
    lag = NA, weights = neighbourhood(stsObj) == 1, scale = NULL, normalize = FALSE), end
    = list(f = ~1, offset = 1), family = c("Poisson", "NegBin1", "NegBinM"), funct_lag =
    geometric_lag, par_lag = 1, min_lag = 1, max_lag = 5, subset = 6:nrow(stsObj),
    optimizer = list(stop = list(tol = 1e-05, niter = 100), regression = list(method =
    "nlminb"), variance = list(method = "nlminb")), verbose = FALSE, start = list(fixed =
    NULL,
    random = NULL, sd.corr = NULL), data = list(t = stsObj@epoch -
    min(stsObj@epoch)), keep.terms = FALSE),
  check.analyticals = FALSE
)
```

Arguments

`stsObj`, `control`, `check.analyticals`

As in `surveillance::hhh4`, but with the following additional elements in the `control` argument in order to specify a distributed lag structure:

- `funct_lag` Function to compute the lag weights u_q (see details) depending on a scalar parameter `par_lag`. The function has to take the following arguments:
 - `par_lag` A scalar parameter to steer u_q . It should be specified in a way which allows it to take any value in the real numbers
 - `min_lag`, `max_lag` Minimum and maximum lags; e.g. `min_lag = 3`, `max_lag = 6` will assign all weights to lags 3 through 6. Usually `min_lag` is set to 1, higher values can be useful for direct forecasting at higher horizons.
- `par_lag`, `min_lag`, `max_lag` Specification of the arguments passed to `funct_lag` to compute the distributed lags. Important: the first element of the `subset` argument in `control` needs to be larger than `max_lag` (as for the first `max_lag` observations the fitted values cannot be computed)

`hhh4_lag` requires `par_lag` to be pre-specified (with a default of 1). Using the wrappers `profile_par_lag` and `fit_par_lag` it can also be estimated using a profile likelihood approach.

Details

The standard hhh4 function only allows for models with first lags i.e. of the form

$$mu_{it} = \lambda_{it} X_{i,t-1} + \phi_{it} \sum_{j \neq i} w_{ji} X_{j,t-1} + \nu_{it},$$

see ?hhh4. The extension hhh4_lag allows to specify models of the form

$$mu_{it} = \lambda_{it} \sum_{q=1}^Q u_q X_{i,t-q} + \phi_{it} \sum_{j \neq i} \sum_{q=1}^Q w_{ji} u_q X_{j,t-q} + \nu_{it}.$$

Here the first lags are now replaced by weighted sums of the Q previous observations. The weights u_q , $q = 1, \dots, Q$ sum up to 1 and need to be parametrizable by a single scalar parameter. The value of this parameter needs to be passed as `control$par_lag`. Moreover, a function to obtain a vector of weights from `par_lag` needs to be provided in `control$funct_lag`. Currently four such functions are implemented in the package:

- Geometric lags (function `geometric_lag`; the default). These are specified as

$$u0_q = \alpha * (1 - \alpha)^{q-1}$$

and $u_q = u0_q / \sum_{q=1}^Q u0_q$ for $q = 1, \dots, Q$. The `par_lag` parameter corresponds to $\text{logit}(\alpha)$, i.e. the un-normalized weight of the first lag.

- Poisson lags (function `poisson_lag`). These are specified as

$$u0_q = \alpha^q (q-1)! \exp(-\alpha) / (q-1)!,$$

and $u_q = u0_q / \sum_{q=1}^Q u0_q$ for $q = 1, \dots, Q$. Note that the Poisson distribution is shifted by one to achieve a positive support. The `par_lag` parameter corresponds to $\log(\alpha)$.

- Linearly decaying weights (in function `linear_lag`). These are specified as

$$u0_q = \max(1 - mq, 0)$$

and $u_q = u0_q / \sum_{q=1}^Q u0_q$ for $q = 1, \dots, Q$. The `par_lag` parameter corresponds to $\log(m)$.

- A weighting only between first and second lags (in function `ar2lag`), i.e.

$$u_1 = \alpha, u_2 = 1 - \alpha.$$

The `par_lag` parameter corresponds to $\text{logit}(\alpha)$.

Users can specify their own weighting functions as long as they take the arguments described above and return a vector of weights.

See Also

`profile_par_lag` and `fit_par_lag` estimate `par_lag` in a profiling procedure. `profile_par_lag` is the recommended function, `fit_par_lag` may be quicker for complex models.

Examples

```
## a simple univariate example:
data("salmonella.agona")
## convert old "disProg" to new "sts" data class
salmonella <- disProg2sts(salmonella.agona)
# specify and fit model: fixed geometric lag structure
# with weight 0.8 for first lag
# par_lag is the logit of alpha:
par_lag <- log(0.8/(1 - 0.8))
control_salmonella <- list(end = list(f = addSeason2formula(~ 1)),
                           ar = list(f = addSeason2formula(~ 1)),
                           family = "NegBinM", subset = 6:312,
                           par_lag = par_lag)

fit_salmonella <- hhh4_lag(salmonella, control_salmonella)
summary(fit_salmonella)
# has indeed weight 0.8 on first lag
#
# re-fit with Poisson lags:
par_lag2 <- log(1.2)
control_salmonella2 <- control_salmonella
control_salmonella2$funct_lag = poisson_lag
control_salmonella2$par_lag <- par_lag2
fit_salmonella2 <- hhh4_lag(salmonella, control_salmonella2)
summary(fit_salmonella2)
# the Poisson lag actually allows you to put more weight on
# the second than on the first lag.
```

interpolate_qnbinom	<i>Interpolate between quantiles to avoid edgy display</i>
---------------------	--

Description

An auxiliary function used in `fanplot_prediction`, `fanplot_stationary`

Usage

```
interpolate_qnbinom(p, ...)
```

Arguments

<code>p</code>	A vector of probabilities for which to obtain interpolated quantiles
<code>...</code>	other arguments passed on to <code>pnbinom</code>

is_complex_neighbourhood

Determine whether an hhh4 object was fitted using one of the more complex techniques for handling neighbourhoods

Description

Determine whether an hhh4 object was fitted using one of the more complex techniques for handling neighbourhoods

Usage

```
is_complex_neighbourhood(hhh4obj)
```

Arguments

hhh4obj an hhh4 object

is_fitted_par_lag *Check if the par_lag parameter was fitted*

Description

Check if the par_lag parameter was fitted

Usage

```
is_fitted_par_lag(object)
```

lambda_tilde *Extracting Lambda_Tilde from an hhh4 object with complex neighbourhood structure*

Description

A wrapper around lambda_tilde_complex_neighbourhood and lambda_tilde_simple_neighbourhood.

Usage

```
lambda_tilde(hhh4obj, subset = NULL, periodic = FALSE)
```

Arguments

hhh4obj a hhh4 object for which to extract Lambda_tilde
subset a subset (in time); only required when periodic == FALSE
periodic choose subset to correspond to one full cycle

lambda_tilde_complex_neighbourhood

Extracting Lambda_Tilde from an hhh4 object with complex neighbourhood structure

Description

Extracting Lambda_Tilde from an hhh4 object with complex neighbourhood structure. Used for calculations of longterm predictions and stationary distributions.

Usage

```
lambda_tilde_complex_neighbourhood(hhh4obj, subset = NULL, periodic = FALSE)
```

Arguments

hhh4obj	a hhh4 object for which to extract Lambda_tilde
subset	a subset (in time); only required when periodic == FALSE
periodic	choose subset to correspond to one full cycle

linear_lag

Function to obtain linearly decaying weights

Description

This function generates linearly decaying weights which are subsequently used inside of get_weighted_lags. To be passed to hhh4_lag or profile_par_lag as the control\$funct_lag argument. There are different ways of specifying linearly decaying weights, the version implemented here is

$$u0_q = \max(1 - mq, 0)$$

and $u_q = u0_q / \sum_{q=1}^Q u0_q$ for $q = 1, \dots, Q$.

Usage

```
linear_lag(par_lag, min_lag, max_lag)
```

Arguments

par_lag	a parameter to steer the lag structure, here $\text{logit}(m)$ where m is the linear decay factor; see details of hhh4lag or profile_par_lag.
min_lag	smallest lag to include; the support of the Poisson form starts only at min_lag. Defaults to 1.
max_lag	highest lag to include; higher lags are cut off and the remaining weights standardized. Defaults to 5.

```
log_normal_lag      #' This function generates (shifted) discrete gamma weights which are
                    #' subsequently used inside of get_weighted_lags. To be passed #'
                    #' to hhh4_lag or profile_par_lag as the control$funct_lag argu-
                    #' ment. #' @param par_lag a parameter vector of length 2 to steer the
                    #' lag structure, here log(shape) and log(rate), #' where shape and
                    #' rate are the parameters of the discrete gamma distribution as imple-
                    #' mented in the extraDistr package. #' @param min_lag smallest lag
                    #' to include; the support of the Poisson form starts only at min_lag.
                    #' Defaults to 1. #' @param max_lag highest lag to include; higher
                    #' lags are cut off and he remaining weights standardized. Defaults to 5.
                    #' @author Maria Dunbar, Johannes Bracher #' @export This func-
                    #' tion generates discretized log-normal weights which are subsequently
                    #' used inside of get_weighted_lags. To be passed to hhh4_lag or
                    #' profile_par_lag as the control$funct_lag argument.
```

Description

#' This function generates (shifted) discrete gamma weights which are subsequently used inside of get_weighted_lags. To be passed #' to hhh4_lag or profile_par_lag as the control\$funct_lag argument. #' @param par_lag a parameter vector of length 2 to steer the lag structure, here *log(shape)* and *log(rate)*, #' where *shape* and *rate* are the parameters of the discrete gamma distribution as implemented in the extraDistr package. #' @param min_lag smallest lag to include; the support of the Poisson form starts only at min_lag. Defaults to 1. #' @param max_lag highest lag to include; higher lags are cut off and he remaining weights standardized. Defaults to 5. #' @author Maria Dunbar, Johannes Bracher #' @export This function generates discretized log-normal weights which are subsequently used inside of get_weighted_lags. To be passed to hhh4_lag or profile_par_lag as the control\$funct_lag argument.

Usage

```
log_normal_lag(par_lag, min_lag, max_lag)
```

Arguments

par_lag	a parameter vector of length 2 to steer the lag structure, here <i>meanlog</i> and <i>sdlog</i> , where <i>meanlog</i> and <i>sdlog</i> are the parameters of the log-normal distribution.
min_lag	smallest lag to include; the support of the Poisson form starts only at min_lag. Defaults to 1.
max_lag	highest lag to include; higher lags are cut off and he remaining weights standardized. Defaults to 5.

Author(s)

Maria Dunbar, Johannes Bracher

logLik.hhh4lag	<i>A modified version of logLik.hhh4</i>
----------------	--

Description

A modified version of terms.hhh4 to deal with the added features of the logLik.hhh4 class.

Usage

```
## S3 method for class 'hhh4lag'
logLik(object, ...)
```

matrix_is_cyclic	<i>Check whether the rows of a matrix show a cyclic pattern</i>
------------------	---

Description

Needed to determine whether stationary_moments is applicable (works only for models with periodic parameter structure)

Usage

```
matrix_is_cyclic(matr, length_of_period)
```

Arguments

matr	The parameter matrix to check.
length_of_period	Usually 52 (52 weeks per year).

Value

logical: does the matrix show a periodic pattern?

neOffsetArray.hhh4lag *A modified version of neOffsetArray*

Description

A modified version of neOffsetArray to deal with the added features of the hhh4lag class.

Usage

```
neOffsetArray.hhh4lag(
  object,
  pars = coefW(object),
  subset = object$control$subset
)
```

norobl *Data set on norovirus gastroenteritis in Bremen and Lower Saxony*

Description

Case counts of norovirus gastroenteritis in the German states of Bremen and Lower Saxony, 2011-2017; stored as an sts object

Case counts of rotavirus gastroenteritis in the German states of Bremen and Lower Saxony, 2011-2017; stored as an sts object

Case counts of campylobacteriosis in the German states of Bremen and Lower Saxony, 2011-2017; stored as an sts object

Case counts of norovirus gastroenteritis in the twelve districts of Berlin, Germany, 2011-2017; stored as an sts object

Case counts of rotavirus gastroenteritis in the twelve districts of Berlin, Germany, 2011-2017; stored as an sts object

Author(s)

Johannes Bracher

Source

Surveillance counts retrieved from SurvStat@RKI 2.0 service (<https://survstat.rki.de>), Robert Koch Institute, Berlin as of 30 May 2017.

Surveillance counts retrieved from SurvStat@RKI 2.0 service (<https://survstat.rki.de>), Robert Koch Institute, Berlin as of 30 May 2017.

Surveillance counts retrieved from SurvStat@RKI 2.0 service (<https://survstat.rki.de>), Robert Koch Institute, Berlin as of 30 May 2017.

Surveillance counts retrieved from SurvStat@RKI 2.0 service (<https://survstat.rki.de>), Robert Koch Institute, Berlin.

Surveillance counts retrieved from SurvStat@RKI 2.0 service (<https://survstat.rki.de>), Robert Koch Institute, Berlin.

numeric_fisher_hhh4lag

Numerical evaluation of the covariance matrix including the additional parameter par_lg

Description

Numerical evaluation of the covariance matrix including the additional parameter par_lg

Usage

```
numeric_fisher_hhh4lag(best_mod)
```

Arguments

best_mod	an hhh4lag object; should be generated in profile_par_lag so that the par_lag parameter is already optimized.
----------	---

oneStepAhead_hhh4lag *Predictive Model Assessment for hhh4_lag Models*

Description

A version of surveillance::oneStepAhead for hhh4_lag objects. NOTE: by default the lag_par parameter is not re-fit in this function!

Usage

```
oneStepAhead_hhh4lag(
  result,
  tp,
  type = c("rolling", "first", "final"),
  refit_par_lag = FALSE,
  which.start = c("current", "final"),
  keep.estimates = FALSE,
  verbose = TRUE,
  cores = 1
)
```

Arguments

refit_par_lag	Only new argument compared to surveillance::oneStepAhead: should the lag weighting parameter lag_par be re-fitted in each iteration? Defaults to FALSE.
---------------	---

plot.oneStepAhead	<i>simple plot of one-step-ahead forecasts</i>
-------------------	--

Description

simple plot of one-step-ahead forecasts

Usage

```
## S3 method for class 'oneStepAhead'
plot(x, unit = 1, probs = 1:99/100, start = NULL, means.args = NULL, ...)
```

plot_moments_by_unit	<i>Plot predictive or stationary moments by unit</i>
----------------------	--

Description

Plot negative binomial approximation of predictive or stationary distributions. Usually to be used with aggregated predictions (where columns correspond to regions or age groups; no temporal structure kept).

Usage

```
plot_moments_by_unit(
  mom,
  probs = 1:99/100,
  add_observed = TRUE,
  add_pred_means = TRUE,
  fan.col = colorRampPalette(c("darkgreen", "gray90")),
  pt.col = "red",
  pt.cex = 0.3,
  mean.col = "black",
  mean.lty = "dashed",
  ln = NULL,
  rlab = NULL,
  style = "boxfan",
  space = 0.5,
  add_legend = FALSE,
  probs.legend = c(1, 25, 50, 75, 99)/100,
  ylim = NULL,
  main = NULL,
  xlab = NULL,
  las = NULL,
  axes = TRUE,
  ...
)
```

Arguments

<code>mom</code>	an object of class <code>predictive_moments_hhh4</code> or <code>stationary_moments_hhh4</code> , usually an aggregated prediction without temporal structure (aggregated using <code>aggregate_prediction</code>)
<code>probs</code>	probabilities displayed in <code>fanplot</code> (passed to <code>fanplot::fan</code>)
<code>add_observed</code>	should observed values be added to the plot?
<code>fan.col</code>	color palette for <code>fanplot</code> (passed to <code>fanplot::fan</code>)
<code>pt.col, pt.cex</code>	point color and size for observations
<code>mean.col, mean.lty</code>	line color and type for predictive/stationary means
<code>ln, rlab, style, space</code>	additional arguments passed to <code>fanplot::fan</code>
<code>add_legend</code>	should a legend with the colour coding be added?
<code>probs_legend</code>	probabilities to be displayed in the legend
<code>ylim, main, xlab, las, axes</code>	usual plotting parameters

Examples

```
# load data:
data("noroBL")

#####
# fit a bivariate model:
controlBL <- list(end = list(f = addSeason2formula(~ -1 + fe(1, unitSpecific = TRUE))),
  ar = list(f = ~ -1 + fe(1, unitSpecific = TRUE)),
  ne = list(f = ~ -1 + fe(1, unitSpecific = TRUE)),
  family = "NegBinM", subset = 2:260) # not a very parsimonious parametrization, but feasible
fitBL <- hhh4(noroBL, control = controlBL)
pred_mom <- predictive_moments(fitBL, t_condition = 260, lgt = 52, return_Sigma = TRUE)
# Sigma is required in order to aggregate predictions.

#####
# perform an aggregation over time: total burden in the two regions
aggr_matr_total_burden <- matrix(rep(c(1, 0, 0, 1), 52), nrow = 2,
  dimnames = list(c("Bremen", "Lower Saxony"),
    NULL))
pred_mom_total_burden <- aggregate_moments(pred_mom, aggr_matr_total_burden)
plot_moments_by_unit(pred_mom_total_burden, main = "Total burden 2016", add_legend = TRUE)
```

`poisson_lag`

Function to obtain Poisson weights This function generates Poisson weights which are subsequently used inside of `get_weighted_lags`. To be passed to `hhh4_lag` or `profile_par_lag` as the `control$funct_lag` argument.

Description

Function to obtain Poisson weights This function generates Poisson weights which are subsequently used inside of `get_weighted_lags`. To be passed to `hhh4_lag` or `profile_par_lag` as the `control$funct_lag` argument.

Usage

```
poisson_lag(par_lag, min_lag, max_lag)
```

Arguments

<code>par_lag</code>	a parameter to steer the lag structure, here $\log(\mu)$ where μ is the parameter of the Poisson distribution characterizing the lag structure; see details of <code>hhh4lag</code> or <code>profile_par_lag</code> .
<code>min_lag</code>	smallest lag to include; the support of the Poisson form starts only at <code>min_lag</code> . Defaults to 1.
<code>max_lag</code>	highest lag to include; higher lags are cut off and the remaining weights standardized. Defaults to 5.

<code>predictive_moments</code>	<i>Analytical computation of predictive moments for an hhh4 model</i>
---------------------------------	---

Description

This functions calculates the predictive mean vector and covariance matrix for a path forecast from an hhh4 model.

Usage

```
predictive_moments(
  hhh4Obj,
  t_condition,
  lgt,
  return_Sigma = FALSE,
  return_cov_array = FALSE,
  return_mu_decomposed = FALSE,
  return_M = FALSE
)
```

Arguments

<code>hhh4Obj</code>	an hhh4 object
<code>t_condition</code>	the index of the week on which to condition the path forecast, i.e. an integer between 1 and <code>nrow(hhh4Obj\$stsObj@observed)</code> . If you need forecasts beyond the end of your observed time series you need to artificially prolong the <code>sts</code> object used by adding NA values to the end.

<code>lgt</code>	the length of the path forecast, i.e. 52 for forecasting an entire season when using weekly data
<code>return_Sigma</code>	logical: should the entire variance-covariance matrix of the forecast be returned? defaults to FALSE in order to save storage.
<code>return_cov_array</code>	logical: should an array with week-wise covariance matrices be returned?
<code>return_mu_decomposed</code>	logical: should an array with the predictive means decomposed into the different components be returned?
<code>return_M</code>	logical: should the matrix M containing the predictive first and (un-centered) second moments be returned?

Value

An object of class `predictive_moments_hhh4` containing the following components:

- `mu_matrix` A matrix containing the predictive means. Each row corresponds to a time period and each column to a unit.
- `var_matrix` A matrix containing the predictive variances.
- `cov_array` An array containing time period-wise variance-covariance matrices.
- `mu_vector` as `mu_matrix`, but flattened into a vector.
- `Sigma` a large covariance matrix for all elements of the prediction (corresponding to `mu_vector`)
- `M` a matrix containing predictive means and (un-centered) second moments, specifically $E(c(1, X))$ shall be forecasted. Important in the internal calculation, accessible mainly for de-bugging purposes.
- `mu_decomposed` an array with the same number of rows and columns as `mu_matrix`, but three layers corresponding to the contributions of the three components to the means
- `start` the index (in the original `sts` object) of the first time period of the prediction
- `freq` the length of a cycle
- `n_units` the number of units covered in the prediction
- `timepoints` the timepoints covered by the prediction etc.
- `timepoints` as `timepoints`, but calendar time rather than indices
- `condition` A matrix containing the realizations for the conditioning time period (or periods)
- `realizations_matrix` A matrix containing the realizations that have materialized in the period covered by the prediction.
- `type` "predictive"; to distinguish from stationary moments.
- `has_temporal_structure` does the object still have the original temporal structure? can be set to FALSE when aggregated using `aggregate_prediction`.

Examples

```
data("salmonella.agona")
# convert old "disProg" to new "sts" data class:
salmonella <- disProg2sts(salmonella.agona)
control_salmonella <- list(end = list(f = addSeason2formula(~ 1), lag = 1),
                           ar = list(f = addSeason2formula(~ 1), lag = 1),
                           family = "NegBinM", subset = 6:250)
fit_salmonella <- hhh4_lag(salmonella, control_salmonella) # fit model
# obtain prediction:
pred_mom <- predictive_moments(fit_salmonella, t_condition = 250, lgt = 52)
plot(fit_salmonella)
fanplot_prediction(pred_mom, add = TRUE) # add fan plot
```

print.hhh4lag	<i>A modified version of surveillance::print.hhh4</i>
---------------	---

Description

A modified version of `surveillance::print.hhh4` to deal with the added features of the `hhh4lag` class.

Usage

```
## S3 method for class 'hhh4lag'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

print.summary.hhh4lag	<i>A modified version of print.summary.hhh4</i>
-----------------------	---

Description

A modified version of `print.summary.hhh4` to deal with the added features of the `hhh4lag` class.

Usage

```
## S3 method for class 'summary.hhh4lag'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

profile_par_lag	<i>Estimating the lag decay parameter of an hhh4_lag model using profile likelihood</i>
-----------------	---

Description

Wrapper around hhh4_lag to allow for profile likelihood estimation of the scalar parameter governing the lag structure. hhh4_lag can fit models with fixed lag decay parameter; profile_par_lag re-fits the model for different values of par_lag and finds the optimal value. See ?hhh4_lag for details. NOTE: fit_par_lag serves essentially the same purpose, but is based on a grid of potential values for par_lag rather than optimization using optim. profile_par_lag is the recommended option, but fit_par_lag may be somewhat quicker for complex models.

Usage

```
profile_par_lag(
  stsObj,
  control,
  start_par_lag = NULL,
  lower_par_lag = -10,
  upper_par_lag = 10,
  return_full_cov = FALSE,
  reltol_par_lag = 1e-08,
  check.analyticals = FALSE
)
```

Arguments

stsObj, control, check.analyticals

As in surveillance::hhh4, but control allows for some additional arguments in order to specify a distributed lag structure:

- `funct_lag` Function to compute the lag weights u_q (see details) depending on a scalar parameter `par_lag`. The function has to take the following arguments:
 - `par_lag` A scalar parameter to steer u_q . It should be specified in a way which allows it to take any value in the real numbers
 - `min_lag, max_lag` Minimum and maximum lags; e.g. `min_lag = 3, max_lag = 6` will assign all weights to lags 3 through 6. Usually `min_lag` is set to 1, higher values can be useful for direct forecasting at higher horizons.
- `min_lag, max_lag` Specification of the arguments passed to `funct_lag` to compute the distributed lags. Unlike in `hhh4_lag`, `par_lag` is not to be specified as it is estimated from the data. Important: the first element of the subset argument in `control` needs to be larger than `max_lag` (as for the first `max_lag` observations the fitted values cannot be computed)

`start_par_lag` A starting value for `par_lag`

lower_par_lag, upper_par_lag
 lower and upper limits for the value of par_lag; defaults to -10, 10

return_full_cov
 logical: should the full covariance matrix of the parameter estimates (including par_lag) be obtained numerically?

reltol_par_lag the relative tolerance passed to the optim call to identify par_lag

Details

The standard hhh4 function only allows for models with first lags i.e. of the form

$$mu_{it} = \lambda_{it}X_{i,t-1} + \phi_{it} \sum_{j \neq i} w_{ji}X_{j,t-1} + \nu_{it},$$

see ?hhh4. The extension hhh4_lag allows to specify models of the form

$$mu_{it} = \lambda_{it} \sum_{q=1}^Q u_q X_{i,t-q} + \phi_{it} \sum_{j \neq i} \sum_{q=1}^Q w_{ji} u_q X_{j,t-q} + \nu_{it}.$$

Here the first lags are now replaced by weighted sums of the Q previous observations. The weights u_q , $q = 1, \dots, Q$ sum up to 1 and need to be parametrizable by a single scalar parameter. The value of this parameter needs to be passed as `control$par_lag`. Moreover, a function to obtain a vector of weights from `par_lag` needs to be provided in `control$func_lag`. Currently several such functions are implemented in the package:

- Geometric lags (function `geometric_lag`; the default). These are specified as

$$u0_q = \alpha * (1 - \alpha)^{q-1}$$

and $u_q = u0_q / \sum_{q=1}^Q u0_q$ for $q = 1, \dots, Q$. The `par_lag` parameter corresponds to $\text{logit}(\alpha)$, i.e. the un-normalized weight of the first lag.

- Poisson lags (function `poisson_lag`). These are specified as

$$u0_q = \alpha^q (q-1)! \exp(-\alpha) / (q-1)!,$$

and $u_q = u0_q / \sum_{q=1}^Q u0_q$ for $q = 1, \dots, Q$. Note that the Poisson distribution is shifted by one to achieve a positive support. The `par_lag` parameter corresponds to $\log(\alpha)$.

- Linearly decaying weights (in function `linear_lag`). These are specified as

$$u0_q = \max(1 - mq, 0)$$

and $u_q = u0_q / \sum_{q=1}^Q u0_q$ for $q = 1, \dots, Q$. The `par_lag` parameter corresponds to $\text{logit}(m)$.

- A weighting only between first and second lags (in function `ar2lag`), i.e.

$$u_1 = \alpha, u_2 = 1 - \alpha.$$

The `par_lag` parameter corresponds to $\text{logit}(\alpha)$.

- Unrestricted lag can be fitted using `unrestricted_lag`. These are parameterized via a multinomial logit transformation where the first lag is the reference category.
- Discretized log-normal lags are implemented in `log_normal_lag`, see details there.

Users can specify their own weighting functions as long as they take the arguments described above and return a vector of weights.

Value

If `return_full_cov == FALSE`: an `hhh4_lag` object. If `return_full_cov == TRUE`: A list with two elements: `best_mod` is the `hhh4_lag` fit for the best value of `par_lag`; `cov` is an extended covariance matrix for the regression parameters which also includes `par_lag`.

See Also

`hhh4_lag` for fitting models with fixed `par_lag`; `fit_par_lag` for grid-based optimization.

Examples

```
## a simple univariate example:
data("salmonella.agona")
## convert old "disProg" to new "sts" data class
salmonella <- disProg2sts(salmonella.agona)
# specify and fit model: fixed geometric lag structure
control_salmonella <- list(end = list(f = addSeason2formula(~ 1)),
                           ar = list(f = addSeason2formula(~ 1)),
                           family = "NegBinM", subset = 6:312)
fit_salmonella <- profile_par_lag(salmonella, control_salmonella)
summary(fit_salmonella)
# 0.56 on first lag
#
# re-fit with Poisson lags:
control_salmonella2 <- control_salmonella
control_salmonella2$funct_lag = poisson_lag
fit_salmonella2 <- profile_par_lag(salmonella, control_salmonella2)
summary(fit_salmonella2)
# leads to somewhat different decay and very slightly better AIC
```

psi2size.hhh4lag

A modified version of psi2size.hhh4

Description

A modified version of `psi2size.hhh4` to deal with the added features of the `hhh4lag` class. Extracts estimated overdispersion in `dnbinom()` parametrization (and as matrix)

Usage

```
psi2size.hhh4lag(object, subset = object$control$subset, units = NULL)
```

quantile.oneStepAhead *quantiles of the one-step-ahead forecasts*

Description

quantiles of the one-step-ahead forecasts

Usage

```
## S3 method for class 'oneStepAhead'
quantile(x, probs = c(2.5, 10, 50, 90, 97.5)/100, ...)
```

ranef.hhh4lag *A modified version of ranef.hhh4*

Description

A modified version of ranef.hhh4

Usage

```
## S3 method for class 'hhh4lag'
ranef(object, tomatrix = FALSE, intercept = FALSE, ...)
```

residuals.hhh4lag *A modified version of residuals.hhh4*

Description

A modified version of residuals.hhh4 to deal with the added features of the hhh4lag class. Computes deviance residuals.

Usage

```
## S3 method for class 'hhh4lag'
residuals(object, type = c("deviance", "response"), ...)
```

simulate.hhh4lag	<i>Simulate "hhh4_lag" Count Time Series</i>
------------------	--

Description

This function is the equivalent of `surveillance::simulate.hhh4` for model fits of class `hhh4lag`, obtained from `hhh4_lag` or `profile_par_lag`. The arguments are the same as in `surveillance::simulate.hhh4`, the only difference being that `y.start` needs to be a matrix with `object$control$max_lag` rows and `object$nUnit` columns.

Usage

```
## S3 method for class 'hhh4lag'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  y.start = NULL,
  subset = 1:nrow(object$stsObj),
  coefs = coef(object),
  components = c("ar", "ne", "end"),
  simplify = nsim > 1,
  ...
)
```

Details

This function is still being tested!!!

stationary_moments	<i>Analytic calculation of periodically stationary moments implied by a hhh4-model</i>
--------------------	--

Description

Returns the mean vector and covariance matrix of the periodically stationary distribution implied by an `hhh4` object.

Usage

```
stationary_moments(
  hhh4Obj,
  start = 1,
  n_seasons = 1,
  return_Sigma = FALSE,
```

```

    return_cov_array = FALSE,
    return_mu_decomposed = FALSE,
    return_M = FALSE,
    max.iter = 10,
    tolerance = 1e-05
)

```

Arguments

<code>hhh4obj</code>	hhh4 object for which to calculate stationary moments
<code>start</code>	start of the season
<code>n_seasons</code>	number of
<code>return_Sigma</code>	logical: return entire variance/covariance matrix of the prediction; can take a lot of storage
<code>return_cov_array</code>	logical: return an array containing week-wise covariance matrices
<code>return_mu_decomposed</code>	logical: return an array containing a decomposition of stationary means into the three components endemic, epi.own and epi.others.
<code>return_M</code>	logical: return the array M containing un-centered second moments (used internally for calculations)
<code>max.iter</code>	maximum number of iterations before iterative algorithm stops
<code>tolerance</code>	element-wise maximum tolerance (entering into termination criterion for the iterative calculation)

Value

An object of class `stationary_moments_hhh4` containing the following components:

- `mu_matrix` A matrix containing the stationary means. Each row corresponds to a time period and each column to a unit.
- `var_matrix` A matrix containing the stationary variances.
- `cov_array` An array containing time period-wise variance-covariance matrices.
- `mu_vector` as `mu_matrix`, but flattened into a vector.
- `Sigma` a large covariance matrix for all elements of the prediction (corresponding to `mu_vector`)
- `M` a matrix containing stationary means and (un-centered) second moments, specifically $E(c(1, X))$ Important in the internal calculation, accessible mainly for de-bugging purposes.
- `mu_decomposed` an array with the same number of rows and columns as `mu_matrix`, but three layers corresponding to the contributions of the three components to the means
- `start` the position (within a cycle) of the time period to which the first elements of `mu_matrix` etc. correspond (i.e. the `start` argument from the call of `stationary_moments`)
- `freq` the length of a cycle
- `n_seasons` the number of seasons covered in `mu_matrix` etc.
- `n_units` the number of units covered in the prediction

- timepoints the positions within a cycle of the timepoints covered by mu_matrix etc.
- condition NULL. Only relevant in predictive moments, just a place holder here.
- type "stationary"; to distinguish from predictive moments.
- has_temporal_structure does the object still have the original temporal structure? can be set to FALSE when aggregated using aggregate_prediction.

Examples

```
data("salmonella.agona")
## convert old "disProg" to new "sts" data class
salmonella <- disProg2sts(salmonella.agona)
# specify and fit model
control_salmonella <- list(end = list(f = addSeason2formula(~ 1), lag = 1),
                           ar = list(f = addSeason2formula(~ 1), lag = 1),
                           family = "NegBinM")
fit_salmonella <- hhh4(salmonella, control_salmonella)
# obtain periodically stationary moments:
stat_mom <- stationary_moments(fit_salmonella)
# plot periodically stationary means:
fanplot_stationary(stat_mom)
# add paths of the six seasons in the data set:
for(i in 0:5){
  lines(1:52/52, salmonella@observed[(i*52 + 1):((i + 1)*52)], col = "blue")
}
legend("topleft", col = "blue", lty = 1, legend = "observed seasons")
```

summary.hhh4lag

A modified version of surveillance::summary.hhh4

Description

A modified version of `surveillance::summary.hhh4` to deal with the added features of the `hhh4lag` class.

Usage

```
## S3 method for class 'hhh4lag'
summary(object, maxEV = FALSE, ...)
```

terms.hhh4lag	<i>A modified version of terms.hhh4</i>
---------------	---

Description

A modified version of terms.hhh4 to deal with the added features of the hhh4lag class.

Usage

```
## S3 method for class 'hhh4lag'
terms(x, ...)
```

unrestricted_lag	<i>Function to obtain unrestricted lags</i>
------------------	---

Description

This function generates Q lag weights without a parametric constraint. The weights are obtained via a multinomial logit transformation where the first lag is the reference category.

Usage

```
unrestricted_lag(par_lag, min_lag, max_lag)
```

Arguments

par_lag	the parameter vector of length $Q - 1$
min_lag	smallest lag to include; the support of the Poisson form starts only at min_lag. Defaults to 1.
max_lag	highest lag to include; higher lags are cut off and the remaining weights standardized. Defaults to 5.

update.hhh4lag	<i>A modified version of update.hhh4</i>
----------------	--

Description

A modified version of `update.hhh4` to deal with the added features of the `hhh4lag` class. Note that the lag weights are not re-estimated!

Usage

```
## S3 method for class 'hhh4lag'
update(
  object,
  refit_par_lag = TRUE,
  ...,
  S = NULL,
  subset.upper = NULL,
  use.estimates = object$convergence,
  evaluate = TRUE,
  warning_weights = TRUE
)
```

Index

- * **data**
 - dengueSJ, [6](#)
 - noroBL, [23](#)
- aggregate_moments, [3](#)
- ar2_lag, [4](#)
- confint.oneStepAhead, [5](#)
- decompose.hhh4, [5](#)
- decompose.hhh4lag, [5](#)
- dengueSJ, [6](#)
- distr_lag, [6](#)
- ds_score_hhh4, [7](#)
- fanplot_prediction, [8](#)
- fanplot_stationary, [10](#)
- fit_par_lag, [12](#)
- fixef.hhh4lag, [14](#)
- geometric_lag, [14](#)
- get_diags_of_array, [15](#)
- get_weighted_lags, [15](#)
- hhh4_lag, [16](#)
- interpolate_qnbinom, [18](#)
- is_complex_neighbourhood, [19](#)
- is_fitted_par_lag, [19](#)
- lambda_tilde, [19](#)
- lambda_tilde_complex_neighbourhood, [20](#)
- linear_lag, [20](#)
- log_normal_lag, [21](#)
- logLik.hhh4lag, [22](#)
- matrix_is_cyclic, [22](#)
- neOffsetArray.hhh4lag, [23](#)
- noroBL, [23](#)
- numeric_fisher_hhh4lag, [24](#)
- oneStepAhead_hhh4lag, [24](#)
- plot.oneStepAhead, [25](#)
- plot_moments_by_unit, [25](#)
- poisson_lag, [26](#)
- predictive_moments, [27](#)
- print.hhh4lag, [29](#)
- print.summary.hhh4lag, [29](#)
- profile_par_lag, [30](#)
- psi2size.hhh4lag, [32](#)
- quantile.oneStepAhead, [33](#)
- ranef.hhh4lag, [33](#)
- residuals.hhh4lag, [33](#)
- simulate.hhh4lag, [34](#)
- stationary_moments, [34](#)
- summary.hhh4lag, [36](#)
- terms.hhh4lag, [37](#)
- unrestricted_lag, [37](#)
- update.hhh4lag, [38](#)