Package 'sdmTMB'

May 9, 2026

Type Package

Title Spatial and Spatiotemporal SPDE-Based GLMMs with 'TMB'

Version 1.0.0

Description Implements spatial and spatiotemporal GLMMs (Generalized Linear Mixed Effect Models) using 'TMB', 'fmesher', and the SPDE (Stochastic Partial Differential Equation) Gaussian Markov random field approximation to Gaussian random fields. One common application is for spatially explicit species distribution models (SDMs).
See Anderson et al. (2025) <doi:10.18637/jss.v115.i02>.

License GPL-3

Copyright inst/COPYRIGHTS

URL https://sdmTMB.github.io/sdmTMB/, https://github.com/sdmTMB/sdmTMB

BugReports https://github.com/sdmTMB/sdmTMB/issues

Depends R (>= 4.1.0)

Imports assertthat, abind, cli, fmesher, fishMod, generics, graphics, lifecycle, Matrix, methods, mgcv, mvtnorm, nlme, reformulas, rlang, stats, TMB (>= 1.8.0)

Suggests DHARMa, dplyr, effects (>= 4.0-1), estimability, emmeans (>= 1.4), future, future.apply, ggeffects, ggforce, glmmTMB, ggplot2, knitr, lme4, rmarkdown, sf, spatstat.data, splancs, testthat, tibble, visreg, waywiser

LinkingTo RcppEigen, TMB

VignetteBuilder knitr

ByteCompile true

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

SystemRequirements GNU make

NeedsCompilation yes

Author Sean C. Anderson [aut, cre] (ORCID:
<https://orcid.org/0000-0001-9563-1937>),
Eric J. Ward [aut] (ORCID: <https://orcid.org/0000-0002-4359-0296>),
Philina A. English [aut] (ORCID:
<https://orcid.org/0000-0003-2992-6782>),
Lewis A. K. Barnett [aut] (ORCID:
<https://orcid.org/0000-0002-9381-8375>),
James T. Thorson [aut, cph] (ORCID:
<https://orcid.org/0000-0001-7415-1010>, VAST author),
Joe Watson [ctb] (Censored Poisson),
Julia Indivero [ctb] (ORCID: <https://orcid.org/0000-0001-5310-9542>,
Vignette writing),
Jillian C. Dunic [ctb] (ORCID: <https://orcid.org/0000-0002-0729-3083>),
Joseph Barss [ctb],
Cole C. Monnahan [ctb, cph] (ORCID:
<https://orcid.org/0000-0003-0871-6700>, VAST contributor),
Mollie Brooks [ctb, cph] (ORCID:
<https://orcid.org/0000-0001-6963-8326>, glmmTMB author),
Ben Bolker [ctb, cph] (ORCID: <https://orcid.org/0000-0002-2127-0443>,
glmmTMB author),
Kasper Kristensen [ctb, cph] (TMB/glmmTMB author),
Martin Maechler [ctb, cph] (ORCID:
<https://orcid.org/0000-0002-8685-9910>, glmmTMB author),
Arni Magnusson [ctb, cph] (ORCID:
<https://orcid.org/0000-0003-2769-6741>, glmmTMB author),
Hans J. Skaug [ctb, cph] (glmmTMB author, SPDE barrier),
Anders Nielsen [ctb, cph] (ORCID:
<https://orcid.org/0000-0001-9683-9262>, glmmTMB author),
Casper Berg [ctb, cph] (ORCID: <https://orcid.org/0000-0002-3812-5269>,
glmmTMB author),
Koen van Bentham [ctb, cph] (glmmTMB author),
Olav Nikolai Breivik [ctb, cph] (SPDE barrier),
Simon Wood [ctb, cph] (mgcv: smoother prediction),
Paul-Christian Bürkner [ctb, cph] (brms: smoother matrix parsing),
His Majesty the King in Right of Canada, as represented by the Minister
of the Department of Fisheries and Oceans [cph]

Maintainer Sean C. Anderson <sean@seananderson.ca>

Repository CRAN

Date/Publication 2026-01-24 13:40:02 UTC

Contents

`add_utm_columns` *Add UTM coordinates to a data frame*

Description

Add UTM (Universal Transverse Mercator) coordinates to a data frame. This is useful since geostatistical modeling should generally be performed in an equal-distance projection. You can do this yourself separately with the `sf::st_as_sf()`, `sf::st_transform()`, and `sf::st_coordinates()` functions in the **sf** package.

Usage

```
add_utm_columns(
  dat,
  ll_names = c("longitude", "latitude"),
  ll_crs = 4326,
  utm_names = c("X", "Y"),
  utm_crs = get_crs(dat, ll_names),
  units = c("km", "m")
)

get_crs(dat, ll_names = c("longitude", "latitude"))
```

Arguments

`dat`	Data frame that contains longitude and latitude columns.
`ll_names`	Longitude and latitude column names. **Note the order.**
`ll_crs`	Input CRS value for `ll_names`.
`utm_names`	Output column names for the UTM columns.
`utm_crs`	Output CRS value for the UTM zone; tries to detect with `get_crs()` but can be specified manually.
`units`	UTM units.

Details

Note that longitudes west of the prime meridian should be encoded as running from -180 to 0 degrees.

You may wish to work in km's rather than the standard UTM meters so that the range parameter estimate is not too small, which can cause computational issues. This depends on the the scale of your data.

Value

A copy of the input data frame with new columns for UTM coordinates.

Examples

```
d <- data.frame(lat = c(52.1, 53.4), lon = c(-130.0, -131.4))
get_crs(d, c("lon", "lat"))
add_utm_columns(d, c("lon", "lat"))
```

cAIC *Calculate conditional AIC*

Description

Calculates the conditional Akaike Information criterion (cAIC).

Usage

```
cAIC(object, what = c("cAIC", "EDF"), ...)
```

Arguments

`object`	Output from `sdmTMB()`.
`what`	Whether to return the cAIC or the effective degrees of freedom (EDF) for each group of random effects.
`...`	Other arguments for specific methods. Not used.

Details

cAIC is designed to optimize the expected out-of-sample predictive performance for new data that share the same random effects as the in-sample (fitted) data, e.g., spatial interpolation. In this sense, it should be a fast approximation to optimizing the model structure based on k-fold cross-validation.

By contrast, `AIC()` calculates the marginal Akaike Information Criterion, which is designed to optimize expected predictive performance for new data that have new random effects, e.g., extrapolation, or inference about generative parameters.

cAIC also calculates the effective degrees of freedom (EDF) as a byproduct. This is the number of fixed effects that would have an equivalent impact on model flexibility as a given random effect.

Both cAIC and EDF are calculated using Eq. 6 of Zheng, Cadigan, and Thorson (2024).

For models that include profiled fixed effects, these profiles are turned off.

Value

Either the cAIC or the effective degrees of freedom (EDF) by group of random effects depending on the argument `what`.

References

Deriving the general approximation to cAIC used here:

Zheng, N., Cadigan, N., & Thorson, J. T. (2024). A note on numerical evaluation of conditional Akaike information for nonlinear mixed-effects models (arXiv:2411.14185). arXiv. doi:10.48550/arXiv.2411.14185

The utility of EDF to diagnose hierarchical model behaviour:

Thorson, J. T. (2024). Measuring complexity for hierarchical models using effective degrees of freedom. Ecology, 105(7), e4327 doi:10.1002/ecy.4327

Examples

```
mesh <- make_mesh(dogfish, c("X", "Y"), cutoff = 15)
fit <- sdmTMB(catch_weight ~ s(log(depth)),
  time_varying = ~1,
  time_varying_type = "ar1",
  time = "year",
  spatiotemporal = "off",
  mesh = mesh,
  family = tweedie(),
  data = dogfish,
  offset = log(dogfish$area_swept)
)
cAIC(fit)
cAIC(fit, what = "EDF")
AIC(fit)
```

`coef.sdmTMB` *Get fixed-effect coefficients*

Description

Get fixed-effect coefficients

Usage

```
## S3 method for class 'sdmTMB'
coef(object, complete = FALSE, model = 1, ...)
```

Arguments

`object`	The fitted sdmTMB model object
`complete`	Currently ignored
`model`	Linear predictor for delta models. Defaults to the first linear predictor.
`...`	Currently ignored

`cv_to_waywiser` *Convert* `sdmTMB_cv()` *objects to* **sf** *format for spatial assessment with* **waywiser**

Description

[Experimental] Converts cross-validation results to an `sf::sf()` object for use with spatial model assessment tools such as those in the **waywiser** package. This enables multi-scale spatial assessment of model predictions.

Usage

```
cv_to_waywiser(
  object,
  ll_names = c("longitude", "latitude"),
  ll_crs = 4326,
  utm_crs = get_crs(object$data, ll_names)
)
```

Arguments

`object`	An object of class `sdmTMB_cv` from `sdmTMB_cv()`.
`ll_names`	Column names for longitude and latitude in the original data. **Note the order: longitude first, then latitude.**
`ll_crs`	The coordinate reference system (CRS) for the `ll_names` columns. Defaults to `4326` (WGS84 lon/lat).
`utm_crs`	The projected coordinate reference system (CRS) for the output sf object. By default (if you're feeling lucky!) automatically detected using `get_crs()` based on `ll_names`. Can be manually specified as an EPSG code (e.g., `32609`) or any format accepted by `sf::st_crs()`.

Details

This function is particularly useful for assessing spatial models at multiple scales using the **waywiser** package. After converting to sf format, you can use functions like `waywiser::ww_multi_scale()` to evaluate how model performance changes when predictions are aggregated to different spatial scales.

For delta/hurdle models, the combined predictions are returned (e.g., the product of the encounter probability and positive catch rate).

Value

An `sf::sf()` object with POINT geometry containing:

truth The observed response values

estimate The cross-validated predictions

geometry Spatial point locations

See Also

`sdmTMB_cv()`, `get_crs()`, `https://sdmTMB.github.io/sdmTMB/articles/cross-validation.html`

Examples

```
mesh <- make_mesh(pcod_2011, c("X", "Y"), cutoff = 12)

# Run cross-validation
set.seed(123)
```

```
m_cv <- sdmTMB_cv(
  density ~ depth_scaled,
  data = pcod_2011,
  mesh = mesh,
  family = tweedie(),
  k_folds = 2
)

# Convert with default auto-detected CRS based on lon/lat columns:
cv_sf <- cv_to_waywiser(m_cv, ll_names = c("lon", "lat"))

# Or manually specify the desired UTM CRS:
cv_sf <- cv_to_waywiser(m_cv, ll_names = c("lon", "lat"), utm_crs = 32609)

# Use with waywiser for multi-scale assessment
waywiser::ww_multi_scale(
  cv_sf,
  truth,    # column name (unquoted)
  estimate, # column name (unquoted)
  n = list(c(5, 5), c(2, 2)) # 5x5 and 2x2 grid blocks
)
```

`dharma_residuals` *DHARMa residuals*

Description

Plot (and possibly return) **DHARMa** residuals. This is a wrapper function around `DHARMa::createDHARMa()` to facilitate its use with `sdmTMB()` models. **Note:** It is recommended to set `type = "mle-mvn"` in `simulate.sdmTMB()` for the resulting residuals to have the expected distribution. This is *not* the default.

Usage

```
dharma_residuals(
  simulated_response,
  object,
  plot = TRUE,
  return_DHARMa = FALSE,
  test_uniformity = FALSE,
  test_outliers = FALSE,
  test_dispersion = FALSE,
  ...
)
```

Arguments

simulated_response	Output from simulate.sdmTMB(). It is recommended to set type = "mle-mvn" in the call to simulate.sdmTMB() for the residuals to have the expected distribution.
object	Output from sdmTMB().
plot	Logical. Calls DHARMa::plotQQunif().
return_DHARMa	Logical. Return object from DHARMa::createDHARMa()?
test_uniformity	Passed to testUniformity in DHARMa::plotQQunif().
test_outliers	Passed to testOutliers in DHARMa::plotQQunif().
test_dispersion	Passed to testDispersion in DHARMa::plotQQunif().
...	Other arguments to pass to DHARMa::createDHARMa().

Details

See the residuals vignette.

Advantages to these residuals over the ones from the residuals.sdmTMB() method are (1) they work with delta/hurdle models for the combined predictions, not the just the two parts separately, (2) they should work for all families, not the just the families where we have worked out the analytical quantile function, and (3) they can be used with the various diagnostic tools and plots from the **DHARMa** package.

Disadvantages are (1) they are slower to calculate since one must first simulate from the model, (2) the stability of the distribution of the residuals depends on having a sufficient number of simulation draws, (3) uniformly distributed residuals put less emphasis on the tails visually than normally distributed residuals (which may or may not be desired).

Note that **DHARMa** returns residuals that are uniform(0, 1) if the data are consistent with the model whereas randomized quantile residuals from residuals.sdmTMB() are expected to be normal(0, 1).

Value

A data frame of observed and expected values is invisibly returned so you can assign the output to an object and plot the residuals yourself. See the examples.

If return_DHARMa = TRUE, the object from DHARMa::createDHARMa() is returned and any subsequent **DHARMa** functions can be applied.

See Also

simulate.sdmTMB(), residuals.sdmTMB()

Examples

```
# Try Tweedie family:
fit <- sdmTMB(density ~ as.factor(year) + s(depth, k = 3),
  data = pcod_2011, mesh = pcod_mesh_2011,
```

```
  family = tweedie(link = "log"), spatial = "on")

# The `simulated_response` argument is first so the output from
# simulate() can be piped to `dharma_residuals()`.

# We will work with 100 simulations for fast examples, but you'll
# likely want to work with more than this (enough that the results
# are stable from run to run).

# not great:
set.seed(123)
simulate(fit, nsim = 100, type = "mle-mvn") |>
  dharma_residuals(fit)


# delta-lognormal looks better:
set.seed(123)
fit_dl <- update(fit, family = delta_lognormal())
simulate(fit_dl, nsim = 100, type = "mle-mvn") |>
  dharma_residuals(fit)

# or skip the pipe:
set.seed(123)
s <- simulate(fit_dl, nsim = 100, type = "mle-mvn")
# and manually plot it:
r <- dharma_residuals(s, fit_dl, plot = FALSE)
head(r)
plot(r$expected, r$observed)
abline(0, 1)

# return the DHARMa object and work with the DHARMa methods
ret <- simulate(fit_dl, nsim = 100, type = "mle-mvn") |>
  dharma_residuals(fit, return_DHARMa = TRUE)
plot(ret)
```

`Effect.sdmTMB` *Calculate effects*

Description

Used by effects package

Usage

```
Effect.sdmTMB(focal.predictors, mod, ...)
```

Arguments

focal.predictors	a character vector of one or more predictors in the model in any order.
mod	a regression model object. If no specific method exists for the class of mod, Effect.default will be called.
...	arguments to be passed down.

Value

Output from effects::effect(). Can then be plotted with with associated plot() method.

Examples

```
fit <- sdmTMB(present ~ depth_scaled, data = pcod_2011, family = binomial(),
  spatial = "off")
effects::effect("depth_scaled", fit)
plot(effects::effect("depth_scaled", fit))
```

---

emmeans.sdmTMB *Estimated marginal means with the* **emmeans** *package with* **sdmTMB**

Description

Methods for using the **emmeans** package with **sdmTMB**. The **emmeans** package computes estimated marginal means for the fixed effects.

For delta/hurdle models, you can specify which component to analyze using the model argument: model = 1 for the binomial component (encounter probability) or model = 2 for the positive component (e.g., gamma for delta_gamma()). By default, model = 1.

References

https://aosmith.rbind.io/2019/03/25/getting-started-with-emmeans/

Examples

```
mesh <- make_mesh(pcod_2011, c("X", "Y"), cutoff = 20)
fit <- sdmTMB(
  present ~ as.factor(year),
  data = pcod_2011, mesh = mesh,
  family = binomial()
)
fit
emmeans::emmeans(fit, ~ year)
emmeans::emmeans(fit, pairwise ~ year)
emmeans::emmeans(fit, pairwise ~ year, type = "response")
emmeans::emmeans(fit, pairwise ~ year, adjust = "none")
```

```
e <- emmeans::emmeans(fit, ~ year)
plot(e)

e <- emmeans::emmeans(fit, pairwise ~ year)
confint(e)
summary(e, infer = TRUE)
as.data.frame(e)

# interaction of factor with continuous predictor:
fit2 <- sdmTMB(
  present ~ depth_scaled * as.factor(year),
  data = pcod_2011, mesh = mesh,
  family = binomial()
)
fit2
# slopes for each level:
emmeans::emtrends(fit2, ~ year, var = "depth_scaled")
# test difference in slopes:
emmeans::emtrends(fit2, pairwise ~ year, var = "depth_scaled")
emmeans::emmip(fit2, year ~ depth_scaled,
  at = list(depth_scaled = seq(-2.5, 2.5, length.out = 50)), CIs = TRUE)

# delta/hurdle models:
fit_delta <- sdmTMB(
  density ~ as.factor(year),
  data = pcod_2011, spatial = "off",
  family = delta_gamma()
)
# binomial component (encounter probability):
emmeans::emmeans(fit_delta, ~ year, model = 1)
# positive component (gamma):
emmeans::emmeans(fit_delta, ~ year, model = 2)
```

`get_index` *Extract a relative biomass/abundance index, center of gravity, effective area occupied, or weighted average*

Description

Extract a relative biomass/abundance index, center of gravity, effective area occupied, or weighted average

Usage

```
get_index(obj, bias_correct = TRUE, level = 0.95, area = 1, silent = TRUE, ...)

get_index_split(
  obj,
  newdata,
```

```
  bias_correct = FALSE,
  nsplit = 1,
  level = 0.95,
  area = 1,
  silent = FALSE,
  predict_args = list(),
  ...
)

get_cog(
  obj,
  bias_correct = FALSE,
  level = 0.95,
  format = c("long", "wide"),
  area = 1,
  silent = TRUE,
  ...
)

get_weighted_average(
  obj,
  vector,
  bias_correct = FALSE,
  level = 0.95,
  area = 1,
  silent = TRUE,
  ...
)

get_eao(obj, bias_correct = FALSE, level = 0.95, area = 1, silent = TRUE, ...)
```

Arguments

obj	Output from `predict.sdmTMB()` with `return_tmb_object = TRUE` (the usual case). Alternatively, if `sdmTMB()` was called with `do_index = TRUE`, or if using `get_index_split()`, an object from `sdmTMB()`.
bias_correct	Should bias correction be implemented via `TMB::sdreport()`? Bias correction accounts for the non-linear transformation of random effects when calculating the index. Recommended to be `TRUE` for final analyses, but can be set to `FALSE` for faster calculation while experimenting with models. See Thorson and Kristensen (2016) in the References.
level	The confidence level.
area	Grid cell area for area weighting the index. Can be: (1) a numeric vector of length `nrow(newdata)` with area for each grid cell, (2) a single numeric value to apply to all grid cells, or (3) a character value giving the column name in `newdata` containing areas.
silent	Silent?

...	Passed to TMB::sdreport().
newdata	New data (e.g., a prediction grid by year) to pass to predict.sdmTMB() in the case of get_index_split().
nsplit	The number of splits to do the calculation in. For memory intensive operations (large grids and/or models), it can be helpful to do the prediction, area integration, and bias correction on subsets of time slices (e.g., years) instead of all at once. If nsplit > 1, this will usually be slower but with reduced memory use.
predict_args	A list of arguments to pass to predict.sdmTMB() in the case of get_index_split().
format	Long or wide.
vector	A numeric vector of the same length as the prediction data, containing the values to be averaged (e.g., depth, temperature).

Value

For get_index(): A data frame with columns for time, estimate (area-weighted total abundance or biomass), lower and upper confidence intervals, log estimate, and standard error of the log estimate.

For get_cog(): A data frame with columns for time, estimate (center of gravity: the abundance-weighted mean x and y coordinates), lower and upper confidence intervals, and standard error of center of gravity coordinates.

For get_eao(): A data frame with columns for time, estimate (effective area occupied: the area required if the population was spread evenly at the arithmetic mean density), lower and upper confidence intervals, log EAO, and standard error of the log EAO estimates.

For get_weighted_average(): A data frame with columns for time, estimate (weighted average of the provided vector, weighted by predicted density), lower and upper confidence intervals, and standard error of the estimates.

References

Geostatistical model-based indices of abundance (along with many newer papers):

Shelton, A.O., Thorson, J.T., Ward, E.J., and Feist, B.E. 2014. Spatial semiparametric models improve estimates of species abundance and distribution. Canadian Journal of Fisheries and Aquatic Sciences 71(11): 1655–1666. doi:10.1139/cjfas20130508

Thorson, J.T., Shelton, A.O., Ward, E.J., and Skaug, H.J. 2015. Geostatistical delta-generalized linear mixed models improve precision for estimated abundance indices for West Coast groundfishes. ICES J. Mar. Sci. 72(5): 1297–1310. doi:10.1093/icesjms/fsu243

Geostatistical model-based centre of gravity:

Thorson, J.T., Pinsky, M.L., and Ward, E.J. 2016. Model-based inference for estimating shifts in species distribution, area occupied and centre of gravity. Methods Ecol Evol 7(8): 990–1002. doi:10.1111/2041210X.12567

Geostatistical model-based effective area occupied:

Thorson, J.T., Rindorf, A., Gao, J., Hanselman, D.H., and Winker, H. 2016. Density-dependent changes in effective area occupied for sea-bottom-associated marine fishes. Proceedings of the Royal Society B: Biological Sciences 283(1840): 20161853. doi:10.1098/rspb.2016.1853

Bias correction:

Thorson, J.T., and Kristensen, K. 2016. Implementing a generic method for bias correction in statistical models using random effects, with spatial and population dynamics examples. Fisheries Research 175: 66–74. doi:10.1016/j.fishres.2015.11.016

See Also

`get_index_sims()`

Examples

```
library(ggplot2)

# use a small number of knots for this example to make it fast:
mesh <- make_mesh(pcod, c("X", "Y"), n_knots = 60)

# fit a spatiotemporal model:
m <- sdmTMB(
 data = pcod,
 formula = density ~ 0 + as.factor(year),
 time = "year", mesh = mesh, family = tweedie(link = "log")
)

# prepare a prediction grid:
nd <- replicate_df(qcs_grid, "year", unique(pcod$year))

# Note `return_tmb_object = TRUE` and the prediction grid:
predictions <- predict(m, newdata = nd, return_tmb_object = TRUE)

# biomass index:
ind <- get_index(predictions, bias_correct = TRUE)
ind
ggplot(ind, aes(year, est)) + geom_line() +
  geom_ribbon(aes(ymin = lwr, ymax = upr), alpha = 0.4) +
  ylim(0, NA)

# do that in 2 chunks
# only necessary for very large grids to save memory
# will be slower but save memory
# note the first argument is the model fit object:
ind <- get_index_split(m, newdata = nd, nsplit = 2, bias_correct = TRUE)

# center of gravity:
cog <- get_cog(predictions, format = "wide")
cog
ggplot(cog, aes(est_x, est_y, colour = year)) +
  geom_point() +
  geom_linerange(aes(xmin = lwr_x, xmax = upr_x)) +
  geom_linerange(aes(ymin = lwr_y, ymax = upr_y)) +
  scale_colour_viridis_c()

# effective area occupied:
```

```
eao <- get_eao(predictions)
eao
ggplot(eao, aes(year, est)) + geom_line() +
  geom_ribbon(aes(ymin = lwr, ymax = upr), alpha = 0.4) +
  ylim(0, NA)

# weighted average (e.g., depth-weighted by biomass):
wa <- get_weighted_average(predictions, vector = nd$depth)
wa
ggplot(wa, aes(year, est)) + geom_line() +
  geom_ribbon(aes(ymin = lwr, ymax = upr), alpha = 0.4)
```

`get_index_sims` *Calculate a population index via simulation from the joint precision matrix*

Description

[Experimental]

Calculate a population index via simulation from the joint precision matrix. Compared to `get_index()`, this version can be faster if bias correction was turned on in `get_index()` while being approximately equivalent. **This is an experimental function.** This function usually works reasonably well, but we make no guarantees. It is recommended to use `get_index()` with `bias_correct = TRUE` for final inference.

Usage

```
get_index_sims(
  obj,
  level = 0.95,
  return_sims = FALSE,
  area = rep(1, nrow(obj)),
  est_function = stats::median,
  area_function = function(x, area) x + log(area),
  agg_function = function(x) sum(exp(x))
)
```

Arguments

`obj`	`predict.sdmTMB()` output with `nsim > 0`.
`level`	The confidence level.
`return_sims`	Logical. Return simulation draws? The default (`FALSE`) is a quantile summary of those simulation draws.

area	A vector of grid cell/polyon areas for each year-grid cell (row of data) in obj. Adjust this if cells are not of unit area or not all the same area (e.g., some cells are partially over land/water). Note that the area vector is added as log(area) to the raw values in obj. In other words, the function assumes a log link, which typically makes sense.
est_function	Function to summarize the estimate (the expected value). mean() would be an alternative to median().
area_function	Function to apply area weighting. Assuming a log link, the function(x, area) x + log(area) default makes sense. If in natural space, function(x, area) x * area makes sense.
agg_function	Function to aggregate samples within each time slice. Assuming a log link, the function(x) sum(exp(x)) default makes sense. If in natural space, function(x) sum(x) makes sense.

Details

Can also be used to produce an index from a model fit with **tmbstan**.

This function does nothing more than summarize and reshape the matrix of simulation draws into a data frame.

Value

A data frame. If return_sims = FALSE:

- name of column (e.g. year) that was supplied to sdmTMB() time argument
- est: estimate
- lwr: lower confidence interval value
- upr: upper confidence interval value
- log_est: log estimate
- se: standard error on the log estimate

If return_sims = TRUE, samples from the index values in a long-format data frame:

- name of column (e.g. year) that was supplied to sdmTMB() time argument
- .value: sample value
- .iteration: sample number

See Also

get_index()

Examples

```
m <- sdmTMB(density ~ 0 + as.factor(year),
  data = pcod_2011, mesh = pcod_mesh_2011, family = tweedie(link = "log"),
  time = "year"
)
```

```
qcs_grid_2011 <- replicate_df(qcs_grid, "year", unique(pcod_2011$year))
p <- predict(m, newdata = qcs_grid_2011, nsim = 100)
x <- get_index_sims(p)
x_sims <- get_index_sims(p, return_sims = TRUE)

if (require("ggplot2", quietly = TRUE)) {
  ggplot(x, aes(year, est, ymin = lwr, ymax = upr)) +
    geom_line() +
    geom_ribbon(alpha = 0.4)
  ggplot(x_sims, aes(as.factor(year), .value)) +
    geom_violin()
}

# Demo custom functions if working in natural space:
ind <- get_index_sims(
  exp(p),
  agg_function = function(x) sum(x),
  area_function = function(x, area) x * area
)
```

`get_pars` *Get TMB parameter list*

Description

Get TMB parameter list

Usage

```
get_pars(object)
```

Arguments

`object`	Fit from `sdmTMB()`

Value

A named list of parameter values

Examples

```
fit <- sdmTMB(present ~ 1, data = pcod_2011, family = binomial(), spatial = "off")
pars <- get_pars(fit)
names(pars)
```

`get_range_edge` *Calculate range edges via simulation from the joint precision matrix*

Description

[Experimental]

Calculate range edges as density-weighted quantiles along a spatial axis. Range edges are calculated as the positions along a user-supplied spatial axis (e.g., latitude, coastal distance) where the cumulative proportion of density equals specified quantiles (e.g., 0.01 and 0.99 for the lower and upper 1% range edges). Uncertainty is calculated via simulation from the joint precision matrix.

Usage

```
get_range_edge(
  obj,
  axis,
  quantiles = c(0.025, 0.975),
  level = 0.95,
  return_sims = FALSE
)
```

Arguments

`obj`	`predict.sdmTMB()` output with `nsim > 0`. The prediction object should include predictions on a spatial grid that covers the area of interest.
`axis`	Numeric vector of the same length as the prediction data, representing the spatial axis along which to calculate range edges (e.g., latitude, coastal distance values). This should align with the rows of the prediction matrix.
`quantiles`	Numeric vector of quantiles to calculate. Default is `c(0.025, 0.975)` for lower and upper 1% range edges. Common alternatives include `c(0.01, 0.99)` for 1% edges or `c(0.05, 0.5, 0.95)` to include the median.
`level`	The confidence level for uncertainty intervals.
`return_sims`	Logical. Return simulation draws? The default (`FALSE`) returns a quantile summary of the simulation draws.

Details

This function implements a similar approach to VAST's range edge calculations, following methods from Fredston et al. (2021) and similar studies. The method:

1. Orders spatial locations by position along the specified axis
2. Calculates cumulative proportion of total density along that axis
3. Finds positions where cumulative proportion equals target quantiles
4. Uses simulation from the joint precision to quantify uncertainty

To find the exact position where the cumulative proportion equals a target quantile, the function uses linear interpolation between adjacent grid points. This provides more accurate range edge estimates than selecting the closest grid point, especially on coarser grids or for extreme quantiles (e.g., 0.01, 0.99).

Value

A data frame. If `return_sims = FALSE`:

- name of time column (e.g., `year`) that was supplied to `sdmTMB()` time argument
- `quantile`: the quantile value (from `quantiles` argument)
- `est`: estimated range edge position
- `lwr`: lower confidence interval
- `upr`: upper confidence interval
- `se`: standard error

If `return_sims = TRUE`, simulation draws from range edge positions in long format:

- name of time column (e.g., `year`)
- `quantile`: the quantile value
- `.value`: simulated range edge position
- `.iteration`: simulation number

References

Fredston, A. L., Pinsky, M., Selden, R. L., Szuwalski, C., Thorson, J. T., Gaines, S. D., & Halpern, B. S. (2021). Range edges of North American marine species are tracking temperature over decades. Global Change Biology, 27(13), 3145-3156. doi:10.1111/gcb.15614

Examples

```
# Fit a spatiotemporal model
mesh <- make_mesh(pcod, c("X", "Y"), n_knots = 100)
m <- sdmTMB(
  density ~ 0 + as.factor(year),
  data = pcod, mesh = mesh, family = tweedie(link = "log"),
  time = "year", spatiotemporal = "iid", spatial = "on"
)

# Create prediction grid
nd <- replicate_df(qcs_grid, "year", unique(pcod$year))

# Get predictions with simulations
p <- predict(m, newdata = nd, nsim = 100)

# Calculate range edges along latitude (Y coordinate)
edges <- get_range_edge(p, axis = nd$Y)
edges
```

```
# Plot range edges over time
if (require("ggplot2", quietly = TRUE)) {
  ggplot(edges, aes(year, est, colour = as.factor(quantile))) +
    geom_line() +
    geom_ribbon(aes(ymin = lwr, ymax = upr, fill = as.factor(quantile)),
      alpha = 0.2
    ) +
    labs(y = "Latitude", colour = "Quantile", fill = "Quantile")
}

# Get simulation draws for further analysis
edges_sims <- get_range_edge(p, axis = nd$Y, return_sims = TRUE)
```

`make_category_svc` *Set up spatially varying coefficients for category composition models*

Description

This function helps set up the data structure, formula, and mapping needed for fitting spatially varying coefficient models with categories (e.g., ages, length bins, species) that have both spatial and spatiotemporal random fields. It's particularly useful for age or length composition standardization models.

Usage

```
make_category_svc(
  data,
  category_column,
  time_column,
  share_spatial_sd = TRUE,
  share_spatiotemporal_sd = TRUE
)
```

Arguments

`data`	Data frame containing the composition data.
`category_column`	Character. Name of the category column (e.g., "Age", "length_bin", "species").
`time_column`	Character. Name of the time column (e.g., "Year").
`share_spatial_sd`	Logical. If `TRUE`, all categories share the same spatial SD. If `FALSE`, each category gets its own spatial SD.
`share_spatiotemporal_sd`	Logical. If `TRUE`, all category-time combinations share the same spatiotemporal SD. If `FALSE`, each gets its own.

Details

This function creates spatially varying coefficient structures for composition models by setting up:

1. **Spatial fields**: One field per category (e.g., age-specific spatial fields) 2. **Spatiotemporal fields**: One field per category-time combination (e.g., age-year fields)

The sharing of variance parameters is controlled by `share_spatial_sd` and `share_spatiotemporal_sd`. When `TRUE`, all fields of that type share the same variance parameter, which is more parsimonious but assumes similar variance magnitudes across categories.

The resulting model structure allows each category to have its own spatial pattern and temporal variation while controlling parameter sharing for identifiability and computational efficiency.

Value

A list containing:

- `data_expanded`: Data frame with added model matrix columns for use in `sdmTMB()`.
- `svc_formula`: Formula for the `spatial_varying` argument in `sdmTMB()`.
- `svc_map`: Map list for the `map` argument in `sdmTMBcontrol()`.
- `info`: List with summary information about the model structure.

Examples

```
set.seed(123)
data <- data.frame(
  age = factor(rep(1:3, each = 20)),
  year = rep(2020:2022, 20),
  abundance = rnorm(60),
  x = runif(60), y = runif(60)
)

# Set up model components
setup <- make_category_svc(
  data = data,
  category_column = "age",
  time_column = "year",
  share_spatial_sd = TRUE,
  share_spatiotemporal_sd = TRUE
)

# Check the setup
setup$info

# See the age composition standardization vignette for more details
```

make_mesh *Construct an SPDE mesh for sdmTMB*

Description

Construct an SPDE mesh for use with sdmTMB.

Usage

```
make_mesh(
  data,
  xy_cols,
  type = c("kmeans", "cutoff", "cutoff_search"),
  cutoff,
  n_knots,
  seed = 42,
  mesh = NULL,
  fmesher_func = fmesher::fm_rcdt_2d_inla,
  convex = NULL,
  concave = convex,
  ...
)

## S3 method for class 'sdmTMBmesh'
plot(x, ...)
```

Arguments

data	A data frame.
xy_cols	A character vector of x and y column names contained in data. These should likely be in an equal distance projection. For a helper function to convert to UTMs, see add_utm_columns().
type	Method to create the mesh. Also see mesh argument to supply your own mesh.
cutoff	An optional cutoff if type is "cutoff". The minimum allowed triangle edge length.
n_knots	The number of desired knots if type is not "cutoff".
seed	Random seed. Affects stats::kmeans() determination of knot locations if type = "kmeans".
mesh	An optional mesh created via **fmesher** instead of using the above convenience options.
fmesher_func	Which **fmesher** function to use. Options include fmesher::fm_rcdt_2d_inla() and fmesher::fm_mesh_2d_inla() along with version without the _inla on the end.
convex	If specified, passed to fmesher::fm_nonconvex_hull(). Distance to extend non-convex hull from data.

concave	If specified, passed to fmesher::fm_nonconvex_hull(). "Minimum allowed reentrant curvature". Defaults to convex.
...	Passed to graphics::plot().
x	Output from make_mesh().

Value

make_mesh(): A list of class sdmTMBmesh. The element mesh is the output from fmesher_func (default is fmesher::fm_mesh_2d_inla()). See mesh$mesh$n for the number of vertices.

plot.sdmTMBmesh(): A plot of the mesh and data points. To make your own **ggplot2** version, pass your_mesh$mesh to inlabru::gg().

Examples

```
# Extremely simple cutoff:
mesh <- make_mesh(pcod, c("X", "Y"), cutoff = 5, type = "cutoff")
plot(mesh)

# Using a k-means algorithm to assign vertices:
mesh <- make_mesh(pcod, c("X", "Y"), n_knots = 50, type = "kmeans")
plot(mesh)


# But, it's better to develop more tailored meshes:

# Pass arguments via '...' to fmesher::fm_mesh_2d_inla():
mesh <- make_mesh(
  pcod, c("X", "Y"),
  fmesher_func = fmesher::fm_mesh_2d_inla,
  cutoff = 8, # minimum triangle edge length
  max.edge = c(20, 40), # inner and outer max triangle lengths
  offset = c(5, 40) # inner and outer border widths
)
plot(mesh)

# Or define a mesh directly with fmesher (formerly in INLA):
inla_mesh <- fmesher::fm_mesh_2d_inla(
  loc = cbind(pcod$X, pcod$Y), # coordinates
  max.edge = c(25, 50), # max triangle edge length; inner and outer meshes
  offset = c(5, 25),  # inner and outer border widths
  cutoff = 5 # minimum triangle edge length
)
mesh <- make_mesh(pcod, c("X", "Y"), mesh = inla_mesh)
plot(mesh)
```

pcod *Example fish survey data*

Description

Various fish survey datasets.

Usage

```
pcod

pcod_2011

pcod_mesh_2011

qcs_grid

dogfish

yelloweye

hbll_s_grid

wcvi_grid
```

Format

`pcod`: Trawl survey data for Pacific Cod in Queen Charlotte Sound. A data frame.

`pcod_2011`: A version of `pcod` for years 2011 and after (smaller for speed). A data frame.

`pcod_mesh_2011`: A mesh pre-built for `pcod_2011` for examples. A list of class `sdmTMBmesh`.

`qcs_grid` A 2x2km prediction grid for Queen Charlotte Sound. A data frame.

`dogfish`: Trawl survey data for Pacific Spiny Dogfish on West Coast Vancouver Island. A data frame.

`yelloweye`: Survey data for Yelloweye Rockfish from the Hard Bottom Longline Survey (South) off West Coast Vancouver Island.

`hbll_s_grid`: A survey domain grid to go with `yelloweye`. A data frame.

`wcvi_grid`: A survey domain grid to go with `dogfish`. A data frame.

plot_anisotropy *Plot anisotropy from an sdmTMB model*

Description

Anisotropy is when spatial correlation is directionally dependent. In `sdmTMB()`, the default spatial correlation is isotropic, but anisotropy can be enabled with `anisotropy = TRUE`. These plotting functions help visualize that estimated anisotropy.

Usage

```
plot_anisotropy(object, return_data = FALSE)

plot_anisotropy2(object, model = 1)
```

Arguments

`object`	An object from `sdmTMB()`.
`return_data`	Logical. Return a data frame? `plot_anisotropy()` only.
`model`	Which model if a delta model (only for `plot_anisotropy2()`; `plot_anisotropy()` always plots both).

Value

`plot_anisotropy()`: One or more ellipses illustrating the estimated anisotropy. The ellipses are centered at coordinates of zero in the space of the X-Y coordinates being modeled. The ellipses show the spatial and/or spatiotemporal range (distance at which correlation is effectively independent) in any direction from zero. Uses **ggplot2**. If anisotropy was turned off when fitting the model, `NULL` is returned instead of a **ggplot2** object.

`plot_anisotropy2()`: A plot of eigenvectors illustrating the estimated anisotropy. A list of the plotted data is invisibly returned. Uses base graphics. If anisotropy was turned off when fitting the model, `NULL` is returned instead of a plot object.

References

Code adapted from VAST R package

Examples

```
mesh <- make_mesh(pcod_2011, c("X", "Y"), n_knots = 80, type = "kmeans")
fit <- sdmTMB(
  data = pcod_2011,
  formula = density ~ 1,
  mesh = mesh,
  family = tweedie(),
  share_range = FALSE,
  anisotropy = TRUE #<
```

```
)
plot_anisotropy(fit)
plot_anisotropy2(fit)
```

`plot_pc_matern` *Plot PC Matérn priors*

Description

Plot PC Matérn priors

Usage

```
plot_pc_matern(
  range_gt,
  sigma_lt,
  range_prob = 0.05,
  sigma_prob = 0.05,
  range_lims = c(range_gt * 0.1, range_gt * 10),
  sigma_lims = c(0, sigma_lt * 2),
  plot = TRUE
)
```

Arguments

`range_gt`	A value one expects the spatial or spatiotemporal range is **g**reater **t**han with `1 - range_prob` probability.
`sigma_lt`	A value one expects the spatial or spatiotemporal marginal standard deviation (`sigma_0` or `sigma_E` internally) is **l**ess **t**han with `1 - sigma_prob` probability.
`range_prob`	Probability. See description for `range_gt`.
`sigma_prob`	Probability. See description for `sigma_lt`.
`range_lims`	Plot range variable limits.
`sigma_lims`	Plot sigma variable limits.
`plot`	Logical controlling whether plot is drawn (defaults to `TRUE`).

Value

A plot from `image()`. Invisibly returns the underlying matrix data. The rows are the sigmas. The columns are the ranges. Column and row names are provided.

See Also

`pc_matern()`

Examples

```
plot_pc_matern(range_gt = 5, sigma_lt = 1)
plot_pc_matern(range_gt = 5, sigma_lt = 10)
plot_pc_matern(range_gt = 5, sigma_lt = 1, sigma_prob = 0.2)
plot_pc_matern(range_gt = 5, sigma_lt = 1, range_prob = 0.2)
```

---

plot_smooth *Plot a smooth term from an sdmTMB model*

Description

Deprecated: use `visreg::visreg()`**. See** `visreg_delta()` **for examples.**

Usage

```
plot_smooth(
  object,
  select = 1,
  n = 100,
  level = 0.95,
  ggplot = FALSE,
  rug = TRUE,
  return_data = FALSE
)
```

Arguments

`object`	An `sdmTMB()` model.
`select`	The smoother term to plot.
`n`	The number of equally spaced points to evaluate the smoother along.
`level`	The confidence level.
`ggplot`	Logical: use the **ggplot2** package?
`rug`	Logical: add rug lines along the lower axis?
`return_data`	Logical: return the predicted data instead of making a plot?

Details

Note:

- Any numeric predictor is set to its mean
- Any factor predictor is set to its first-level value
- The time element (if present) is set to its minimum value
- The x and y coordinates are set to their mean values

Value

A plot of a smoother term.

Examples

```
d <- subset(pcod, year >= 2000 & density > 0)
pcod_spde <- make_mesh(d, c("X", "Y"), cutoff = 30)
m <- sdmTMB(
  data = d,
  formula = log(density) ~ s(depth_scaled) + s(year, k = 5),
  mesh = pcod_spde
)
plot_smooth(m)
```

---

predict.sdmTMB *Predict from an sdmTMB model*

Description

Make predictions from an **sdmTMB** model; can predict on the original or new data.

Usage

```
## S3 method for class 'sdmTMB'
predict(
  object,
  newdata = NULL,
  type = c("link", "response"),
  se_fit = FALSE,
  re_form = NULL,
  re_form_iid = NULL,
  nsim = 0,
  sims_var = "est",
  model = c(NA, 1, 2),
  offset = NULL,
  mcmc_samples = NULL,
  return_tmb_object = FALSE,
  return_tmb_report = FALSE,
  return_tmb_data = FALSE,
  ...
)
```

Arguments

object	A model fitted with sdmTMB().
newdata	A data frame to make predictions on. This should be a data frame with the same predictor columns as in the fitted data and a time column (if this is a spatiotemporal model) with the same name as in the fitted data.

`type` Should the `est` column be in link (default) or response space?

`se_fit` Should standard errors on predictions be calculated? Warning: can be slow for large datasets or high-resolution projections when random fields are included. For faster uncertainty estimation, either use `re_form = NA` to exclude random fields or use the `nsim` argument to simulate from the joint precision matrix.

`re_form` `NULL` to include all spatial/spatiotemporal random fields in predictions. `~0` or `NA` for population-level predictions (predictions from fixed effects only, marginalizing over random fields). Often used with `se_fit = TRUE` to visualize marginal effects. Does not affect `get_index()` calculations.

`re_form_iid` `NULL` to specify including all random intercepts in the predictions. `~0` or `NA` for population-level predictions. No other options (e.g., some but not all random intercepts) are implemented yet. Only affects predictions with `newdata`. This *does* affects `get_index()`.

`nsim` If `> 0`, simulate from the joint precision matrix with `nsim` draws. Returns a matrix of `nrow(newdata)` by `nsim` with each column representing one draw of the linear predictor (in link space). Simulating from the joint precision matrix accounts for uncertainty in both fixed and random effects. Use this to derive uncertainty on predictions (e.g., `apply(x, 1, sd)`) or propagate uncertainty to derived quantities. This is the fastest way to characterize spatial uncertainty with sdmTMB.

`sims_var` Experimental: Which TMB reported variable from the model should be extracted from the joint precision matrix simulation draws? Defaults to link-space predictions. Options include: `"omega_s"`, `"zeta_s"`, `"epsilon_st"`, and `"est_rf"` (as described below). Other options will be passed verbatim.

`model` Which component to predict from delta/hurdle models when `nsim > 0` or `mcmc_samples` is supplied. `NA` (default) returns the combined prediction from both components; `1` returns the binomial component only; `2` returns the positive component only. Predictions are on the link or response scale depending on `type`. For regular predictions (without simulation), both components are returned. See the delta-model vignette.

`offset` A numeric vector of optional offset values. If left at default `NULL`, the offset is implicitly left at 0.

`mcmc_samples` See `extract_mcmc()` in the sdmTMBextra package for more details and the Bayesian vignette. If specified, the predict function will return a matrix of a similar form as if `nsim > 0` but representing Bayesian posterior samples from the Stan model.

`return_tmb_object` Logical. If `TRUE`, will include the TMB object in a list format output. Necessary for the `get_index()` or `get_cog()` functions.

`return_tmb_report` Logical: return the output from the TMB report? For regular prediction, this is all the reported variables at the MLE parameter values. For `nsim > 0` or when `mcmc_samples` is supplied, this is a list where each element is a sample and the contents of each element is the output of the report for that sample.

`return_tmb_data` Logical: return formatted data for TMB? Used internally.

`...` Not implemented.

Value

If `return_tmb_object = FALSE` (and `nsim = 0` and `mcmc_samples = NULL`):

A data frame:

- `est`: Estimate in link space (everything included)
- `est_non_rf`: Estimate from everything except random fields (fixed effects, random intercepts, time-varying effects, etc.)
- `est_rf`: Estimate from all random fields combined
- `omega_s`: Spatial random field (models consistent spatial patterns)
- `zeta_s`: Spatially varying coefficient field (models how effects vary across space)
- `epsilon_st`: Spatiotemporal random field (models spatial patterns that vary over time)

If `return_tmb_object = TRUE` (and `nsim = 0` and `mcmc_samples = NULL`):

A list:

- `data`: The data frame described above
- `report`: The TMB report on parameter values
- `obj`: The TMB object returned from the prediction run
- `fit_obj`: The original TMB model object

In this case, you likely only need the `data` element as an end user. The other elements are included for other functions.

If `nsim > 0` or `mcmc_samples` is not `NULL`:

A matrix:

- Columns represent samples
- Rows represent predictions with one row per row of `newdata`

Examples

```
d <- pcod_2011
mesh <- make_mesh(d, c("X", "Y"), cutoff = 30) # a coarse mesh for example speed
m <- sdmTMB(
 data = d, formula = density ~ 0 + as.factor(year) + depth_scaled + depth_scaled2,
 time = "year", mesh = mesh, family = tweedie(link = "log")
)

# Predictions at original data locations ------------------------------

predictions <- predict(m)
head(predictions)

predictions$resids <- residuals(m) # randomized quantile residuals

library(ggplot2)
ggplot(predictions, aes(X, Y, col = resids)) + scale_colour_gradient2() +
```

```
  geom_point() + facet_wrap(~year)
hist(predictions$resids)
qqnorm(predictions$resids);abline(a = 0, b = 1)

# Predictions onto new data ------------------------------------------

qcs_grid_2011 <- replicate_df(qcs_grid, "year", unique(pcod_2011$year))
predictions <- predict(m, newdata = qcs_grid_2011)


# A short function for plotting our predictions:
plot_map <- function(dat, column = est) {
  ggplot(dat, aes(X, Y, fill = {{ column }})) +
    geom_raster() +
    facet_wrap(~year) +
    coord_fixed()
}

plot_map(predictions, exp(est)) +
  scale_fill_viridis_c(trans = "sqrt") +
  ggtitle("Prediction (fixed effects + all random effects)")

plot_map(predictions, exp(est_non_rf)) +
  ggtitle("Prediction (fixed effects and any time-varying effects)") +
  scale_fill_viridis_c(trans = "sqrt")

plot_map(predictions, est_rf) +
  ggtitle("All random field estimates") +
  scale_fill_gradient2()

plot_map(predictions, omega_s) +
  ggtitle("Spatial random effects only") +
  scale_fill_gradient2()

plot_map(predictions, epsilon_st) +
  ggtitle("Spatiotemporal random effects only") +
  scale_fill_gradient2()

# Visualizing a marginal effect ----------------------------------------

# See the visreg package or the ggeffects::ggeffect() or
# ggeffects::ggpredict() functions
# To do this manually:

nd <- data.frame(depth_scaled =
  seq(min(d$depth_scaled), max(d$depth_scaled), length.out = 100))
nd$depth_scaled2 <- nd$depth_scaled^2

# Because this is a spatiotemporal model, you'll need at least one time
# element. If time isn't also a fixed effect then it doesn't matter what you pick:
nd$year <- 2011L # L: integer to match original data
p <- predict(m, newdata = nd, se_fit = TRUE, re_form = NA)
ggplot(p, aes(depth_scaled, exp(est),
```

```
  ymin = exp(est - 1.96 * est_se), ymax = exp(est + 1.96 * est_se))) +
  geom_line() + geom_ribbon(alpha = 0.4)

# Plotting marginal effect of a spline ----------------------------------

m_gam <- sdmTMB(
 data = d, formula = density ~ 0 + as.factor(year) + s(depth_scaled, k = 5),
 time = "year", mesh = mesh, family = tweedie(link = "log")
)
if (require("visreg", quietly = TRUE)) {
  visreg::visreg(m_gam, "depth_scaled")
}

# or manually:
nd <- data.frame(depth_scaled =
  seq(min(d$depth_scaled), max(d$depth_scaled), length.out = 100))
nd$year <- 2011L
p <- predict(m_gam, newdata = nd, se_fit = TRUE, re_form = NA)
ggplot(p, aes(depth_scaled, exp(est),
  ymin = exp(est - 1.96 * est_se), ymax = exp(est + 1.96 * est_se))) +
  geom_line() + geom_ribbon(alpha = 0.4)

# Forecasting ----------------------------------------------------------
mesh <- make_mesh(d, c("X", "Y"), cutoff = 15)

unique(d$year)
m <- sdmTMB(
  data = d, formula = density ~ 1,
  spatiotemporal = "AR1", # using an AR1 to have something to forecast with
  extra_time = 2019L, # `L` for integer to match our data
  spatial = "off",
  time = "year", mesh = mesh, family = tweedie(link = "log")
)

# Add a year to our grid:
grid2019 <- qcs_grid_2011[qcs_grid_2011$year == max(qcs_grid_2011$year), ]
grid2019$year <- 2019L # `L` because `year` is an integer in the data
qcsgrid_forecast <- rbind(qcs_grid_2011, grid2019)

predictions <- predict(m, newdata = qcsgrid_forecast)
plot_map(predictions, exp(est)) +
  scale_fill_viridis_c(trans = "log10")
plot_map(predictions, epsilon_st) +
  scale_fill_gradient2()

# Estimating local trends ----------------------------------------------

d <- pcod
d$year_scaled <- as.numeric(scale(d$year))
mesh <- make_mesh(pcod, c("X", "Y"), cutoff = 25)
m <- sdmTMB(data = d, formula = density ~ depth_scaled + depth_scaled2,
  mesh = mesh, family = tweedie(link = "log"),
  spatial_varying = ~ 0 + year_scaled, time = "year", spatiotemporal = "off")
```

```
nd <- replicate_df(qcs_grid, "year", unique(pcod$year))
nd$year_scaled <- (nd$year - mean(d$year)) / sd(d$year)
p <- predict(m, newdata = nd)

plot_map(subset(p, year == 2003), zeta_s_year_scaled) + # pick any year
  ggtitle("Spatial slopes") +
  scale_fill_gradient2()

plot_map(p, est_rf) +
  ggtitle("Random field estimates") +
  scale_fill_gradient2()

plot_map(p, exp(est_non_rf)) +
  ggtitle("Prediction (fixed effects only)") +
  scale_fill_viridis_c(trans = "sqrt")

plot_map(p, exp(est)) +
  ggtitle("Prediction (fixed effects + all random effects)") +
  scale_fill_viridis_c(trans = "sqrt")
```

`project` *Project from an* **sdmTMB** *model using simulation*

Description

[Experimental]

The function enables projecting forward in time from an **sdmTMB** model using a simulation approach for computational efficiency. This can be helpful for calculating predictive intervals for long projections where including those time elements in `extra_time` during model estimation can be slow.

Inspiration for this approach comes from the **VAST** function `project_model()`.

Usage

```
project(
  object,
  newdata,
  nsim = 1,
  uncertainty = c("both", "random", "none"),
  silent = FALSE,
  sims_var = "eta_i",
  sim_re = c(0, 1, 0, 0, 1, 0),
  return_tmb_report = FALSE,
  ...
)
```

Arguments

object	A fitted model from sdmTMB().
newdata	A new data frame to predict on. Should contain both historical and any new time elements to predict on.
nsim	Number of simulations.
uncertainty	How to sample uncertainty for the fitted parameters: "both" for the joint fixed and random effect precision matrix, "random" for the random effect precision matrix (holding the fixed effects at their MLE), or "none" for neither.
silent	Silent?
sims_var	Element to extract from the **TMB** report. Also see return_tmb_report.
sim_re	A vector of 0s and 1s representing which random effects to simulate in the projection. Generally, leave this untouched. Order is: spatial fields, spatiotemporal fields, spatially varying coefficient fields, random intercepts, time-varying coefficients, smoothers. The default is to simulate spatiotemporal fields and time-varying coefficients, if present.
return_tmb_report	Return the **TMB** report from simulate()? This lets you parse out whatever elements you want from the simulation including grabbing multiple elements from one set of simulations. See examples.
...	Passed to predict.sdmTMB().

Value

Default: a list with elements est and epsilon_st (if spatiotemporal effects are present). Each list element includes a matrix with rows corresponding to rows in newdata and nsim columns. For delta models, the components are est1, est2, epsilon_st, and epsilon_st2 for the 1st and 2nd linear predictors. In all cases, these returned values are in *link* space.

If return_tmb_report = TRUE, a list of **TMB** reports from simulate(). Run names() on the output to see the options.

Author(s)

J.T. Thorson wrote the original version in the **VAST** package. S.C. Anderson wrote this version inspired by the **VAST** version with help from A.J. Allyn.

References

project_model() in the **VAST** package.

Examples

```
library(ggplot2)

mesh <- make_mesh(dogfish, c("X", "Y"), cutoff = 25)
historical_years <- 2004:2022
to_project <- 10
```

```
future_years <- seq(max(historical_years) + 1, max(historical_years) + to_project)
all_years <- c(historical_years, future_years)
proj_grid <- replicate_df(wcvi_grid, "year", all_years)

# we could fit our model like this, but for long projections, this becomes slow:
if (FALSE) {
  fit <- sdmTMB(
    catch_weight ~ 1,
    time = "year",
    offset = log(dogfish$area_swept),
    extra_time = all_years, #< note that all years here
    spatial = "on",
    spatiotemporal = "ar1",
    data = dogfish,
    mesh = mesh,
    family = tweedie(link = "log")
  )
}

# instead, we could fit our model like this and then take simulation draws
# from the projection time period:
fit2 <- sdmTMB(
  catch_weight ~ 1,
  time = "year",
  offset = log(dogfish$area_swept),
  extra_time = historical_years, #< does *not* include projection years
  spatial = "on",
  spatiotemporal = "ar1",
  data = dogfish,
  mesh = mesh,
  family = tweedie(link = "log")
)

# we will only use 20 `nsim` so this example runs quickly
# you will likely want many more (> 200) in practice so the result
# is relatively stable

set.seed(1)
out <- project(fit2, newdata = proj_grid, nsim = 20)
names(out)
est_mean <- apply(out$est, 1, mean) # summarize however you'd like
est_se <- apply(out$est, 1, sd)

# visualize:
proj_grid$est_mean <- est_mean
ggplot(subset(proj_grid, year > 2021), aes(X, Y, fill = est_mean)) +
  geom_raster() +
  facet_wrap(~year) +
  coord_fixed() +
  scale_fill_viridis_c() +
  ggtitle("Projection simulation (mean)")

# visualize the spatiotemporal random fields:
```

```
proj_grid$eps_mean <- apply(out$epsilon_st, 1, mean)
proj_grid$eps_se <- apply(out$epsilon_st, 1, sd)
ggplot(subset(proj_grid, year > 2021), aes(X, Y, fill = eps_mean)) +
  geom_raster() +
  facet_wrap(~year) +
  scale_fill_gradient2() +
  coord_fixed() +
  ggtitle("Projection simulation\n(spatiotemporal fields)")

ggplot(subset(proj_grid, year > 2021), aes(X, Y, fill = eps_se)) +
  geom_raster() +
  facet_wrap(~year) +
  scale_fill_viridis_c() +
  coord_fixed() +
  ggtitle("Projection simulation\n(spatiotemporal fields standard error)")
```

`replicate_df` *Replicate a prediction data frame over time*

Description

Useful for replicating prediction grids across time slices used in model fitting.

Usage

```
replicate_df(dat, time_name, time_values)
```

Arguments

`dat`	Data frame.
`time_name`	Name of time column in output.
`time_values`	Time values to replicate `dat` over.

Value

A data frame replicated over `time_values` with a new column based on `time_name`.

Examples

```
df <- data.frame(variable = c("a", "b"))
replicate_df(df, time_name = "year", time_values = 1:3)

head(qcs_grid)
nd <- replicate_df(qcs_grid, "year", unique(pcod$year))
head(nd)
table(nd$year)
```

`residuals.sdmTMB` *Residuals method for sdmTMB models*

Description

See the residual-checking vignette: `browseVignettes("sdmTMB")` or on the documentation site. See notes about types of residuals in 'Details' section below.

Usage

```
## S3 method for class 'sdmTMB'
residuals(
  object,
  type = c("mle-mvn", "mle-eb", "mle-mcmc", "response", "pearson", "deviance"),
  model = c(1, 2),
  mcmc_samples = NULL,
  qres_func = NULL,
  ...
)
```

Arguments

`object`	An `sdmTMB()` model.
`type`	Residual type. See details.
`model`	Which delta/hurdle model component?
`mcmc_samples`	A vector of MCMC samples of the linear predictor in link space. See the `predict_mle_mcmc()` function in the sdmTMBextra package.
`qres_func`	A custom quantile residuals function. Function should take the arguments `object, y, mu, ...` and return a vector of length `length(y)`.
`...`	Passed to custom `qres_func` function. Unused.

Details

Randomized quantile residuals:

`mle-mvn`, `mle-eb`, and `mle-mcmc` are all implementations of randomized quantile residuals (Dunn & Smyth 1996), which are also known as probability integral transform (PIT) residuals (Smith 1985). If the data are consistent with model assumptions, these residuals should be distributed as normal(0, 1). Randomization is added to account for integer or binary response observations. For example, for a Poisson observation likelihood with observations `y` and mean predictions `mu`, we would create randomized quantile residuals as:

```
a <- ppois(y - 1, mu)
b <- ppois(y, mu)
u <- runif(n = length(y), min = a, max = b)
qnorm(u)
```

Types of residuals:

Acronyms:

- EB: Empirical Bayes
- MCMC: Markov chain Monte Carlo
- MLE: Maximum Likelihood Estimate
- MVN: Multivariate normal

`mle-mvn`: Fixed effects are held at their MLEs and random effects are taken from a single approximate posterior sample. The "approximate" part refers to the sample being taken from the random effects' assumed MVN distribution. In practice, the sample is obtained based on the mode and Hessian of the random effects taking advantage of sparsity in the Hessian for computational efficiency. This sample is taken with `obj$MC()`, where `obj` is the **TMB** object created with `TMB::MakeADFun()`. See Waagepetersen (2006) and the description in the source code for the internal **TMB** function `TMB:::oneSamplePosterior()`. Residuals are converted to randomized quantile residuals as described above.

`mle-eb`: Fixed effects are held at their MLEs and random effects are taken as their EB estimates. These used to be the default residuals in **sdmTMB** (and were called `mle-laplace`). They are available for backwards compatibility and for research purposes but they are *not* recommended for checking goodness of fit. Residuals are converted to randomized quantile residuals as described above.

`mle-mcmc`: Fixed effects are held at their MLEs and random effects are taken from a single posterior sample obtained with MCMC. These are an excellent option since they make no assumption about the distribution of the random effects (compared to the `mle-mvn` option) but can be slow to obtain. See Waagepetersen (2006) and Thygesen et al. (2017). Residuals are converted to randomized quantile residuals as described above.

See the **sdmTMBextra** package for the function `predict_mle_mcmc()`, which can generate the MCMC samples to pass to the `mcmc_samples` argument. Ideally MCMC is run until convergence and then the last iteration can be used for residuals. The defaults may not be sufficient for many models.

`response`: These are simple observed minus predicted residuals.

`pearson`: These are Pearson residuals: response residuals scaled by the standard deviation. If weights are present, the residuals are then multiplied by sqrt(weights).

Value

A vector of residuals. Note that randomization from any single random effect posterior sample and from any randomized quantile routines will result in different residuals with each call. It is suggested to **set a randomization seed** and to not go "fishing" for the perfect residuals or to present all inspected residuals.

References

Dunn, P.K. & Smyth, G.K. (1996). Randomized Quantile Residuals. Journal of Computational and Graphical Statistics, 5, 236–244.

Smith, J.Q. (1985). Diagnostic checks of non-standard time series models. Journal of Forecasting, 4, 283–291.

Waagepetersen, R. (2006). A simulation-based goodness-of-fit test for random effects in generalized linear mixed models. Scandinavian Journal of Statistics, 33(4), 721-731.

Thygesen, U.H., Albertsen, C.M., Berg, C.W., Kristensen, K., and Nielsen, A. 2017. Validation of ecological state space models using the Laplace approximation. Environ Ecol Stat 24(2): 317–339. doi:10.1007/s1065101703724

Rufener, M.-C., Kristensen, K., Nielsen, J.R., and Bastardie, F. 2021. Bridging the gap between commercial fisheries and survey data to model the spatiotemporal dynamics of marine species. Ecological Applications. e02453. doi:10.1002/eap.2453

See Also

simulate.sdmTMB(), dharma_residuals()

Examples

```
mesh <- make_mesh(pcod_2011, c("X", "Y"), cutoff = 10)
fit <- sdmTMB(
  present ~ as.factor(year) + poly(depth, 2),
  data = pcod_2011, mesh = mesh,
  family = binomial()
)

# the default "mle-mvn" residuals use fixed effects at their MLE and a
# single sample from the approximate random effect posterior:
set.seed(9283)
r <- residuals(fit, type = "mle-mvn")
qqnorm(r)
abline(0, 1)

# response residuals will be not be normally distributed unless
# the family is Gaussian:
r <- residuals(fit, type = "response")
qqnorm(r)
abline(0, 1)

# "mle-eb" are quick but are not expected to be N(0, 1); not recommended:
set.seed(2321)
r <- residuals(fit, type = "mle-eb")
qqnorm(r)
abline(0, 1)

# see also "mle-mcmc" residuals with the help of the sdmTMBextra package
# we can fake them here by taking a single sample from the joint precision
# matrix and pretending they are MCMC samples:
set.seed(82728)
p <- predict(fit, nsim = 1) # pretend these are from sdmTMBextra::predict_mle_mcmc()
r <- residuals(fit, mcmc_samples = p)
qqnorm(r)
abline(0, 1)
```

run_extra_optimization *Run extra optimization on an already fitted object*

Description

[Experimental]

Usage

```
run_extra_optimization(object, nlminb_loops = 0, newton_loops = 1)
```

Arguments

object	An object from `sdmTMB()`.
nlminb_loops	How many extra times to run `stats::nlminb()` optimization. Sometimes restarting the optimizer at the previous best values aids convergence.
newton_loops	How many extra Newton optimization loops to try with `stats::optimHess()`. Sometimes aids convergence.

Value

An updated model fit of class `sdmTMB`.

Examples

```
# Run extra optimization steps to help convergence:
# (Not typically needed)
fit <- sdmTMB(density ~ 0 + poly(depth, 2) + as.factor(year),
  data = pcod_2011, mesh = pcod_mesh_2011, family = tweedie())
fit_1 <- run_extra_optimization(fit, newton_loops = 1)
max(fit$gradients)
max(fit_1$gradients)
```

sanity *Sanity check of an sdmTMB model*

Description

Sanity check of an sdmTMB model

Usage

```
sanity(object, big_sd_log10 = 2, gradient_thresh = 0.001, silent = FALSE)
```

Arguments

object	Fitted model from sdmTMB().
big_sd_log10	Value to check size of standard errors against. A value of 2 would indicate that standard errors greater than 10^2 (i.e., 100) should be flagged.
gradient_thresh	Gradient threshold to issue warning.
silent	Logical: suppress messages? Useful to set to TRUE if running large numbers of models and just interested in returning sanity list objects.

Details

If object is NA, NULL, or of class "try-error", sanity() will return FALSE. This is to facilitate using sanity() on models with try() or tryCatch(). See the examples section.

Value

An invisible named list of checks.

Examples

```
fit <- sdmTMB(
  present ~ s(depth),
  data = pcod_2011, mesh = pcod_mesh_2011,
  family = binomial()
)
sanity(fit)

s <- sanity(fit)
s

# If fitting many models in a loop, you may want to wrap
# sdmTMB() in try() to handle errors. sanity() will take an object
# of class "try-error" and return FALSE.
# Here, we will use stop() to simulate a failed sdmTMB() fit:
failed_fit <- try(stop())
s2 <- sanity(failed_fit)
all(unlist(s))
all(unlist(s2))
```

---

sdmTMB *Fit a spatial or spatiotemporal GLMM with TMB*

Description

Fit a spatial or spatiotemporal generalized linear mixed effects model (GLMM) with the TMB (Template Model Builder) R package. Spatial and spatiotemporal random fields are approximated using the SPDE (stochastic partial differential equation) approach, which allows for efficient modeling of data that are correlated in space and/or time. See the model description vignette for details.

Usage

```
sdmTMB(
  formula,
  data,
  mesh,
  time = NULL,
  family = gaussian(link = "identity"),
  spatial = c("on", "off"),
  spatiotemporal = c("iid", "ar1", "rw", "off"),
  share_range = TRUE,
  time_varying = NULL,
  time_varying_type = c("rw", "rw0", "ar1"),
  spatial_varying = NULL,
  weights = NULL,
  offset = NULL,
  extra_time = NULL,
  reml = FALSE,
  silent = TRUE,
  anisotropy = FALSE,
  control = sdmTMBcontrol(),
  priors = sdmTMBpriors(),
  knots = NULL,
  bayesian = FALSE,
  previous_fit = NULL,
  do_fit = TRUE,
  do_index = FALSE,
  predict_args = NULL,
  index_args = NULL,
  experimental = NULL
)
```

Arguments

formula	Model formula. IID random intercepts and slopes are possible using **lme4** syntax, e.g., + (1 | g) or + (0 + depth | g) or + (1 + depth | g) where g is a column of class character or factor representing groups. Penalized splines are possible via **mgcv** with s(). Optionally a list for delta (hurdle) models. See examples and details below.
data	A data frame.
mesh	An object from make_mesh().
time	An optional time column name (as character). Can be left as NULL for a model with only spatial random fields; however, if the data are actually spatiotemporal and you wish to use get_index() or get_cog() downstream, supply the time argument.
family	The family and link. Supports gaussian(), Gamma(), binomial(), poisson(), Beta(), betabinomial(), nbinom2(), truncated_nbinom2(), nbinom1(), truncated_nbinom1(), censored_poisson(), gamma_mix(), lognormal_mix(), student(), tweedie(),

and gengamma(). Delta/hurdle models (for zero-inflated data) include: delta_beta(), delta_gamma(), delta_gamma_mix(), delta_lognormal_mix(), delta_lognormal(), and delta_truncated_nbinom2(). See the delta-model vignette for details. For binomial family options, see 'Binomial families' in the Details section below.

spatial
: Estimate spatial random fields? Options are 'on' / 'off' or TRUE / FALSE. Optionally, a list for delta models, e.g. list('on', 'off').

spatiotemporal
: Estimate the spatiotemporal random fields as 'iid' (independent and identically distributed; default), stationary 'ar1' (first-order autoregressive), a random walk ('rw'), or fixed at 0 'off'. Will be set to 'off' if time = NULL. If a delta model, can be a list. E.g., list('off', 'ar1'). Guidance: Use 'iid' if temporal correlation is negligible or already accounted for in fixed effects; 'ar1' if correlation between consecutive time steps decays gradually; 'rw' if changes between time steps are cumulative (each step builds on the last). If the AR1 correlation coefficient (rho) is estimated close to 1 (say > 0.99), consider switching to 'rw'. See the model description vignette for mathematical details. Capitalization is ignored. TRUE gets converted to 'iid' and FALSE gets converted to 'off'.

share_range
: Logical: estimate a shared spatial and spatiotemporal range parameter (TRUE, default) or independent range parameters (FALSE). If a delta model, can be a list. E.g., list(TRUE, FALSE).

time_varying
: An optional one-sided formula describing covariates that should be modelled as a time-varying process. Set the type of process with time_varying_type. See the help for time_varying_type for warnings about modelling the first time step. Structure shared in delta models.

time_varying_type
: Type of time-varying process to apply to time_varying formula. Options: 'rw' (random walk, default), 'rw0' (random walk with mean-zero prior on first time step), or 'ar1' (autoregressive, for coefficients that fluctuate around a mean). For 'rw0' and 'ar1', the coefficient starts at zero in the first time step. For 'rw' (default), the first time step is estimated separately—in this case, avoid including the same covariates in both formula and time_varying to prevent non-identifiability (use ~ 0 or ~ -1 in at least one). Structure shared in delta models.

spatial_varying
: An optional one-sided formula of coefficients that should vary in space as random fields. Allows the effect of a covariate to differ spatially. You likely want to include the same variable as a fixed effect in formula to estimate the average effect—the spatial field then represents deviations from that average (since it has mean zero). For example, use formula = y ~ depth and spatial_varying = ~ 0 + depth to model an average depth effect plus spatially varying deviations. If a (scaled) time column is used, this creates a local-time-trend model. See doi:10.1111/ecog.05176 and the spatial trends vignette. Predictors should usually be centered to have mean zero and standard deviation approximately 1. **The spatial intercept is controlled by the** spatial **argument**; set spatial = 'on' or 'off' to include or exclude it. For factor predictors, if spatial_varying excludes the intercept (~ 0 or ~ -1), set spatial = 'off' to match. Structure must be shared in delta models.

`weights` A numeric vector representing optional likelihood weights for the conditional model. Implemented as in **glmmTMB**: weights do not have to sum to one and are not internally modified. Can also be used for trials with the binomial family; the `weights` argument needs to be a vector and not a name of the variable in the data frame. See the Details section below.

`offset` A numeric vector representing the model offset *or* a character value representing the column name of the offset. In delta/hurdle models, this applies only to the positive component. Usually a log transformed variable.

`extra_time` Optional extra time slices (e.g., years) to include for interpolation or forecasting with the predict function. See the Details section below.

`reml` Logical: use REML (restricted maximum likelihood) estimation rather than maximum likelihood? REML accounts for uncertainty in estimating fixed effects and can reduce bias in variance parameter estimates, but prevents likelihood-based model comparison (e.g., AIC) between models with different fixed effects. Use `TRUE` if your focus is on random effect variance parameters; use `FALSE` (default) if comparing models with different fixed effects or performing index standardization.

`silent` Silent or include optimization details? Helpful to set to `FALSE` for models that take a while to fit.

`anisotropy` Logical: allow for anisotropy (spatial correlation that is directionally dependent)? See `plot_anisotropy()`. Must be shared across delta models.

`control` Optimization control options via `sdmTMBcontrol()`.

`priors` Optional penalties/priors via `sdmTMBpriors()`. Must currently be shared across delta models.

`knots` Optional named list containing knot values to be used for basis construction of smoothing terms. See `mgcv::gam()` and `mgcv::gamm()`. E.g., `s(x, bs = 'cc', k = 4), knots = list(`

`bayesian` Logical indicating if the model will be passed to **tmbstan**. If `TRUE`, Jacobian adjustments are applied to account for parameter transformations when priors are applied.

`previous_fit` A previously fitted sdmTMB model to initialize the optimization with. Can greatly speed up fitting. Note that the model must be set up *exactly* the same way. However, the data and `weights` arguments can change, which can be useful for cross-validation.

`do_fit` Fit the model (`TRUE`) or return the processed data without fitting (`FALSE`)?

`do_index` Do index standardization calculations while fitting? Saves memory and time when working with large datasets or projection grids since the TMB object doesn't have to be rebuilt with `predict.sdmTMB()` and `get_index()`. If `TRUE`, then `predict_args` must have a `newdata` element supplied and `area` can be supplied to `index_args`. Most users can ignore this option. The fitted object can be passed directly to `get_index()`.

`predict_args` A list of arguments to pass to `predict.sdmTMB()` **if** `do_index = TRUE`. Most users can ignore this option.

`index_args` A list of arguments to pass to `get_index()` **if** `do_index = TRUE`. Currently, only `area` is supported. Bias correction can be done when calling `get_index()` on the resulting fitted object. Most users can ignore this option.

`experimental` A named list for esoteric or in-development options. Here be dragons.

Details

Model description

sdmTMB fits GLMMs with spatial and/or spatiotemporal random fields, which account for correlation in the data due to spatial proximity, or alternatively, latent spatial and spatiotemporal effects. Spatial fields represent consistent spatial patterns, while spatiotemporal fields represent spatial patterns that vary over time. See the model description vignette for mathematical details and the paper: doi:10.18637/jss.v115.i02

Binomial families

Following the structure of `stats::glm()` and **glmmTMB**, a binomial family can be specified in one of 4 ways: (1) the response may be a factor (and the model classifies the first level versus all others), (2) the response may be binomial (0/1), (3) the response can be a matrix of form `cbind(success, failure)`, and (4) the response may be the observed proportions, and the 'weights' argument is used to specify the Binomial size (N) parameter (`prob ~ ..., weights = N`).

Smooth terms

Smooth terms can be included following GAMs (generalized additive models) using `+ s(x)`, which implements a smooth from `mgcv::s()`. **sdmTMB** uses penalized smooths, constructed via `mgcv::smooth2random()`. This is a similar approach implemented in **gamm4** and **brms**, among other packages. Within these smooths, the same syntax commonly used in `mgcv::s()` or `mgcv::t2()` can be applied, e.g. 2-dimensional smooths may be constructed with `+ s(x, y)` or `+ t2(x, y)`; smooths can be specific to various factor levels, `+ s(x, by = group)`; the basis function dimensions may be specified, e.g. `+ s(x, k = 4)`; and various types of splines may be constructed such as cyclic splines to model seasonality (perhaps with the `knots` argument also be supplied).

Threshold models

A linear break-point relationship for a covariate can be included via `+ breakpt(variable)` in the formula, where `variable` is a single covariate corresponding to a column in `data`. In this case, the relationship is linear up to a point and then constant (hockey-stick shaped).

Similarly, a logistic-function threshold model can be included via `+ logistic(variable)`. This option models the relationship as a logistic function of the 50% and 95% values. This is similar to length- or size-based selectivity in fisheries, and is parameterized by the points at which f(x) = 0.5 or 0.95. See the threshold vignette.

Note that only a single threshold covariate can be included and the same covariate is included in both components for the delta families.

Extra time: forecasting or interpolating

Extra time slices (e.g., years) can be included for interpolation or forecasting with the predict function via the `extra_time` argument. The predict function requires all time slices to be defined when fitting the model to ensure the various time indices are set up correctly. Be careful if including extra time slices that the model remains identifiable. For example, including `+ as.factor(year)` in `formula` will render a model with no data to inform the expected value in a missing year. `sdmTMB()` makes no attempt to determine if the model makes sense for forecasting or interpolation. The options `time_varying`, `spatiotemporal = "rw"`, `spatiotemporal = "ar1"`, or a smoother on the time column provide mechanisms to predict over missing time slices with process error.

`extra_time` can also be used to fill in missing time steps for the purposes of a random walk or AR(1) process if the gaps between time steps are uneven.

extra_time can include only extra time steps or all time steps including those found in the fitted data. This latter option may be simpler.

Regularization and priors

You can achieve regularization via penalties (priors) on the fixed effect parameters. See sdmTMBpriors(). You can fit the model once without penalties and look at the output of print(your_model) or tidy(your_model) or fit the model with do_fit = FALSE and inspect head(your_modeltmb_dataX_ij[[1]]) if you want to see how the formula is translated to the fixed effect model matrix. Also see the Bayesian vignette.

Delta/hurdle models

Delta models (also known as hurdle models) can be fit as two separate models or at the same time by using an appropriate delta family. E.g.: delta_gamma(), delta_beta(), delta_lognormal(), and delta_truncated_nbinom2(). If fit with a delta family, by default the formula, spatial, and spatiotemporal components are shared. Some elements can be specified independently for the two models using a list format. These include formula, spatial, spatiotemporal, and share_range. The first element of the list is for the binomial component and the second element is for the positive component (e.g., Gamma). Other elements must be shared for now (e.g., spatially varying coefficients, time-varying coefficients). Furthermore, there are currently limitations if specifying two formulas as a list: the two formulas cannot have smoothers or threshold effects. For now, these must be specified through a single formula that is shared across the two models.

The main advantage of specifying such models using a delta family (compared to fitting two separate models) is (1) coding simplicity and (2) calculation of uncertainty on derived quantities such as an index of abundance with get_index() using the generalized delta method within TMB. Also, selected parameters can be shared across the models.

See the delta-model vignette.

Index standardization

For index standardization, you may wish to include 0 + as.factor(year) (or whatever the time column is called) in the formula. See a basic example of index standardization in the relevant package vignette. You will need to specify the time argument. See get_index().

Value

An object (list) of class sdmTMB. Useful elements include:

- sd_report: output from TMB::sdreport()
- gradients: marginal log likelihood gradients with respect to each fixed effect
- model: output from stats::nlminb()
- data: the fitted data
- spde: the object that was supplied to the mesh argument
- family: the family object, which includes the inverse link function as family$linkinv()
- tmb_params: The parameters list passed to TMB::MakeADFun()
- tmb_map: The 'map' list passed to TMB::MakeADFun()
- tmb_data: The data list passed to TMB::MakeADFun()
- tmb_obj: The TMB object created by TMB::MakeADFun()

References

Main reference introducing the package to cite when using sdmTMB:

Anderson, S.C., E.J. Ward, P.A. English, L.A.K. Barnett., J.T. Thorson. 2025. sdmTMB: an R package for fast, flexible, and user-friendly generalized linear mixed effects models with spatial and spatiotemporal random fields. Journal of Statistical Software. 115(2):1–46. doi:10.18637/jss.v115.i02.

Reference for local trends:

Barnett, L.A.K., E.J. Ward, S.C. Anderson. 2021. Improving estimates of species distribution change by incorporating local trends. Ecography. 44(3):427-439. doi:10.1111/ecog.05176.

Further explanation of the model and application to calculating climate velocities:

English, P., E.J. Ward, C.N. Rooper, R.E. Forrest, L.A. Rogers, K.L. Hunter, A.M. Edwards, B.M. Connors, S.C. Anderson. 2021. Contrasting climate velocity impacts in warm and cool locations show that effects of marine warming are worse in already warmer temperate waters. Fish and Fisheries. 23(1) 239-255. doi:10.1111/faf.12613.

Discussion of and illustration of some decision points when fitting these models:

Commander, C.J.C., L.A.K. Barnett, E.J. Ward, S.C. Anderson, T.E. Essington. 2022. The shadow model: how and why small choices in spatially explicit species distribution models affect predictions. PeerJ 10: e12783. doi:10.7717/peerj.12783.

Application and description of threshold/break-point models:

Essington, T.E., S.C. Anderson, L.A.K. Barnett, H.M. Berger, S.A. Siedlecki, E.J. Ward. 2022. Advancing statistical models to reveal the effect of dissolved oxygen on the spatial distribution of marine taxa using thresholds and a physiologically based index. Ecography. 2022: e06249 doi:10.1111/ecog.06249.

Application to fish body condition:

Lindmark, M., S.C. Anderson, M. Gogina, M. Casini. Evaluating drivers of spatiotemporal individual condition of a bottom-associated marine fish. bioRxiv 2022.04.19.488709. doi:10.1101/2022.04.19.488709.

Several sections of the original TMB model code were adapted from the VAST R package:

Thorson, J.T. 2019. Guidance for decisions using the Vector Autoregressive Spatio-Temporal (VAST) package in stock, ecosystem, habitat and climate assessments. Fish. Res. 210:143–161. doi:10.1016/j.fishres.2018.10.013.

Code for the `family` *R-to-TMB implementation, selected parameterizations of the observation likelihoods, general package structure inspiration, and the idea behind the TMB prediction approach were adapted from the glmmTMB R package:*

Brooks, M.E., K. Kristensen, K.J. van Benthem, A. Magnusson, C.W. Berg, A. Nielsen, H.J. Skaug, M. Maechler, B.M. Bolker. 2017. glmmTMB Balances Speed and Flexibility Among Packages for Zero-inflated Generalized Linear Mixed Modeling. The R Journal, 9(2):378-400. doi:10.32614/rj2017066.

Implementation of geometric anisotropy with the SPDE and use of random field GLMMs for index standardization:

Thorson, J.T., A.O. Shelton, E.J. Ward, H.J. Skaug. 2015. Geostatistical delta-generalized linear mixed models improve precision for estimated abundance indices for West Coast groundfishes. ICES J. Mar. Sci. 72(5): 1297–1310. doi:10.1093/icesjms/fsu243.

Examples

```
library(sdmTMB)

# Build a mesh to implement the SPDE approach:
mesh <- make_mesh(pcod_2011, c("X", "Y"), cutoff = 20)

# - this example uses a fairly coarse mesh so these examples run quickly
# - 'cutoff' is the minimum distance between mesh vertices in units of the
#   x and y coordinates
# - 'cutoff = 10' might make more sense in applied situations for this dataset
# - or build any mesh in 'fmesher' and pass it to the 'mesh' argument in make_mesh()`
# - the mesh is not needed if you will be turning off all
#   spatial/spatiotemporal random fields

# Quick mesh plot:
plot(mesh)

# Fit a Tweedie spatial random field GLMM with a smoother for depth:
fit <- sdmTMB(
  density ~ s(depth),
  data = pcod_2011, mesh = mesh,
  family = tweedie(link = "log")
)
fit

# Extract coefficients:
tidy(fit, conf.int = TRUE)
tidy(fit, effects = "ran_par", conf.int = TRUE)

# Perform several 'sanity' checks:
sanity(fit)

# Predict on the fitted data; see ?predict.sdmTMB
p <- predict(fit)

# Predict on new data:
p <- predict(fit, newdata = qcs_grid)
head(p)


# Visualize the depth effect with ggeffects:
ggeffects::ggpredict(fit, "depth [all]") |> plot()

# Visualize depth effect with visreg: (see ?visreg_delta)
visreg::visreg(fit, xvar = "depth") # link space; randomized quantile residuals
visreg::visreg(fit, xvar = "depth", scale = "response")
visreg::visreg(fit, xvar = "depth", scale = "response", gg = TRUE, rug = FALSE)

# Add spatiotemporal random fields:
fit <- sdmTMB(
  density ~ 0 + as.factor(year),
  time = "year", #<
```

```
  data = pcod_2011, mesh = mesh,
  family = tweedie(link = "log")
)
fit

# Make the fields AR1:
fit <- sdmTMB(
  density ~ s(depth),
  time = "year",
  spatial = "off",
  spatiotemporal = "ar1", #<
  data = pcod_2011, mesh = mesh,
  family = tweedie(link = "log")
)
fit

# Make the fields a random walk:
fit <- sdmTMB(
  density ~ s(depth),
  time = "year",
  spatial = "off",
  spatiotemporal = "rw", #<
  data = pcod_2011, mesh = mesh,
  family = tweedie(link = "log")
)
fit

# Depth smoothers by year:
fit <- sdmTMB(
  density ~ s(depth, by = as.factor(year)), #<
  time = "year",
  spatial = "off",
  spatiotemporal = "rw",
  data = pcod_2011, mesh = mesh,
  family = tweedie(link = "log")
)
fit

# 2D depth-year smoother:
fit <- sdmTMB(
  density ~ s(depth, year), #<
  spatial = "off",
  data = pcod_2011, mesh = mesh,
  family = tweedie(link = "log")
)
fit

# Turn off spatial random fields:
fit <- sdmTMB(
  present ~ poly(log(depth)),
  spatial = "off", #<
  data = pcod_2011, mesh = mesh,
  family = binomial()
```

```
)
fit

# Which, matches glm():
fit_glm <- glm(
  present ~ poly(log(depth)),
  data = pcod_2011,
  family = binomial()
)
summary(fit_glm)
AIC(fit, fit_glm)

# Delta/hurdle binomial-Gamma model:
fit_dg <- sdmTMB(
  density ~ poly(log(depth), 2),
  data = pcod_2011, mesh = mesh,
  spatial = "off",
  family = delta_gamma() #<
)
fit_dg

# Delta model with different formulas and spatial structure:
fit_dg <- sdmTMB(
  list(density ~ depth_scaled, density ~ poly(depth_scaled, 2)), #<
  data = pcod_2011, mesh = mesh,
  spatial = list("off", "on"), #<
  family = delta_gamma()
)
fit_dg

# Delta/hurdle truncated NB2:
pcod_2011$count <- round(pcod_2011$density)
fit_nb2 <- sdmTMB(
  count ~ s(depth),
  data = pcod_2011, mesh = mesh,
  spatial = "off",
  family = delta_truncated_nbinom2() #<
)
fit_nb2

# Regular NB2:
fit_nb2 <- sdmTMB(
  count ~ s(depth),
  data = pcod_2011, mesh = mesh,
  spatial = "off",
  family = nbinom2() #<
)
fit_nb2

# IID random intercepts by year:
pcod_2011$fyear <- as.factor(pcod_2011$year)
fit <- sdmTMB(
  density ~ s(depth) + (1 | fyear), #<
```

```
  data = pcod_2011, mesh = mesh,
  family = tweedie(link = "log")
)
fit

# IID random slopes and intercepts (implicit) by year:
fit <- sdmTMB(
  density ~ (depth | fyear), #<
  data = pcod_2011, mesh = mesh,
  family = tweedie(link = "log")
)

# Spatially varying coefficient of year:
pcod_2011$year_scaled <- as.numeric(scale(pcod_2011$year))
fit <- sdmTMB(
  density ~ year_scaled,
  spatial_varying = ~ 0 + year_scaled, #<
  data = pcod_2011, mesh = mesh, family = tweedie(), time = "year"
)
fit

# Time-varying effects of depth and depth squared:
fit <- sdmTMB(
  density ~ 0 + as.factor(year),
  time_varying = ~ 0 + depth_scaled + depth_scaled2, #<
  data = pcod_2011, time = "year", mesh = mesh,
  family = tweedie()
)
print(fit)
# Extract values:
est <- as.list(fit$sd_report, "Estimate")
se <- as.list(fit$sd_report, "Std. Error")
est$b_rw_t[, , 1]
se$b_rw_t[, , 1]

# Linear break-point effect of depth:
fit <- sdmTMB(
  density ~ breakpt(depth_scaled), #<
  data = pcod_2011,
  mesh = mesh,
  family = tweedie()
)
fit
```

`sdmTMBcontrol` *Optimization control options*

Description

`sdmTMB()` and `stats::nlminb()` control options.

Usage

```
sdmTMBcontrol(
  eval.max = 2000L,
  iter.max = 1000L,
  normalize = FALSE,
  nlminb_loops = 1L,
  newton_loops = 1L,
  getsd = TRUE,
  quadratic_roots = FALSE,
  start = NULL,
  map = NULL,
  lower = NULL,
  upper = NULL,
  censored_upper = NULL,
  multiphase = TRUE,
  profile = FALSE,
  get_joint_precision = TRUE,
  parallel = getOption("sdmTMB.cores", 1L),
  suppress_nlminb_warnings = TRUE,
  collapse_spatial_variance = FALSE,
  collapse_threshold = 0.01,
  ...
)
```

Arguments

eval.max	Maximum number of evaluations of the objective function allowed.
iter.max	Maximum number of iterations allowed.
normalize	Logical: use TMB::normalize() to normalize the process likelihood using the Laplace approximation? Can result in a substantial speed boost in some cases. This used to default to FALSE prior to May 2021. Currently not working for models fit with REML or random intercepts.
nlminb_loops	How many times to run stats::nlminb() optimization. Sometimes restarting the optimizer at the previous best values aids convergence. If the maximum gradient is still too large, try increasing this to 2.
newton_loops	How many Newton optimization steps to try after running stats::nlminb(). This sometimes aids convergence by further reducing the log-likelihood gradient with respect to the fixed effects. This calculates the Hessian at the current MLE with stats::optimHess() using a finite-difference approach and uses this to update the fixed effect estimates.
getsd	Logical indicating whether to call TMB::sdreport().
quadratic_roots	Experimental feature for internal use right now; may be moved to a branch. Logical: should quadratic roots be calculated? Note: on the sdmTMB side, the first two coefficients are used to generate the quadratic parameters. This means that if you want to generate a quadratic profile for depth, and depth and depth^2

are part of your formula, you need to make sure these are listed first and that an intercept isn't included. For example, `formula = cpue ~ 0 + depth + depth2 + as.factor(year)`.

`start` A named list specifying the starting values for parameters. You can see the necessary structure by fitting the model once and inspecting `your_model$tmb_obj$env$parList()`. Elements of `start` that are specified will replace the default starting values.

`map` A named list with factor `NA`s specifying parameter values that should be fixed at a constant value. See the documentation in `TMB::MakeADFun()`. This should usually be used with `start` to specify the fixed value.

`lower` An optional named list of lower bounds within the optimization. Parameter vectors with the same name (e.g., `b_j` or `ln_kappa` in some cases) can be specified as a numeric vector. E.g. `lower = list(b_j = c(-5, -5))`. Note that `stats::optimHess()` does not implement lower and upper bounds, so you must set `newton_loops = 0` if setting limits.

`upper` An optional named list of upper bounds within the optimization.

`censored_upper` An optional vector of upper bounds for `sdmTMBcontrol()`. Values of `NA` indicate an unbounded right-censored to distribution, values greater that the observation indicate and upper bound, and values equal to the observation indicate no censoring.

`multiphase` Logical: estimate the fixed and random effects in phases? Phases are usually faster and more stable.

`profile` Logical: should population-level/fixed effects be profiled out of the likelihood? These are then appended to the random effects vector without the Laplace approximation. See `TMB::MakeADFun()`. *This can dramatically speed up model fit if there are many fixed effects but is experimental at this stage.*

`get_joint_precision`
Logical. Passed to `getJointPrecision` in `TMB::sdreport()`. Must be `TRUE` to use simulation-based methods in `predict.sdmTMB()` or `[get_index_sims()]`. If not needed, setting this `FALSE` will reduce object size.

`parallel` Argument currently ignored. For parallel processing with 3 cores, as an example, use `TMB::openmp(n = 3, DLL = "sdmTMB")`. But be careful, because it's not always faster with more cores and there is definitely an upper limit.

`suppress_nlminb_warnings`
Suppress uninformative warnings from `stats::nlminb()` arising when a function evaluation is `NA`, which are then replaced with `Inf` and avoided during estimation?

`collapse_spatial_variance`
Logical: should spatial and/or spatiotemporal random fields be automatically dropped if their estimated standard deviation is effectively zero (i.e., below `collapse_threshold`)? This helps prevent overfitting and numerical instability when the data provide little evidence for spatial or spatiotemporal variation. I.e., when the variance parameter is estimated on or near the boundary of zero. When enabled, the model will be automatically refitted via `update.sdmTMB()` with the corresponding field(s) disabled. This adds a computational cost (a single model refit if collapsing occurs) but can yield a simpler, more stable model and more reliable inference. Default is `FALSE` for backwards compatibility.

collapse_threshold
: Numeric: the standard deviation threshold below which random fields are considered to be collapsing to zero. Only used when `collapse_spatial_variance = TRUE`. Values are on the standard deviation scale (i.e., square root of variance). Default is 0.01.

...
: Anything else. See the 'Control parameters' section of `stats::nlminb()`.

Details

Usually used within `sdmTMB()`. For example:

```
sdmTMB(..., control = sdmTMBcontrol(newton_loops = 2))
```

Value

A list of control arguments

Examples

```
sdmTMBcontrol()
```

---

sdmTMBpriors *Prior distributions*

Description

[Experimental]

Optional priors/penalties on model parameters. This results in penalized likelihood within TMB or can be used as priors if the model is passed to **tmbstan** (see the Bayesian vignette).

Note that Jacobian adjustments are only made if `bayesian = TRUE` when the `sdmTMB()` model is fit. I.e., the final model will be fit with **tmbstan** and priors are specified then `bayesian` should be set to `TRUE`. Otherwise, leave `bayesian = FALSE`.

`pc_matern()` is the Penalized Complexity prior for the Matern covariance function.

Usage

```
sdmTMBpriors(
  matern_s = pc_matern(range_gt = NA, sigma_lt = NA),
  matern_st = pc_matern(range_gt = NA, sigma_lt = NA),
  phi = halfnormal(NA, NA),
  ar1_rho = normal(NA, NA),
  tweedie_p = normal(NA, NA),
  b = normal(NA, NA),
  sigma_V = gamma_cv(NA, NA),
  threshold_breakpt_slope = normal(NA, NA),
  threshold_breakpt_cut = normal(NA, NA),
  threshold_logistic_s50 = normal(NA, NA),
```

```
  threshold_logistic_s95 = normal(NA, NA),
  threshold_logistic_smax = normal(NA, NA)
)

normal(location = 0, scale = 1)

halfnormal(location = 0, scale = 1)

gamma_cv(location, cv)

mvnormal(location = 0, scale = diag(length(location)))

pc_matern(range_gt, sigma_lt, range_prob = 0.05, sigma_prob = 0.05)
```

Arguments

matern_s	A PC (Penalized Complexity) prior (pc_matern()) on the spatial random field Matérn parameters.
matern_st	Same as matern_s but for the spatiotemporal random field. Note that you will likely want to set share_fields = FALSE if you choose to set both a spatial and spatiotemporal Matérn PC prior since they both include a prior on the spatial range parameter.
phi	A halfnormal() prior for the dispersion parameter in the observation distribution.
ar1_rho	A normal() prior for the AR1 random field parameter. Note the parameter has support -1 < ar1_rho < 1.
tweedie_p	A normal() prior for the Tweedie power parameter. Note the parameter has support 1 < tweedie_p < 2 so choose a mean appropriately.
b	normal() priors for the main population-level 'beta' effects.
sigma_V	gamma_cv() priors for any time-varying parameter SDs.
threshold_breakpt_slope	A normal() prior for the slope of the linear (hockey stick) function.
threshold_breakpt_cut	A normal() prior for the cutoff of the linear (hockey stick) function.
threshold_logistic_s50	A normal() prior for the parameter at which f(x) = 0.5.
threshold_logistic_s95	A normal() prior for the parameter at which f(x) = 0.95.
threshold_logistic_smax	A normal() prior for the parameter at which f(x) is maximized.
location	Location parameter(s). Typically the mean.
scale	Scale parameter. For normal()/halfnormal(): standard deviation(s). For mvnormal(): variance-covariance matrix.
cv	Coefficient of variation (SD/mean).

range_gt	A value one expects the spatial or spatiotemporal range is **g**reater **t**han with 1 - range_prob probability.
sigma_lt	A value one expects the spatial or spatiotemporal marginal standard deviation (sigma_O or sigma_E internally) is **l**ess **t**han with 1 - sigma_prob probability.
range_prob	Probability. See description for range_gt.
sigma_prob	Probability. See description for sigma_lt.

Details

Meant to be passed to the priors argument in sdmTMB().

normal() and halfnormal() define normal and half-normal priors that, at this point, must have a location (mean) parameter of 0. halfnormal() is the same as normal() but can be used to make the syntax clearer. It is intended to be used for parameters that have support > 0.

See https://arxiv.org/abs/1503.00256 for a description of the PC prior for Gaussian random fields. Quoting the discussion (and substituting the argument names in pc_matern()): "In the simulation study we observe good coverage of the equal-tailed 95% credible intervals when the prior satisfies P(sigma > sigma_lt) = 0.05 and P(range < range_gt) = 0.05, where sigma_lt is between 2.5 to 40 times the true marginal standard deviation and range_gt is between 1/10 and 1/2.5 of the true range."

Keep in mind that the range is dependent on the units and scale of the coordinate system. In practice, you may choose to try fitting the model without a PC prior and then constraining the model from there. A better option would be to simulate from a model with a given range and sigma to choose reasonable values for the system or base the prior on knowledge from a model fit to a similar system but with more spatial information in the data.

Value

A named list with values for the specified priors.

References

Fuglstad, G.-A., Simpson, D., Lindgren, F., and Rue, H. (2016) Constructing Priors that Penalize the Complexity of Gaussian Random Fields. arXiv:1503.00256

Simpson, D., Rue, H., Martins, T., Riebler, A., and Sørbye, S. (2015) Penalising model component complexity: A principled, practical approach to constructing priors. arXiv:1403.4630

See Also

plot_pc_matern()

Examples

```
normal(0, 1)
halfnormal(0, 1)
gamma_cv(0.5, 0.2)
mvnormal(c(0, 0))
pc_matern(range_gt = 5, sigma_lt = 1)
plot_pc_matern(range_gt = 5, sigma_lt = 1)
```

```
d <- subset(pcod, year > 2011)
pcod_spde <- make_mesh(d, c("X", "Y"), cutoff = 30)

# - no priors on population-level effects (`b`)
# - halfnormal(0, 10) prior on dispersion parameter `phi`
# - Matern PC priors on spatial `matern_s` and spatiotemporal
#   `matern_st` random field parameters
m <- sdmTMB(density ~ s(depth, k = 3),
  data = d, mesh = pcod_spde, family = tweedie(),
  share_range = FALSE, time = "year",
  priors = sdmTMBpriors(
    phi = halfnormal(0, 10),
    matern_s = pc_matern(range_gt = 5, sigma_lt = 1),
    matern_st = pc_matern(range_gt = 5, sigma_lt = 1)
  )
)

# - no prior on intercept
# - normal(0, 1) prior on depth coefficient
# - no prior on the dispersion parameter `phi`
# - Matern PC prior
m <- sdmTMB(density ~ depth_scaled,
  data = d, mesh = pcod_spde, family = tweedie(),
  spatiotemporal = "off",
  priors = sdmTMBpriors(
    b = normal(c(NA, 0), c(NA, 1)),
    matern_s = pc_matern(range_gt = 5, sigma_lt = 1)
  )
)

# You get a prior, you get a prior, you get a prior!
# (except on the annual means; see the `NA`s)
m <- sdmTMB(density ~ 0 + depth_scaled + depth_scaled2 + as.factor(year),
  data = d, time = "year", mesh = pcod_spde, family = tweedie(link = "log"),
  share_range = FALSE, spatiotemporal = "AR1",
  priors = sdmTMBpriors(
    b = normal(c(0, 0, NA, NA, NA), c(2, 2, NA, NA, NA)),
    phi = halfnormal(0, 10),
    # tweedie_p = normal(1.5, 2),
    ar1_rho = normal(0, 1),
    matern_s = pc_matern(range_gt = 5, sigma_lt = 1),
    matern_st = pc_matern(range_gt = 5, sigma_lt = 1))
)
```

---

sdmTMB_cv *Cross validation with sdmTMB models*

Description

Performs k-fold or leave-future-out cross validation with sdmTMB models. Returns the sum of log likelihoods of held-out data (log predictive density), which can be used to compare models—higher values indicate better out-of-sample prediction. By default, creates folds randomly and stratified by time (set a seed for reproducibility), but folds can be manually assigned via `fold_ids`. See Ward and Anderson (2025) in the References and the cross-validation vignette.

Usage

```
sdmTMB_cv(
  formula,
  data,
  mesh_args,
  mesh = NULL,
  time = NULL,
  k_folds = 8,
  fold_ids = NULL,
  lfo = FALSE,
  lfo_forecast = 1,
  lfo_validations = 5,
  parallel = TRUE,
  use_initial_fit = FALSE,
  save_models = TRUE,
  future_globals = NULL,
  ...
)
```

Arguments

`formula`	Model formula.
`data`	A data frame.
`mesh_args`	Arguments for `make_mesh()`. If supplied, the mesh will be reconstructed for each fold.
`mesh`	Output from `make_mesh()`. If supplied, the same mesh will be used for all folds. This is faster and usually what you want.
`time`	The name of the time column. Leave as `NULL` if this is only spatial data.
`k_folds`	Number of folds.
`fold_ids`	Optional vector containing user fold IDs. Can also be a single string, e.g. `"fold_id"` representing the name of the variable in `data`. Ignored if `lfo` is TRUE
`lfo`	Logical. Use leave-future-out (LFO) cross validation? If `TRUE`, data from earlier time steps are used to predict future time steps. The `time` argument must be specified. See Details section below.
`lfo_forecast`	If `lfo = TRUE`, number of time steps ahead to forecast. For example, `lfo_forecast = 1` means fitting to time steps 1 to T and validating on T + 1. See Details section below.

`lfo_validations`
: If `lfo = TRUE`, number of times to step through the LFO process (i.e., number of validation folds). Defaults to 5. See Details section below.

`parallel`
: If `TRUE` and a `future::plan()` is supplied, will be run in parallel.

`use_initial_fit`
: Fit the first fold and use those parameter values as starting values for subsequent folds? Can be faster with many folds.

`save_models`
: Logical. If `TRUE` (default), the fitted model object for each fold is stored in the output. If `FALSE`, models are not saved, which can substantially reduce memory usage for large datasets or many folds. When `FALSE`, functions that require access to the fitted models (e.g., `tidy()`, `cv_to_waywiser()`) will not work.

`future_globals`
: A character vector of global variables used within arguments if an error is returned that **future.apply** can't find an object. This vector is appended to `TRUE` and passed to the argument `future.globals` in `future.apply::future_lapply()`. Useful if global objects are used to specify arguments like priors, families, etc.

`...`
: All other arguments required to run the `sdmTMB()` model. The `weights` argument is supported and will be combined with the internal fold-assignment mechanism (held-out data are assigned weight 0).

Details

Parallel processing

Parallel processing can be used by setting a `future::plan()`.

For example:

```
library(future)
plan(multisession)
# now use sdmTMB_cv() ...
```

Leave-future-out cross validation (LFOCV)

An example of LFOCV with 9 time steps, `lfo_forecast = 1`, and `lfo_validations = 2`:

- Fit data to time steps 1 to 7, predict and validate step 8.
- Fit data to time steps 1 to 8, predict and validate step 9.

An example of LFOCV with 9 time steps, `lfo_forecast = 2`, and `lfo_validations = 3`:

- Fit data to time steps 1 to 5, predict and validate step 7.
- Fit data to time steps 1 to 6, predict and validate step 8.
- Fit data to time steps 1 to 7, predict and validate step 9.

Note these are time steps as they are presented in order in the data. For example, in the `pcod` data example below steps between data points are not always one year but an `lfo_forecast = 2` forecasts 2 time steps as presented not two years.

See example below.

Value

A list:

- `data`: Original data plus columns for fold ID (`cv_fold`), CV predicted value (`cv_predicted`), CV log likelihood (`cv_loglik`), and CV deviance residuals (`cv_deviance_resid`).
- `models`: A list of fitted models, one per fold. `NULL` if `save_models = FALSE`.
- `fold_loglik`: Sum of log likelihoods of held-out data per fold (log predictive density per fold). More positive values indicate better out-of-sample prediction.
- `sum_loglik`: Sum of `fold_loglik` across all folds (total log predictive density). Use this to compare models—more positive values are better.
- `pdHess`: Logical vector: was the Hessian positive definite for each fold?
- `converged`: Logical: did all folds converge (all `pdHess` `TRUE`)?
- `max_gradients`: Maximum absolute gradient for each fold.

References

Ward, E.J., and S.C. Anderson. 2025. Approximating spatial processes with too many knots degrades the quality of probabilistic predictions. bioRxiv 2025.11.14.688354. doi:10.1101/2025.11.14.688354.

Examples

```
mesh <- make_mesh(pcod, c("X", "Y"), cutoff = 25)

# Set parallel processing first if desired with the future package.
# See the Details section above.

m_cv <- sdmTMB_cv(
  density ~ 0 + depth_scaled + depth_scaled2,
  data = pcod, mesh = mesh, spatial = "off",
  family = tweedie(link = "log"), k_folds = 2
)

m_cv$fold_loglik
m_cv$sum_loglik

head(m_cv$data)
m_cv$models[[1]]
m_cv$max_gradients



# Create mesh each fold:
m_cv2 <- sdmTMB_cv(
  density ~ 0 + depth_scaled + depth_scaled2,
  data = pcod, mesh_args = list(xy_cols = c("X", "Y"), cutoff = 20),
  family = tweedie(link = "log"), k_folds = 2
)

# Use fold_ids:
```

```
m_cv3 <- sdmTMB_cv(
  density ~ 0 + depth_scaled + depth_scaled2,
  data = pcod, mesh = mesh,
  family = tweedie(link = "log"),
  fold_ids = rep(seq(1, 3), nrow(pcod))[seq(1, nrow(pcod))]
)

# LFOCV:
m_lfocv <- sdmTMB_cv(
  present ~ s(year, k = 4),
  data = pcod,
  lfo = TRUE,
  lfo_forecast = 2,
  lfo_validations = 3,
  family = binomial(),
  mesh = mesh,
  spatial = "off", # fast example
  spatiotemporal = "off", # fast example
  time = "year" # must be specified
)

# See how the LFOCV folds were assigned:
fold_table <- table(m_lfocv$data$cv_fold, m_lfocv$data$year)
fold_table
```

`sdmTMB_stacking` *Perform stacking with log scores on* `sdmTMB_cv()` *output*

Description

[Experimental]

This approach is described in Yao et al. (2018) doi:10.1214/17BA1091. The general method minimizes (or maximizes) some quantity across models. For simple models with normal error, this may be the root mean squared error (RMSE), but other approaches include the log score. We adopt the latter here, where log scores are used to generate the stacking of predictive distributions

Usage

```
sdmTMB_stacking(model_list, include_folds = NULL)
```

Arguments

`model_list`	A list of models fit with `sdmTMB_cv()` to generate estimates of predictive densities. You will want to set the seed to the same value before fitting each model or manually construct the fold IDs so that they are the same across models.
`include_folds`	An optional numeric vector specifying which folds to include in the calculations. For example, if 5 folds are used for k-fold cross validation, and the first 4 are needed to generate these weights, `include_folds = 1:4`.

Value

A vector of model weights.

References

Yao, Y., Vehtari, A., Simpson, D., and Gelman, A. 2018. Using Stacking to Average Bayesian Predictive Distributions (with Discussion). Bayesian Analysis 13(3): 917–1007. International Society for Bayesian Analysis. doi:10.1214/17BA1091

Examples

```
# Set parallel processing if desired. See 'Details' in ?sdmTMB_cv

# Depth as quadratic:
set.seed(1)
m_cv_1 <- sdmTMB_cv(
  density ~ 0 + depth_scaled + depth_scaled2,
  data = pcod_2011, mesh = pcod_mesh_2011,
  family = tweedie(link = "log"), k_folds = 2
)
# Depth as linear:
set.seed(1)
m_cv_2 <- sdmTMB_cv(
  density ~ 0 + depth_scaled,
  data = pcod_2011, mesh = pcod_mesh_2011,
  family = tweedie(link = "log"), k_folds = 2
)

# Only an intercept:
set.seed(1)
m_cv_3 <- sdmTMB_cv(
  density ~ 1,
  data = pcod_2011, mesh = pcod_mesh_2011,
  family = tweedie(link = "log"), k_folds = 2
)

models <- list(m_cv_1, m_cv_2, m_cv_3)
weights <- sdmTMB_stacking(models)
weights
```

---

set_delta_model *Set delta model for* ggeffects::ggpredict()

Description

Set a delta model component to predict from with ggeffects::ggpredict().

Usage

```
set_delta_model(x, model = c(NA, 1, 2))
```

Arguments

x	An sdmTMB() model fit with a delta family such as delta_gamma().
model	Which delta/hurdle linear predictor to predict/plot with. NA does the combined prediction, 1 does the binomial part, and 2 does the positive part.

Details

A complete version of the examples below would be:

```
fit <- sdmTMB(density ~ poly(depth_scaled, 2), data = pcod_2011,
  spatial = "off", family = delta_gamma())

# binomial part:
set_delta_model(fit, model = 1) |>
  ggeffects::ggpredict("depth_scaled [all]")

# gamma part:
set_delta_model(fit, model = 2) |>
  ggeffects::ggpredict("depth_scaled [all]")

# combined:
set_delta_model(fit, model = NA) |>
  ggeffects::ggpredict("depth_scaled [all]")
```

But cannot be run on CRAN until the next version of **ggeffects** is available on CRAN. For now, you can install the GitHub version of **ggeffects**. https://github.com/strengejacke/ggeffects.

Value

The fitted model with a new attribute named `delta_model_predict`. We suggest you use `set_delta_model()` in a pipe (as in the examples) so that this attribute does not persist. Otherwise, predict.sdmTMB() will choose this model component by default. You can also remove the attribute yourself after:

```
attr(fit, "delta_model_predict") <- NULL
```

Examples

```
fit <- sdmTMB(density ~ poly(depth_scaled, 2), data = pcod_2011,
  spatial = "off", family = delta_gamma())

# binomial part:
set_delta_model(fit, model = 1)

# gamma part:
set_delta_model(fit, model = 2)
```

```
# combined:
set_delta_model(fit, model = NA)
```

`sigma.sdmTMB` *Extract residual standard deviation or dispersion parameter*

Description

Extract residual standard deviation or dispersion parameter

Usage

```
## S3 method for class 'sdmTMB'
sigma(object, ...)
```

Arguments

`object`	The fitted sdmTMB model object
`...`	Currently ignored

`simulate.sdmTMB` *Simulate from a fitted sdmTMB model*

Description

`simulate.sdmTMB` is an S3 method for producing a matrix of simulations from a fitted model. This is similar to `lme4::simulate.merMod()` and `glmmTMB::simulate.glmmTMB()`. It can be used with the **DHARMa** package among other uses.

Usage

```
## S3 method for class 'sdmTMB'
simulate(
  object,
  nsim = 1L,
  seed = sample.int(1e+06, 1L),
  type = c("mle-eb", "mle-mvn"),
  model = c(NA, 1, 2),
  newdata = NULL,
  re_form = NULL,
  mle_mvn_samples = c("single", "multiple"),
  mcmc_samples = NULL,
  return_tmb_report = FALSE,
  observation_error = TRUE,
```

```
  size = NULL,
  silent = FALSE,
  ...
)
```

Arguments

object	sdmTMB model
nsim	Number of response lists to simulate. Defaults to 1.
seed	Random number seed
type	How parameters should be treated. "mle-eb": fixed effects are at their maximum likelihood (MLE) estimates and random effects are at their empirical Bayes (EB) estimates. "mle-mvn": fixed effects are at their MLEs but random effects are taken from a single approximate sample. This latter option is a suggested approach if these simulations will be used for goodness of fit testing (e.g., with the DHARMa package).
model	If a delta/hurdle model, which model to simulate from? NA = combined, 1 = first model, 2 = second mdoel.
newdata	Optional new data frame from which to simulate.
re_form	NULL to specify a simulation conditional on fitted random effects (this only simulates observation error). ~0 or NA to simulate new random affects (smoothers, which internally are random effects, will not be simulated as new).
mle_mvn_samples	Applies if type = "mle-mvn". If "single", take a single MVN draw from the random effects. If "multiple", take an MVN draw from the random effects for each of the nsim.
mcmc_samples	An optional matrix of MCMC samples. See extract_mcmc() in the sdmTMBextra package.
return_tmb_report	Return the **TMB** report from simulate()? This lets you parse out whatever elements you want from the simulation. Not usually needed.
observation_error	Logical. Simulate observation error?
size	A vector of size (trials) in the case of a binomial family with newdata specified. If left NULL with newdata, will be assumed to be a vector of 1s.
silent	Logical. Silent?
...	Extra arguments passed to predict.sdmTMB(). E.g., one may wish to pass an offset argument if newdata are supplied in a model with an offset.

Value

Returns a matrix; number of columns is nsim.

See Also

simulate_new()

Examples

```
# start with some data simulated from scratch:
set.seed(1)
predictor_dat <- data.frame(X = runif(300), Y = runif(300), a1 = rnorm(300))
mesh <- make_mesh(predictor_dat, xy_cols = c("X", "Y"), cutoff = 0.1)
dat <- simulate_new(
  formula = ~ 1 + a1,
  data = predictor_dat,
  mesh = mesh,
  family = poisson(),
  range = 0.5,
  sigma_O = 0.2,
  seed = 42,
  B = c(0.2, -0.4) # B0 = intercept, B1 = a1 slope
)
fit <- sdmTMB(observed ~ 1 + a1, data = dat, family = poisson(), mesh = mesh)

# simulate from the model:
s1 <- simulate(fit, nsim = 300)
dim(s1)

# test whether fitted models are consistent with the observed number of zeros:
sum(s1 == 0)/length(s1)
sum(dat$observed == 0) / length(dat$observed)

# simulate with random effects sampled from their approximate posterior
s2 <- simulate(fit, nsim = 1, params = "mle-mvn")
# these may be useful in conjunction with DHARMa simulation-based residuals

# simulate with new random fields:
s3 <- simulate(fit, nsim = 1, re_form = ~ 0)
```

`simulate_new` *Simulate from a spatial/spatiotemporal model*

Description

`simulate_new()` uses TMB to simulate *new* data given specified parameter values. `simulate.sdmTMB()`, on the other hand, takes an *existing* model fit and simulates new observations and optionally new random effects. **Note:** `sdmTMB_simulate()` is retained as a synonym for backwards compatibility. We recommend using `simulate_new()` going forward as it more clearly conveys the function's purpose. `sdmTMB_simulate()` may eventually be deprecated.

Usage

```
simulate_new(
  formula,
  data,
  mesh,
```

```
  family = gaussian(link = "identity"),
  time = NULL,
  B = NULL,
  range = NULL,
  rho = NULL,
  sigma_O = NULL,
  sigma_E = NULL,
  sigma_Z = NULL,
  phi = NULL,
  tweedie_p = NULL,
  df = NULL,
  threshold_coefs = NULL,
  fixed_re = list(omega_s = NULL, epsilon_st = NULL, zeta_s = NULL),
  previous_fit = NULL,
  seed = sample.int(1e+06, 1),
  time_varying = NULL,
  time_varying_type = c("rw", "rw0", "ar1"),
  sigma_V = NULL,
  rho_time = NULL,
  ...
)

sdmTMB_simulate(
  formula,
  data,
  mesh,
  family = gaussian(link = "identity"),
  time = NULL,
  B = NULL,
  range = NULL,
  rho = NULL,
  sigma_O = NULL,
  sigma_E = NULL,
  sigma_Z = NULL,
  phi = NULL,
  tweedie_p = NULL,
  df = NULL,
  threshold_coefs = NULL,
  fixed_re = list(omega_s = NULL, epsilon_st = NULL, zeta_s = NULL),
  previous_fit = NULL,
  seed = sample.int(1e+06, 1),
  time_varying = NULL,
  time_varying_type = c("rw", "rw0", "ar1"),
  sigma_V = NULL,
  rho_time = NULL,
  ...
)
```

Arguments

formula	A *one-sided* formula describing the fixed-effect structure. Random intercepts are not (yet) supported. Fixed effects should match the corresponding B argument vector of coefficient values.
data	A data frame containing the predictors described in formula and the time column if time is specified.
mesh	Output from make_mesh().
family	Family as in sdmTMB(). Delta families are not supported. Instead, simulate the two component models separately and combine.
time	The time column name.
B	A vector of beta values (fixed-effect coefficient values).
range	Parameter that controls the decay of spatial correlation. If a vector of length 2, share_range will be set to FALSE and the spatial and spatiotemporal ranges will be unique.
rho	Spatiotemporal correlation between years; should be between -1 and 1.
sigma_O	SD of spatial process (Omega).
sigma_E	SD of spatiotemporal process (Epsilon).
sigma_Z	SD of spatially varying coefficient field (Zeta).
phi	Observation error scale parameter (e.g., SD in Gaussian).
tweedie_p	Tweedie p (power) parameter; between 1 and 2.
df	Student-t degrees of freedom.
threshold_coefs	An optional vector of threshold coefficient values if the formula includes breakpt() or logistic(). If breakpt(), these are slope and cut values. If logistic(), these are the threshold at which the function is 50% of the maximum, the threshold at which the function is 95% of the maximum, and the maximum. See the model description vignette for details.
fixed_re	A list of optional random effects to fix at specified (e.g., previously estimated) values. Values of NULL will result in the random effects being simulated.
previous_fit	(**Deprecated**; please use simulate.sdmTMB()). An optional previous sdmTMB() fit to pull parameter values. Will be over-ruled by any non-NULL specified parameter arguments.
seed	Seed number.
time_varying	An optional one-sided formula describing time-varying covariates passed through to sdmTMB() for building a time-varying random effect design matrix.
time_varying_type	Type of temporal process applied to time_varying. Must be one of 'rw', 'rw0', or 'ar1'.
sigma_V	SD(s) of the time-varying process. Provide a single value or a vector matching the number of time-varying coefficients.
rho_time	Autoregressive correlation(s) for time-varying parameters when time_varying_type = "ar1". Values must lie between -1 and 1 and may be supplied as a single value or a vector the same length as sigma_V.
...	Any other arguments to pass to sdmTMB().

Value

A data frame where:

- The 1st column is the time variable (if present).
- The 2nd and 3rd columns are the spatial coordinates.
- `omega_s` represents the simulated spatial random effects (only if present).
- `zeta_s` represents the simulated spatial varying covariate field (only if present).
- `epsilon_st` represents the simulated spatiotemporal random effects (only if present).
- `eta` is the true value in link space
- `mu` is the true value in inverse link space.
- `observed` represents the simulated process with observation error.
- The remaining columns are the fixed-effect model matrix.

See Also

`simulate.sdmTMB()`

Examples

```
set.seed(123)

# make fake predictor(s) (a1) and sampling locations:
predictor_dat <- data.frame(
  X = runif(300), Y = runif(300),
  a1 = rnorm(300), year = rep(1:6, each = 50)
)
mesh <- make_mesh(predictor_dat, xy_cols = c("X", "Y"), cutoff = 0.1)

sim_dat <- simulate_new(
  formula = ~ 1 + a1,
  data = predictor_dat,
  time = "year",
  mesh = mesh,
  family = gaussian(),
  range = 0.5,
  sigma_E = 0.1,
  phi = 0.1,
  sigma_O = 0.2,
  seed = 42,
  B = c(0.2, -0.4) # B0 = intercept, B1 = a1 slope
)
head(sim_dat)

if (require("ggplot2", quietly = TRUE)) {
  ggplot(sim_dat, aes(X, Y, colour = observed)) +
    geom_point() +
    facet_wrap(~year) +
    scale_color_gradient2()
}
```

```
# fit to the simulated data:
fit <- sdmTMB(observed ~ a1, data = sim_dat, mesh = mesh, time = "year")
fit
```

---

spread_sims *Extract parameter simulations from the joint precision matrix*

Description

`spread_sims()` returns a wide-format data frame. `gather_sims()` returns a long-format data frame. The format matches the format in the **tidybayes** `spread_draws()` and `gather_draws()` functions.

Usage

```
spread_sims(object, nsim = 200)

gather_sims(object, nsim = 200)
```

Arguments

`object`	Output from `sdmTMB()`.
`nsim`	The number of simulation draws.

Value

A data frame. `gather_sims()` returns a long-format data frame:

- `.iteration`: the sample ID
- `.variable`: the parameter name
- `.value`: the parameter sample value

`spread_sims()` returns a wide-format data frame:

- `.iteration`: the sample ID
- columns for each parameter with a sample per row

Examples

```
m <- sdmTMB(density ~ depth_scaled,
  data = pcod_2011, mesh = pcod_mesh_2011, family = tweedie())
head(spread_sims(m, nsim = 10))
head(gather_sims(m, nsim = 10))
samps <- gather_sims(m, nsim = 1000)

if (require("ggplot2", quietly = TRUE)) {
  ggplot(samps, aes(.value)) + geom_histogram() +
    facet_wrap(~.variable, scales = "free_x")
}
```

`tidy.sdmTMB` *Turn sdmTMB model output into a tidy data frame*

Description

Turn sdmTMB model output into a tidy data frame

Usage

```
## S3 method for class 'sdmTMB'
tidy(
  x,
  effects = c("fixed", "ran_pars", "ran_vals", "ran_vcov"),
  model = 1,
  conf.int = TRUE,
  conf.level = 0.95,
  exponentiate = FALSE,
  silent = FALSE,
  ...
)

## S3 method for class 'sdmTMB_cv'
tidy(x, ...)
```

Arguments

`x`	Output from `sdmTMB()`.
`effects`	A character value. One of `"fixed"` ('fixed' or main-effect parameters), `"ran_pars"` (standard deviations, spatial range, and other random effect and dispersion-related terms), `"ran_vals"` (individual random intercepts or slopes, if included; behaves like `ranef()`), or `"ran_vcov"` (list of variance covariance matrices for the random effects, by model and group).
`model`	Which model to tidy if a delta model (1 or 2). The `model` will be ignored when effects is `"ran_vals"` (all returned in a single dataframe)
`conf.int`	Include a confidence interval?
`conf.level`	Confidence level for CI.
`exponentiate`	Whether to exponentiate the fixed-effect coefficient estimates and confidence intervals.
`silent`	Omit any messages?
`...`	Extra arguments (not used).

Details

Follows the conventions of the **broom** and **broom.mixed** packages.

Currently, `effects = "ran_pars"` also includes dispersion-related terms (e.g., `phi`), which are not actually associated with random effects.

Standard errors for spatial variance terms fit in log space (e.g., variance terms, range, or parameters associated with the observation error) are omitted to avoid confusion. Confidence intervals are still available.

Value

A data frame

Examples

```
fit <- sdmTMB(density ~ poly(depth_scaled, 2, raw = TRUE),
  data = pcod_2011, mesh = pcod_mesh_2011,
  family = tweedie()
)
tidy(fit)
tidy(fit, conf.int = TRUE)
tidy(fit, "ran_pars", conf.int = TRUE)

pcod_2011$fyear <- as.factor(pcod_2011$year)
fit <- sdmTMB(density ~ poly(depth_scaled, 2, raw = TRUE) + (1 | fyear),
  data = pcod_2011, mesh = pcod_mesh_2011,
  family = tweedie()
)
tidy(fit, "ran_vals")
```

---

update.sdmTMB *Update an sdmTMB model*

Description

This method updates an sdmTMB model with new arguments, automatically handling the mesh object to avoid environment issues when loading models from saved files.

Usage

```
## S3 method for class 'sdmTMB'
update(object, formula., ..., evaluate = TRUE)
```

Arguments

`object`	An sdmTMB model object
`formula.`	Optional updated formula
`...`	Other arguments to update in the model call
`evaluate`	If `TRUE` (default), the updated call is evaluated; if `FALSE`, the call is returned unevaluated

Value

An updated sdmTMB model object (if `evaluate = TRUE`) or an unevaluated call (if `evaluate = FALSE`)

Examples

```
mesh <- make_mesh(pcod_2011, c("X", "Y"), cutoff = 20)
fit <- sdmTMB(density ~ 1, data = pcod_2011, mesh = mesh,
  family = tweedie(link = "log"))
fit2 <- update(fit, family = delta_gamma())
```

---

visreg_delta *Plot sdmTMB models with the* **visreg** *package*

Description

sdmTMB models fit with regular (non-delta) families can be passed to `visreg::visreg()` or `visreg::visreg2d()` directly. Examples are shown below. Delta models can use the helper functions `visreg_delta()` or `visreg2d_delta()` described here.

Usage

```
visreg_delta(object, ..., model = c(1, 2))

visreg2d_delta(object, ..., model = c(1, 2))
```

Arguments

`object`	Fit from `sdmTMB()`
`...`	Any arguments passed to `visreg::visreg()` or `visreg::visreg2d()`
`model`	1st or 2nd delta model

Details

Note the residuals are currently randomized quantile residuals, *not* deviance residuals as is usual for GLMs with **visreg**.

Value

A plot from the visreg package. Optionally, the data plotted invisibly if `plot = FALSE`. This is useful if you want to make your own plot after.

Examples

```
if (require("ggplot2", quietly = TRUE) &&
  require("visreg", quietly = TRUE)) {


  fit <- sdmTMB(
    density ~ s(depth_scaled),
    data = pcod_2011,
    spatial = "off",
    family = tweedie()
  )
  visreg::visreg(fit, xvar = "depth_scaled")

  visreg::visreg(fit, xvar = "depth_scaled", scale = "response")
  v <- visreg::visreg(fit, xvar = "depth_scaled")
  head(v$fit)
  # now use ggplot2 etc. if desired

  # Delta model example:
  fit_dg <- sdmTMB(
    density ~ s(depth_scaled, year, k = 8),
    data = pcod_2011, mesh = pcod_mesh_2011,
    spatial = "off",
    family = delta_gamma()
  )
  visreg_delta(fit_dg, xvar = "depth_scaled", model = 1, gg = TRUE)
  visreg_delta(fit_dg, xvar = "depth_scaled", model = 2, gg = TRUE)
  visreg_delta(fit_dg,
    xvar = "depth_scaled", model = 1,
    scale = "response", gg = TRUE
  )
  visreg_delta(fit_dg,
    xvar = "depth_scaled", model = 2,
    scale = "response"
  )
  visreg_delta(fit_dg,
    xvar = "depth_scaled", model = 2,
    scale = "response", gg = TRUE, rug = FALSE
  )
  visreg2d_delta(fit_dg,
    xvar = "depth_scaled", yvar = "year",
    model = 2, scale = "response"
  )
  visreg2d_delta(fit_dg,
    xvar = "depth_scaled", yvar = "year",
    model = 1, scale = "response", plot.type = "persp"
  )
  visreg2d_delta(fit_dg,
    xvar = "depth_scaled", yvar = "year",
    model = 2, scale = "response", plot.type = "gg"
  )
```

```
}
```

Index