Package 'sdmTMB'

October 17, 2026

Type Package

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

Version 1.2.0

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, igraph, lifecycle, Matrix, methods, mgcv, mvtnorm, nlme, reformulas (>= 0.4.4), rlang, RTMB (>= 1.9), 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, loo, MASS, numDeriv, rmarkdown, sf, spatstat.data, splancs, testthat, tibble, visreg, waywiser, withr

LinkingTo RcppEigen, TMB

VignetteBuilder knitr

ByteCompile true

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

LazyData true

SystemRequirements GNU make

Config/roxygen2/version 8.1.1

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 2026-10-16 23:30:02 UTC

Contents

add_utm_columns 4

cAIC 5
coef.sdmTMB 6
compare_deviance 7
cv_to_waywiser 9
dharma_residuals 10
Effect.sdmTMB 13
elpd.sdmTMB_cv 13
emmeans.sdmTMB 14
Families 15
get_censored_upper 20
get_index 22
get_index_sims 27
get_pars 29
get_prior_parameters 29
get_range_edge 31
make_areal_domain 33
make_category_svc 34
make_mesh 36
make_zero_one_map 38
nonlocal_terms 40
ohio_df 41
pcod 42
plot_anisotropy 43
plot_nonlocal_covariate 44
plot_pc_matern 47
plot_smooth 48
predict.sdmTMB 49
project 55
replicate_df 59
residuals.sdmTMB 59
run_extra_optimization 62
sanity 63
sdmTMB 64
sdmTMBcontrol 76
sdmTMBpriors 79
sdmTMB_cv 84
sdmTMB_stacking 89
set_delta_model 90
sigma.sdmTMB 91
simulate.sdmTMB 92
simulate_new 94
spread_sims 98
tidy.sdmTMB 99
update.sdmTMB 101
visreg_delta 102

Index **105**

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 a projected coordinate reference system with distance-based units. You can do this yourself 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 it 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 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 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

`compare_deviance` *Compare deviance between two models*

Description

Calculates the proportion of deviance explained by a model relative to a simpler model, usually an intercept-only model. This is a pseudo-R^2. Any parameters that the deviance depends on, other than the mean, are held fixed at their values from the first model.

Usage

```
compare_deviance(object, reduced)
```

Arguments

`object`	Output from `sdmTMB()`. The model of interest. Its shape parameters (see Details) are used for both models.
`reduced`	Output from `sdmTMB()` fit to the same response with the same family. For a pseudo-R^2, an intercept-only model with no random effects or random fields (see Details).

Details

The proportion of deviance explained, $1 - D/D_0$, is a common pseudo-R^2 for generalized linear models (Cameron and Windmeijer 1997). It is the proportion of the Kullback-Leibler divergence between the data and the null model that the fitted model removes. It reduces to the usual R^2 for a Gaussian linear model and is the "deviance explained" reported by **mgcv**.

Shape parameters. The deviance is twice the log-likelihood difference between a saturated model and the fitted model. For some families the saturated model, and hence the deviance, depends on parameters other than the mean:

- negative binomial families: the dispersion parameter `phi`,
- `tweedie()`: the power parameter `p` (but not `phi`),
- `lognormal()`: `phi` (through the bias correction to the mean),
- `gengamma()`: `phi` and `Q`.

If these parameters are estimated separately for each model, the two deviances are measured against different saturated models and their ratio is not meaningful. A simpler model typically estimates more overdispersion (e.g., a smaller NB2 `phi`), which lowers its deviance and understates the deviance explained by the more complex model. The naive ratio can even be negative.

This function therefore refits `reduced` with these parameters fixed at the values estimated in `object`. This matches how **mgcv** computes the null deviance for its negative binomial and Tweedie families. Families whose deviance depends only on the mean (e.g., `poisson()`, `binomial()`, `Gamma()`, `gaussian()`) need no refit. Because `object` supplies these parameters, swapping the arguments gives a different answer.

Random effects and random fields. For models with random effects, including spatial and spatiotemporal random fields, the deviance is conditional on the predicted random effects. The deviance explained is therefore a conditional pseudo-R^2: the in-sample variation captured by the fixed effects and the predicted random effects together, similar in spirit to the conditional R^2 for mixed models. It is a reasonable summary of in-sample fit, but note that:

- Like R^2, it does not penalize complexity. More flexible random fields (e.g., adding spatiotemporal fields or a finer mesh) almost always increase it, and a sufficiently flexible field can approach 1. Do not use it to choose among random effect structures, random field structures, or meshes. Use `cAIC()` (the conditional likelihood with a penalty for effective degrees of freedom), `AIC()`, or `sdmTMB_cv()` instead.
- It describes fit to the observed data, not predictive performance for new locations or times.
- `reduced` should not include random effects or random fields. Shape parameters such as `phi` also determine how variation is partitioned between random effects and observation error, so fixing them in a `reduced` model that has random effects changes its random effect estimates and makes the result hard to interpret.
- The difference in deviance does not follow a chi-squared distribution and is not a likelihood-ratio test.

For delta models, the deviance is summed over both components.

Value

A data frame with the deviance of each model, their difference (`reduced` minus `object`), and the proportion of the deviance of `reduced` explained by `object`. The refitted `reduced` model is included as attribute `"reduced"`, and the fixed parameter values (on their internal scale) as attribute `"fixed"`.

References

Cameron, A. C., and Windmeijer, F. A. G. (1997). An R-squared measure of goodness of fit for some common nonlinear regression models. Journal of Econometrics, 77(2), 329–342. doi:10.1016/S03044076(96)018180

See Also

`stats::deviance()`, `cAIC()`

Examples

```
fit <- sdmTMB(density ~ depth_scaled + depth_scaled2,
  data = pcod_2011, spatial = "off", family = tweedie())
fit0 <- sdmTMB(density ~ 1,
  data = pcod_2011, spatial = "off", family = tweedie())
compare_deviance(fit, fit0)

# Not comparable, since each model estimates its own Tweedie p:
1 - deviance(fit) / deviance(fit0)

# Conditional pseudo-R^2 including a spatial random field:
mesh <- make_mesh(pcod_2011, c("X", "Y"), cutoff = 20)
```

```
fit_sp <- sdmTMB(density ~ depth_scaled + depth_scaled2,
  data = pcod_2011, mesh = mesh, family = tweedie())
compare_deviance(fit_sp, fit0)
```

`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 residual diagnostics*

Description

Plot (and optionally return) **DHARMa** residual diagnostics. This is a wrapper around DHARMa::createDHARMa() for use with sdmTMB() models. **Note:** We recommend setting type = "mle-mvn" in simulate.sdmTMB() so the resulting residuals 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()`. We recommend setting `type = "mle-mvn"` in the call to `simulate.sdmTMB()` so the residuals 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 of these residuals relative to the `residuals.sdmTMB()` method are (1) they work with delta/hurdle models for the combined predictions, not just the two parts separately, (2) they should work for all families, not 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(). It can then be plotted with the 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))
```

elpd.sdmTMB_cv *Expected log predictive density from cross validation for* **loo**

Description

Converts the pointwise out-of-sample log predictive densities (cv_loglik) from sdmTMB_cv() into a **loo** elpd_generic object. This lets loo::loo_compare() compare models, including a standard error of the difference in expected log predictive density.

Usage

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

Arguments

x	Output from sdmTMB_cv().
...	Not used.

Details

Models compared with loo::loo_compare() must be fit to the same data with the same folds (e.g., via fold_ids) and should use the same predictive argument. **loo** only checks that the number of observations matches. The standard errors reflect variation across observations, not the Monte Carlo error from nsim.

The printed object refers to a "1 by N log-likelihood matrix" because the pointwise values are already integrated over draws within sdmTMB_cv().

Value

An object of class elpd_generic and loo. See loo::elpd().

See Also

sdmTMB_cv(), compare_deviance() for in-sample deviance explained.

Examples

```
if (requireNamespace("loo", quietly = TRUE)) {
  mesh <- make_mesh(pcod_2011, c("X", "Y"), cutoff = 20)
  folds <- rep(1:4, length.out = nrow(pcod_2011))
  m1 <- sdmTMB_cv(density ~ 1, data = pcod_2011, mesh = mesh,
    family = tweedie(), fold_ids = folds)
  m2 <- sdmTMB_cv(density ~ depth_scaled + depth_scaled2,
    data = pcod_2011, mesh = mesh, family = tweedie(), fold_ids = folds)
  loo::loo_compare(list(null = loo::elpd(m1), depth = loo::elpd(m2)))
}
```

---

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 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)
```

`Families` *Additional families*

Description

Additional families compatible with sdmTMB(). In addition to gaussian(), Gamma(), binomial(), and poisson(), sdmTMB provides:

- Continuous: student(), lognormal(), gengamma()
- Proportions: Beta(), ordbeta()
- Non-negative with exact zeros: tweedie()
- Counts: nbinom2(), nbinom1(), truncated_nbinom2(), truncated_nbinom1(), betabinomial()
- Censored counts: censored_poisson(), censored_nbinom2(), censored_nbinom1(), censored_binomial(), censored_betabinomial()
- Two-component mixtures: gamma_mix(), lognormal_mix(), nbinom2_mix()
- Delta/hurdle: delta_gamma(), delta_lognormal(), delta_gengamma(), delta_beta(), delta_truncated_nbinom2(), delta_truncated_nbinom1(), delta_gamma_mix(), delta_lognormal_mix()

See 'Binomial families', 'Censored families', and 'Delta/hurdle models' in the Details of sdmTMB().

Usage

```
student(link = "identity", df = NULL)

lognormal(link = "log")

gengamma(link = "log")

Beta(link = "logit")

ordbeta(link = "logit")

tweedie(link = "log")

nbinom2(link = "log")

nbinom1(link = "log")

truncated_nbinom2(link = "log")

truncated_nbinom1(link = "log")

betabinomial(link = "logit")

censored_poisson(link = "log")

censored_nbinom2(link = "log")

censored_nbinom1(link = "log")

censored_binomial(link = "logit")
```

```
censored_betabinomial(link = "logit")

gamma_mix(link = "log", p_extreme = NULL)

lognormal_mix(link = "log", p_extreme = NULL)

nbinom2_mix(link = "log", p_extreme = NULL)

delta_gamma(link1, link2 = "log", type = c("standard", "poisson-link"))

delta_lognormal(link1, link2 = "log", type = c("standard", "poisson-link"))

delta_gengamma(link1, link2 = "log", type = c("standard", "poisson-link"))

delta_beta(link1 = "logit", link2 = "logit")

delta_truncated_nbinom2(link1 = "logit", link2 = "log")

delta_truncated_nbinom1(link1 = "logit", link2 = "log")

delta_gamma_mix(link1 = "logit", link2 = "log", p_extreme = NULL)

delta_lognormal_mix(
  link1,
  link2 = "log",
  type = c("standard", "poisson-link"),
  p_extreme = NULL
)
```

Arguments

link	Link.
df	Student-t degrees of freedom parameter. Can be NULL to estimate (default) or a numeric value > 1 to fix at a specific value.
p_extreme	Optional fixed probability for the extreme component. If NULL (default), this is estimated. If specified, must be a proportion between 0 and 1.
link1	Link for first part of delta/hurdle model. Defaults to "logit" for type = "standard" and "log" for type = "poisson-link".
link2	Link for second part of delta/hurdle model.
type	Delta/hurdle family type. "standard" for a classic hurdle model. "poisson-link" for a Poisson-link delta model (Thorson 2018).

Details

For student(), the degrees of freedom parameter is estimated by default (df = NULL). You can fix it at a specific value by providing a number > 1 (e.g., df = 3).

The gengamma() family was implemented by J.T. Thorson and uses the Prentice (1974) parameterization such that the lognormal occurs as the internal parameter gengamma_Q (reported in print() or summary() as "Generalized gamma Q") approaches 0. If Q matches phi the distribution should be the gamma.

The ordbeta() family implements the ordered-beta regression of Kubinec (2023) for continuous responses on the closed unit interval [0, 1] with point masses at the endpoints. It is a parsimonious alternative to zero-one-inflated beta models because the three components (zeros, ones, and continuous (0, 1) values) share a single linear predictor. Two internal cutpoints are estimated and reported on the response scale (plogis()).

The nbinom2 negative binomial parameterization is the NB2 where the variance grows quadratically with the mean (Hilbe 2011).

The nbinom1 negative binomial parameterization lets the variance grow linearly with the mean (Hilbe 2011).

The censored_*() families treat some counts as known only to lie within a range, e.g., to account for hook competition in longline surveys (Watson et al. 2023). Supply the bounds with sdmTMB(censored_upper = ...).

- censored_poisson(), censored_nbinom1(), and censored_nbinom2() have no upper limit on the count; use an offset of log hooks for effort.
- censored_binomial() and censored_betabinomial() take the number of trials (e.g., hooks) via weights, which caps the count.

All but censored_poisson() need the RTMB backend (the default). See censored_method in sdmTMBcontrol() for how censored_betabinomial() computes its likelihood, and the hook competition article.

The families ending in _mix() are 2-component mixtures where each distribution has its own mean but they share a scale parameter. (Thorson et al. 2011). See the model-description vignette for details. The parameter p_extreme = plogis(logit_p_extreme) is the probability of the extreme (larger) mean and exp(log_ratio_mix) + 1 is the ratio of the larger extreme mean to the "regular" mean. You can see these parameters in model$sd_report. The parameter p_extreme can be fixed a priori and passed in as a proportion for these families.

The default first-component link (link1) for delta models of type = "standard" is "logit". For type = "poisson-link", the default link1 is "log".

Value

A list with elements common to standard R family objects including family, link, linkfun, and linkinv. Delta/hurdle model families also have elements delta (logical) and type (standard vs. Poisson-link).

References

Generalized gamma family:

Prentice, R.L. 1974. A log gamma model and its maximum likelihood estimation. Biometrika 61(3): 539–544. doi:10.1093/biomet/61.3.539

Stacy, E.W. 1962. A Generalization of the Gamma Distribution. The Annals of Mathematical Statistics 33(3): 1187–1192. Institute of Mathematical Statistics.

Dunic, J.C., Conner, J., Anderson, S.C., and Thorson, J.T. 2025. The generalized gamma is a flexible distribution that outperforms alternatives when modelling catch rate data. ICES Journal of Marine Science 82(4): fsaf040. doi:10.1093/icesjms/fsaf040.

Ordered beta regression:

Kubinec, R. 2023. Ordered beta regression: a parsimonious, well-fitting model for continuous data with lower and upper bounds. Political Analysis 31(4): 519–536. doi:10.1017/pan.2022.20

Negative binomial families:

Hilbe, J. M. 2011. Negative binomial regression. Cambridge University Press.

Censored families:

Watson, J., Edwards, A.M., and Auger-Méthé, M. 2023. A statistical censoring approach accounts for hook competition in abundance indices from longline surveys. Canadian Journal of Fisheries and Aquatic Sciences. 80(3): 468–486. doi:10.1139/cjfas20220159

Families ending in `_mix()`:

Thorson, J.T., Stewart, I.J., and Punt, A.E. 2011. Accounting for fish shoals in single- and multi-species survey data using mixture distribution models. Can. J. Fish. Aquat. Sci. 68(9): 1681–1693. doi:10.1139/f2011086.

Poisson-link delta families:

Thorson, J.T. 2018. Three problems with the conventional delta-model for biomass sampling data, and a computationally efficient alternative. Canadian Journal of Fisheries and Aquatic Sciences, 75(9), 1369-1382. doi:10.1139/cjfas20170266

Examples

```
student(link = "identity") # estimate df
student(link = "identity", df = 3) # fix df at 3
lognormal(link = "log")
gengamma(link = "log")
Beta(link = "logit")
ordbeta(link = "logit")
tweedie(link = "log")
nbinom2(link = "log")
nbinom1(link = "log")
truncated_nbinom2(link = "log")
truncated_nbinom1(link = "log")
betabinomial(link = "logit")
censored_poisson(link = "log")
censored_nbinom2(link = "log")
censored_nbinom1(link = "log")
censored_binomial(link = "cloglog")
censored_betabinomial(link = "cloglog")
gamma_mix(link = "log")
lognormal_mix(link = "log")
nbinom2_mix(link = "log")
delta_gamma()
delta_lognormal()
delta_gengamma()
delta_beta()
delta_truncated_nbinom2()
```

```
delta_truncated_nbinom1()
delta_gamma_mix()
delta_lognormal_mix()
```

`get_censored_upper` *Calculate an upper bound on catch counts for the censored Poisson family*

Description

Calculate an upper bound on catch counts for the censored Poisson family

Usage

```
get_censored_upper(prop_removed, n_catch, n_hooks, pstar = 0.95)
```

Arguments

`prop_removed`	The proportion of baits removed in each fishing event from *any* species. I.e., the proportion of hooks returning without bait for any reason.
`n_catch`	The observed catch counts on each fishing event of the target species.
`n_hooks`	The number of hooks deployed on each fishing event.
`pstar`	A single value between `0 <= pstar <= 1` specifying the breakdown point of observed catch counts as a result of hook competition.

Details

`pstar` could be obtained via inspecting a GAM or other smoother fit with catch counts as the response, an offset for log(hook count), and proportion of baits removed for each fishing event as the predictor. Check when the curve drops off as the proportion bait removed increases.

Pass the returned vector to the `censored_upper` argument of `sdmTMB()` with `family = censored_poisson()` and the observed catch counts (`n_catch`) as the response. Fishing events with `prop_removed` below `pstar` get an upper bound equal to the observed catch and are therefore treated as uncensored. Alternatively, set `censored_upper` to `Inf` for fishing events above `pstar` to use the right-censored likelihood without an upper bound.

The right-censored Poisson density can be written as:

```
dcens_pois <- function(x, lambda) {
    1 - ppois(x - 1, lambda)
 }
```

and the right-censored Poisson density with an upper limit can be written as:

```
dcens_pois_upper <- function(x, lambda, upper) {
  ppois(upper, lambda) - ppois(x - 1, lambda)
}
```

In practice, these computations are done in log space for numerical stability.

With `censored_binomial()` or `censored_betabinomial()`, the number of hooks is already an upper bound on the count, so `Inf` can be used for censored fishing events.

Value

A numeric vector of upper bound catch counts of the target species to improve convergence of the censored method. Bounds are capped at `n_hooks`.

References

Watson, J., Edwards, A.M., and Auger-Méthé, M. 2023. A statistical censoring approach accounts for hook competition in abundance indices from longline surveys. Canadian Journal of Fisheries and Aquatic Sciences. 80(3): 468–486. doi:10.1139/cjfas20220159

See Also

`censored_poisson()`, `censored_binomial()`, `censored_betabinomial()`, and the hook competition article.

Examples

```
dat <- structure(
  list(
    n_catch = c(
      78L, 63L, 15L, 6L, 7L, 11L, 37L, 99L, 34L, 100L, 77L, 79L,
      98L, 30L, 49L, 33L, 6L, 28L, 99L, 33L
    ),
    prop_removed = c(
      0.61, 0.81, 0.96, 0.69, 0.99, 0.98, 0.25, 0.95, 0.89, 1, 0.95, 0.95,
      0.94, 1, 0.95, 1, 0.84, 0.3, 1, 0.99
    ), n_hooks = c(
      140L, 140L, 140L, 140L, 140L, 140L, 140L, 140L, 140L, 140L, 140L, 140L,
      140L, 140L, 140L, 140L, 140L, 140L, 140L, 140L
    )
  ),
  class = "data.frame", row.names = c(NA, -20L)
)
upr <- get_censored_upper(dat$prop_removed, dat$n_catch, dat$n_hooks, pstar = 0.9)
upr

plot(dat$n_catch, upr, xlab = "N catch", ylab = "N catch upper limit")
abline(0, 1, lty = 2)
above_pstar <- dat[dat$prop_removed > 0.9, ]
upr_pstar <- upr[dat$prop_removed > 0.9]
points(above_pstar$n_catch, upr_pstar, col = "red", pch = 20)
txt <- paste0(
  "Red indicates catch events\n",
  "with >= pstar proportion of hooks\n",
  "coming back without bait.\n\n",
  "The rest are fishing events < pstar\n",
  "('high-quality' events)."
```

```
)
text(10, 120, txt, adj = 0)
```

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

Description

Calculate quantities derived from a model fitted with `sdmTMB()` by combining its predictions over a grid (`newdata`) for each time step, with standard errors and confidence intervals that account for uncertainty in both fixed and random effects:

Usage

```
get_index(
  obj,
  newdata = NULL,
  bias_correct = TRUE,
  level = 0.95,
  area = 1,
  offset = NULL,
  silent = TRUE,
  derived_link = NULL,
  predict_args = list(),
  ...
)

get_index_split(
  obj,
  newdata,
  bias_correct = TRUE,
  nsplit = 1,
  level = 0.95,
  area = 1,
  offset = NULL,
  silent = FALSE,
  predict_args = list(),
  derived_link = NULL,
  ...
)

get_cog(
  obj,
  newdata = NULL,
  bias_correct = FALSE,
  level = 0.95,
```

```
  format = c("long", "wide"),
  area = 1,
  offset = NULL,
  silent = TRUE,
  derived_link = NULL,
  predict_args = list(),
  ...
)

get_weighted_average(
  obj,
  newdata = NULL,
  vector,
  bias_correct = FALSE,
  level = 0.95,
  area = 1,
  offset = NULL,
  silent = TRUE,
  derived_link = NULL,
  predict_args = list(),
  ...
)

get_eao(
  obj,
  newdata = NULL,
  bias_correct = FALSE,
  level = 0.95,
  area = 1,
  offset = NULL,
  silent = TRUE,
  derived_link = NULL,
  predict_args = list(),
  ...
)
```

Arguments

obj	A model fitted with sdmTMB(). For backwards compatibility, output from predict.sdmTMB() with return_tmb_object = TRUE is also accepted (not for get_index_split()).
newdata	A data frame of locations (and times) to combine predictions over, usually a grid covering the area of interest replicated for each time step (see replicate_df()). Same requirements as newdata in predict.sdmTMB(). Results are returned only for time steps in newdata. Required unless obj is legacy prediction output or was fitted with the deprecated do_index = TRUE.
bias_correct	Logical: apply bias correction? Predictions plugged in at the estimated random effects can be biased once transformed to the response scale and summed; bias correction (Thorson and Kristensen 2016) adjusts the estimate for this. Rec-

ommended for final results but slower, so it can be turned off while exploring models. The standard errors are not bias-corrected.

`level` The confidence level of the intervals.

`area` The area of each row of `newdata` (e.g., grid cell area), used to weight predictions when combining them: (1) a numeric vector with one value per row of `newdata`, (2) a single value for all rows, or (3) the name of a column in `newdata`. With the default of 1, the index is a sum of densities rather than an area-weighted total. See Details for other uses.

`offset` An optional numeric vector of offset values, one per row of `newdata` (not a column name). If `NULL` (default), the offset is 0, so predictions are per unit of the offset (e.g., density rather than catch when the offset is log area swept).

`silent` Logical: suppress TMB's output while calculating? In `get_index_split()`, controls the progress bar instead.

`derived_link` Optional link whose inverse is applied to the linear predictor, in place of the model's own link, before combining predictions. Currently only available for non-delta `binomial()`, `censored_binomial()`, `betabinomial()`, and `censored_betabinomial()` models fit with `link = "cloglog"`; there, `derived_link = "log"` sums expected event rates (e.g., catch per hook) instead of probabilities. See the hook competition vignette.

`predict_args` A named list of other arguments to pass to `predict.sdmTMB()` (supply `newdata` and `offset` directly). For example, `list(re_form = NA)` calculates the quantity with the spatial and spatiotemporal random fields set to zero, and `list(re_form_iid = NA)` does the same for IID random effects.

`...` Passed to `TMB::sdreport()` or `RTMB::sdreport()`.

`nsplit` The number of chunks of time steps to calculate the index in separately. For large grids or models, `nsplit > 1` reduces memory use but is usually slower.

`format` The shape of the output: `"long"` or `"wide"` (see Value).

`vector` A numeric vector of values to average (e.g., depth or temperature), one per row of `newdata`.

Details

- `get_index()`: total abundance or biomass (an index).
- `get_index_split()`: the same, calculated a few time steps at a time to reduce memory use.
- `get_cog()`: center of gravity (the mean x and y coordinates, weighted by abundance).
- `get_eao()`: effective area occupied.
- `get_weighted_average()`: the mean of any variable weighted by abundance (e.g., the average depth or temperature occupied).

For each time step, predictions are converted to the response scale (for delta models, combining both components), multiplied by `area`, and combined. With μ_i the prediction and a_i the area for row i:

- Index: $\sum_i a_i \mu_i$.

- Weighted average: $\sum_i a_i \mu_i v_i / \sum_i a_i \mu_i$, for values v_i given by `vector`. Center of gravity is the weighted average of the x and y coordinates, in the units of the coordinates used to build the mesh.
- Effective area occupied: $(\sum_i a_i \mu_i)^2 / \sum_i a_i \mu_i^2$, the area needed to hold the total if density were spread evenly at the abundance-weighted average density.

Standard errors use the delta method. Confidence intervals for the index and effective area occupied are calculated on the log scale, so they are asymmetric and always positive. Those for the center of gravity and weighted average are calculated on the original scale.

More generally, `area` is any multiplier on the response-scale predictions. For example, binomial or beta-binomial models fit to proportions (with `weights` giving the number of trials) predict proportions per trial. Setting `area` to a standard number of trials (e.g., hooks per longline set) gives an index of expected counts. The fitting `weights` are not reused automatically.

Value

A data frame with one row per time step (time column named as in the fitted model), a `type` column, and:

- `get_index()` and `get_index_split()`: `est` (the index), `lwr` and `upr` (confidence interval), `log_est` (log of `est`), `se` (standard error of `log_est`), and `se_natural` (standard error of `est`).
- `get_cog()`: with `format = "long"` (default), two rows per time step, one per coordinate (`coord` is `"X"` or `"Y"`), with `est`, `lwr`, `upr`, and `se` (standard error of `est`). With `format = "wide"`, one row per time step with these columns suffixed `_x` and `_y`.
- `get_eao()`: `est`, `lwr`, `upr`, `log_est`, and `se` (standard error of `log_est`).
- `get_weighted_average()`: `est`, `lwr`, `upr`, and `se` (standard error of `est`).

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 center 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()` for derived quantities from simulation draws.

Examples

```
if (ggplot2_installed()) {
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))

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

# the same, calculated in 2 chunks of years;
# slower but uses less memory for very large grids:
ind <- get_index_split(m, newdata = nd, nsplit = 2, bias_correct = TRUE)

# center of gravity:
cog <- get_cog(m, newdata = nd, 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(m, newdata = nd)
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(m, newdata = nd, 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/polygon 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 default `function(x, area) x + log(area)` 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 default `function(x) sum(exp(x))` 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`:

- the name of the time column supplied to `sdmTMB()` (e.g., `year`)
- `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`, a long-format data frame of index draws:

- the name of the time column supplied to `sdmTMB()` (e.g., `year`)
- `.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_prior_parameters` *Inspect custom priors*

Description

`get_prior_parameters()` lists the parameters that custom priors can refer to: the elements of `par` and `theta` passed to the `custom` and `custom_log_jacobian` functions in `sdmTMBpriors()`, with their values. Use it with a fitted model or one set up with `do_fit = FALSE`.

Usage

```
get_prior_parameters(object, random = FALSE)

get_prior_densities(object)
```

Arguments

object	An sdmTMB() model fit with the RTMB backend. For get_prior_densities(), it must have custom priors.
random	Include the random effects (e.g., random field values)? There are often many of them.

Details

get_prior_densities() evaluates the custom prior functions at the estimated parameters, including random effects at their conditional modes, and returns the log density of each term. Like the objective, it evaluates custom_log_jacobian only if bayesian = TRUE.

par holds the raw parameters, with the names and shapes used internally. Matrix columns are model components (e.g., the two parts of a delta model); the second component's main effects are in b_j2. theta holds natural-scale transformations of par, such as phi = exp(ln_phi) and range = sqrt(8) / kappa.

Elements of par that are fixed, by map or because the model doesn't use them, are left out, since a prior on them would have no effect. All elements of theta are listed, including transformations of parameters this model doesn't use (e.g., tweedie_p in a model without a Tweedie family). Check that the par elements underlying a theta element are listed before putting a prior on it. Standard deviations of random fields the model doesn't include (e.g., sigma_O without a spatial field) are zero.

These names and shapes are internal and could change in future versions of sdmTMB. Check them with get_prior_parameters() when writing custom priors.

Value

get_prior_parameters(): a data frame with one row per element:

- expression: how to refer to the element in a custom prior function;
- list, name: "par" or "theta", and the element's name;
- label: the coefficient name, where available;
- value: the estimate (or starting value if the model isn't fit);
- status: for par, "estimated", "random", or "shared" (estimated jointly with other elements via map); "derived" for theta.

get_prior_densities(): a data frame with one row per term added to the objective: type ("density", or "log_jacobian" if bayesian = TRUE), term (the returned name, or its position), and log_density.

See Also

sdmTMBpriors()

Examples

```
fit <- sdmTMB(density ~ depth_scaled, data = pcod_2011,
  family = tweedie(), mesh = pcod_mesh_2011,
  control = sdmTMBcontrol(backend = "rtmb"),
  priors = sdmTMBpriors(custom = function(par, theta) {
    c(depth = RTMB::dnorm(par$b_j[2], 0, 1, log = TRUE))
  }))
get_prior_parameters(fit)
get_prior_densities(fit)
```

---

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 2.5% 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_areal_domain` *Create an areal spatial domain for SAR/CAR models*

Description

[Experimental]

Build an areal domain object with adjacency weights and unit labels. The returned object can be supplied to `mesh` in `sdmTMB()` in areal SAR/CAR workflows.

Usage

```
make_areal_domain(
  spatial_domain,
  space_column = NULL,
  id_column = NULL,
  adjacency = c("rook", "queen")
)
```

Arguments

`spatial_domain`	A named `igraph` object or an `sf`/`sfc` polygon object.
`space_column`	Column name in model data that identifies areal unit membership. For `sf` polygon input, defaults to `id_column` when `id_column` is supplied; otherwise defaults to `"area"`.

id_column	Optional column name in an sf polygon spatial_domain containing areal unit IDs. If omitted for sf polygons, stable IDs are generated.
adjacency	Polygon adjacency type for sf polygon input: "rook" for shared edges or "queen" for any touching boundary.

Value

A list with class c("sdmTMBareal", "sdmTMBdomain").

Examples

```
data(ohio_df)
data(ohio_sf)

domain <- make_areal_domain(ohio_sf, id_column = "county")
domain$n_s
domain$unit_names


fit <- sdmTMB(
  cases ~ pct_male,
  data = ohio_df,
  mesh = domain,
  spatial_model = "car",
  family = poisson(link = "log"),
  offset = log(ohio_df$pop)
)
```

---

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 generally be in a projected CRS with distance-based units. 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 versions without the _inla suffix.
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)
```

`make_zero_one_map` *Fix factor-level intercepts for groups with all zeros or ones*

Description

A grouping factor level (e.g. factor(year), but also possibly a region, stratum, region-by-year interaction, or any other factor level) with all-zero observations can drive the corresponding factor-coded coefficient to `-Inf`, causing convergence problems. In delta/hurdle and binomial or beta-binomial models, an all-positive or all-success level can similarly drive the encounter coefficient to `+Inf`. Tweedie will have the same issue with all zeros.This helper returns `map` and `start` lists suitable for passing through `sdmTMBcontrol()` to fix those problem coefficients at large positive/negative values. This mirrors an approach taken in **VAST**.

Usage

```
make_zero_one_map(formula, data, group, family, weights = NULL, value = 20)
```

Arguments

`formula`	The fixed-effect formula to be used in `sdmTMB()`. For delta models, either a single formula applied to both linear predictors, or a `list()` of two formulas. The formula should contain `factor(group)` (or `as.factor(group)`, or a pre-converted factor column) so each level has its own column, typically `response ~ 0 + factor(year)`.
`data`	The data frame passed to `sdmTMB()`.
`group`	Character. Name of the grouping column in `data` whose levels should be inspected for all-zero/all-positive observations. Most commonly the time column.
`family`	The sdmTMB family object (e.g., `tweedie()`, `delta_gamma()`).
`weights`	Optional numeric vector of trial sizes for non-delta binomial or beta-binomial models fit with proportion data. This should match the `weights` argument passed to `sdmTMB()`. Ignored for other families.
`value`	Positive numeric. Magnitude at which to fix the intercept; `plogis(20)`, `plogis(-20)`, and `exp(-20)` are all within ~2e-9 of 0 or 1, which is plenty.

Details

Inspect the returned `all_zero_levels` and `all_one_levels` elements before fitting to make sure the detected problem levels match your expectation.

Although the motivating use case is years in fisheries index standardization (e.g., `density ~ 0 + factor(year)`), the same mechanism applies to any factor whose levels each get a column in the design matrix. For an interaction such as region by year, create a combined column (e.g., `data$region_year <- interaction(data$region, data$year, drop = TRUE)`) and use `~ 0 + region_year`.

For Poisson-link delta families (e.g., `delta_gamma(type = "poisson-link")`), the first linear predictor controls both the encounter probability and the positive rate, so fixing it shifts the meaning of the second-linear-predictor coefficients in those levels. The helper still works for all-zero levels (the large negative value makes encounter probability ~0, and the level contributes ~0 to the index), but emits a message for all-positive levels. Although this affects interpretation of the coefficients, the overall combined prediction remains OK.

Value

A named list with elements:

`map` A list with `b_j` (and `b_j2` for delta families) factors suitable for `sdmTMBcontrol(map = ...)`.

`start` A list with `b_j` (and `b_j2`) starting-value vectors suitable for `sdmTMBcontrol(start = ...)`.

`all_zero_levels` Group levels with all-zero observations.

`all_one_levels` Group levels with all-positive observations for delta families, or all-success observations for non-delta binomial/beta-binomial families. Empty for other families.

See Also

`sdmTMBcontrol()`, `sdmTMB()`

Examples

```
d <- pcod
d$density[d$year == 2003] <- 0 # force an all-zero year

m <- make_zero_one_map(
  formula = density ~ 0 + factor(year),
  data = d,
  group = "year",
  family = tweedie()
)
m$all_zero_levels
m$all_one_levels
m$start
m$map


mesh <- make_mesh(d, c("X", "Y"), cutoff = 15)
fit <- sdmTMB(
  density ~ 0 + factor(year),
  data = d,
  mesh = mesh,
  time = "year",
  family = tweedie(),
  spatial = "on",
  spatiotemporal = "off", # off for example speed
  control = sdmTMBcontrol(map = m$map, start = m$start) #<
)
fit
```

```
coef(fit)
```

`nonlocal_terms` *Non-local covariate terms*

Description

Wrappers that mark covariates in the `nonlocal_formula` argument of `sdmTMB()` and `sdmTMB_simulate()`. They are not called directly. `diffusion(x)` spreads covariate `x` across space via an SPDE diffusion operator with estimated scale `kappaS_nl`. `time_lag(x)` carries `x` forward in time via $z_t = (x_t + \kappa_T z_{t-1})/(1 + \kappa_T)$ with estimated `kappaT_nl` and first-order autocorrelation $\rho_T = \kappa_T/(1 + \kappa_T)$. When both wrap the same covariate (`~ diffusion(x) + time_lag(x)`), they select one joint space-time operator with a single transformed predictor and coefficient.

Usage

```
diffusion(x)

time_lag(x, start = c("stationary", "zero"))
```

Arguments

`x`	A bare covariate name found in `data` or `nonlocal_data`.
`start`	The transformed state before the first time slice. `"stationary"` (default) assumes the covariate was held at its first time slice beforehand, so the state starts at the operator's fixed point: the first slice itself for a time lag, or its spatial diffusion for the joint operator. Results are then invariant to shifting the covariate by a constant. `"zero"` starts from zero as in Thorson et al. (2026), which shrinks early time slices toward zero (e.g., $z_1 = (1 - \rho_T)x_1$ for a time lag alone) and makes results depend on how the covariate is centered: centering at a constant `c` is equivalent to assuming the covariate equaled `c` before the first time slice.

Value

These functions error if called outside `nonlocal_formula`.

See Also

`sdmTMB()`, `plot_nonlocal_covariate()`, and the non-local covariates vignette.

Examples

```
# Pass as `nonlocal_formula` in sdmTMB(); see ?plot_nonlocal_covariate
nonlocal_formula <- ~ diffusion(x) + time_lag(x, start = "zero")
nonlocal_formula
```

`ohio_df` *Ohio lung cancer mortality data, reshaped for spatiotemporal examples*

Description

A derived version of `geodaData::ohio_lung`, reshaped into county-year and county geometry formats for package examples and vignettes.

Usage

```
ohio_df

ohio_sf
```

Format

`ohio_df`: A county-year data frame with columns:

county County name.

year Observation year.

cases Observed lung cancer deaths.

pop Population at risk.

pct_male Proportion of the population that is male.

`ohio_sf`: An `sf` object with columns:

county County name.

geometry County polygon geometry.

Source

Derived from `geodaData::ohio_lung`, originally distributed in the CRAN package `geodaData` under CC0. The original dataset is based on the GeoDa Center Ohio lung cancer mortality dataset.

Examples

```
data(ohio_df)
head(ohio_df)
data(ohio_sf)
head(ohio_sf)
```

pcod *Example fish survey data*

Description

Example fish survey datasets used throughout the package documentation and examples.

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 2 x 2 km 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 corresponding to `yelloweye`. A data frame.

`wcvi_grid`: A survey domain grid corresponding to `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_nonlocal_covariate` *Plot Covariate-Diffusion Diagnostics*

Description

Visualize fitted covariate-diffusion transforms or impulse-response kernels for one selected covariate-diffusion term. By default, values are plotted at mesh vertices with the mesh edges shown in light grey. Values can also be evaluated at supplied `newdata` coordinates and plotted as points or a raster.

Usage

```
plot_nonlocal_covariate(
  object,
  component,
  newdata = NULL,
  type = c("point", "raster"),
  covariate = NULL,
  time_value = 1,
  n_steps = 1L,
  common_scale = TRUE
)

plot_nonlocal_kernel(
  object,
  component,
  newdata = NULL,
  type = c("point", "raster"),
  covariate = NULL,
  time_value = 1,
  n_steps = 3L,
  common_scale = FALSE
)
```

Arguments

`object`	A fitted `sdmTMB()` model with `nonlocal_formula`.
`component`	Covariate-diffusion component name. Must be one of `"diffusion"`, `"time_lag"`, or `"combined"`. `"combined"` plots the fitted joint operator and requires both wrappers for `covariate`.

newdata	Optional data frame with x/y coordinate columns matching the fitted mesh. If supplied, values are evaluated at the unique newdata coordinates. If NULL, values are evaluated at mesh vertices.
type	Plot type: "point" or "raster". "raster" requires newdata.
covariate	Optional covariate name from nonlocal_formula. Required when multiple lag covariates were fitted.
time_value	Optional time slice to plot or use for the impulse. Supply either a modeled time value or a 1-based time index. Defaults to 1.
n_steps	Number of transformed slices to plot starting at time_value.
common_scale	Should the plotted panels share a common color scale? Defaults to TRUE for plot_nonlocal_covariate() and FALSE for plot_nonlocal_kernel(). component = "time_lag" alone likely needs common_scale = TRUE to make sense.

Details

plot_nonlocal_covariate() visualizes the original mesh-vertex covariate field and its fitted covariate-diffusion transform for one selected covariate time slice across one or more lagged output time slices. When both diffusion() and time_lag() were fitted for the same covariate, component = "combined" visualizes the single stationary joint operator represented by that covariate's transformed prediction column.

plot_nonlocal_kernel() visualizes an impulse entering and diffusing through one covariate-diffusion component.

Value

A ggplot object.

Examples

```
# Simulate some data for fitting:
set.seed(1)
n_t <- 6
n_sites <- 80
sites <- data.frame(X = runif(n_sites), Y = runif(n_sites))
dat <- data.frame(
  X = rep(sites$X, times = n_t),
  Y = rep(sites$Y, times = n_t),
  year = rep(seq_len(n_t), each = n_sites)
)
dat$x1 <- as.numeric(scale(
  sin(2 * pi * (dat$X + dat$year / 6)) +
    cos(2 * pi * (dat$Y - dat$year / 8)) +
    0.4 * sin(4 * pi * dat$X) * cos(dat$year / 2) +
    rnorm(nrow(dat), sd = 0.15)
))
mesh <- make_mesh(dat, xy_cols = c("X", "Y"), cutoff = 0.12)
nonlocal_data <- merge(
  data.frame(year = seq_len(n_t)),
```

```
  setNames(as.data.frame(mesh$mesh$loc[, 1:2]), c("X", "Y"))
)
nonlocal_data$x1 <- as.numeric(scale(
  sin(2 * pi * (nonlocal_data$X + nonlocal_data$year / 6)) +
    cos(2 * pi * (nonlocal_data$Y - nonlocal_data$year / 8)) +
    0.4 * sin(4 * pi * nonlocal_data$X) * cos(nonlocal_data$year / 2)
))
sim <- simulate_new(
  formula = ~ 1,
  data = dat,
  mesh = mesh,
  time = "year",
  family = gaussian(),
  spatial = "off",
  spatiotemporal = "off",
  range = 0.3,
  sigma_O = 0,
  sigma_E = 0,
  phi = 0.1,
  B = c(0, 0.7),
  nonlocal_formula = ~ diffusion(x1) + time_lag(x1),
  nonlocal_data = nonlocal_data,
  lags_kappaS = 4.4,
  lags_rhoT = 0.3,
  seed = 123
)
dat$observed <- sim$observed

# Fit the model:
fit <- sdmTMB(
  observed ~ 1,
  data = dat,
  mesh = mesh,
  time = "year",
  spatial = "off", # keeping example simple
  spatiotemporal = "off", # keeping example simple
  family = gaussian(),
  nonlocal_formula = ~ diffusion(x1) + time_lag(x1), #<
  nonlocal_data = nonlocal_data
)

plot_nonlocal_covariate(
  fit,
  covariate = "x1",
  component = "diffusion"
)
plot_nonlocal_covariate(
  fit,
  covariate = "x1",
  component = "time_lag",
  time_value = 1,
  n_steps = 2
)
```

```
plot_nonlocal_covariate(
  fit,
  covariate = "x1",
  component = "combined",
  time_value = 1,
  n_steps = 2
)
plot_nonlocal_kernel(
  fit,
  covariate = "x1",
  component = "diffusion"
)
plot_nonlocal_kernel(
  fit,
  covariate = "x1",
  component = "time_lag",
  time_value = 1,
  n_steps = 2,
  common_scale = TRUE #<
)
plot_nonlocal_kernel(
  fit,
  covariate = "x1",
  component = "combined",
  time_value = 1,
  n_steps = 2
)
```

`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_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.
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 graphics::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 a model fitted with sdmTMB(), either for the fitted data or for new data (e.g., a prediction grid). At new locations, the random fields are interpolated from their estimated values at the mesh vertices. Besides the overall prediction (est), the output separates the linear predictor into the contribution of the spatial and spatiotemporal random fields and everything else (see Value).

Usage

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

Arguments

object	A model fitted with sdmTMB().
newdata	A data frame to predict on. If NULL (default), predictions are for the fitted data. Must contain the columns used in the model formulas (including spatial_varying and time_varying), the coordinate columns used to build mesh, and, for spatiotemporal models, the time column. Time values must be among those in the fitted data or extra_time. Factor levels must have been seen in fitting, except for random effect grouping factors (see allow_new_levels).
type	The scale of est: "link" (default; the linear predictor) or "response" (the expected value of the response). For delta models, "response" gives the expected value combining both components. Only est, est1, and est2 depend on type; the other columns are always on the link scale. Standard errors (se_fit) require "link".
se_fit	Logical: calculate standard errors of est on the link scale (returned as est_se)? These account for uncertainty in both fixed and random effects but can be slow for many rows when random fields are included. For faster uncertainty, exclude the random fields with re_form = NA or summarize draws from nsim. Use nsim for response-scale uncertainty.
re_form	Include the spatial and spatiotemporal random fields (including spatially varying coefficient fields)? NULL (default) includes them. NA or ~ 0 excludes them for population-level predictions, so that est equals est_non_rf; the random field columns are then omitted. Often used with se_fit = TRUE to plot covariate effects. IID random effects are set separately with re_form_iid. Both

also apply to derived quantities such as `get_index()` when passed through its `predict_args`.

`re_form_iid` Include the IID random intercepts and slopes (e.g., `(1 | g)` in `formula`)? `NULL` (default) includes them. `NA` or `~ 0` sets them all to zero. Excluding only some of them is not yet supported.

`allow_new_levels` Allow levels of random effect grouping factors in `newdata` that were not seen in fitting? Rows with a new level get a random effect value of zero (a population-level prediction). `TRUE` allows them silently, `FALSE` gives an error (as with `allow.new.levels` in **lme4** and **glmmTMB**), and `NULL` (default) allows them with a warning. Not relevant if `re_form_iid` excludes the random effects. Grouping columns in `newdata` must be factors.

`nsim` Number of draws. If `> 0`, returns a matrix of draws instead of a data frame (see Value). Each draw takes the fixed and random effects from their approximate joint (multivariate normal) distribution and computes `est` (or `sims_var`). Summarize across draws (e.g., `apply(x, 1, sd)` or quantiles) for uncertainty on any scale, or carry the draws through to derived quantities. Usually much faster than `se_fit = TRUE` for models with random fields.

`sims_var` Which quantity to return when `nsim > 0`: `"est"` (default; on the scale set by `type`) or one of the link-scale columns `"est_non_rf"`, `"est_rf"`, `"omega_s"`, `"zeta_s"`, or `"epsilon_st"` (see Value). The model must include the term. For delta models, these come from the component set by `model` (default: the first). With more than one spatially varying coefficient, `"zeta_s"` returns a list of matrices, one per coefficient. For other quantities, use `return_tmb_report = TRUE`.

`sample_fe` Logical. When `nsim > 0`, draw the fixed effects and other estimated parameters along with the random effects (`TRUE`, default)? If `FALSE`, these are held at their estimates and only the random effects (random fields, IID random effects, time-varying coefficients, and smoother coefficients) are drawn, conditional on those estimates. This ignores parameter uncertainty and so typically gives narrower intervals. Fixed effects are held at their estimates even with REML. See also the same argument in `project()`.

`model` For delta models, what `est` (and `est_se` or the draws from `nsim` or `mcmc_samples`) refers to: `NA` (default) combines both components, `1` gives the first (binary) component, and `2` the second (positive) component. The data frame output always also includes each component as `est1` and `est2`. Ignored for other models. See the delta-model vignette.

`offset` A numeric vector of offset values, one per row of `newdata` (not a column name). If `NULL` (default), predictions for the fitted data use the fitted offset, and predictions with `newdata` use an offset of 0, i.e., predictions per unit of the offset (e.g., density rather than catch when the offset is log area swept).

`mcmc_samples` A matrix of posterior samples from a model passed to **tmbstan** (see `bayesian` in `sdmTMB()`), as returned by `sdmTMBextra::extract_mcmc()` in the sdmTMBextra package. If supplied, returns a matrix of posterior draws in the same form as with `nsim`. If `nsim` is also supplied, the last `nsim` samples are used. See the Bayesian vignette.

nonlocal_newdata
: An optional data frame of the nonlocal_formula covariates to predict with instead of those used in fitting (e.g., for a scenario with different conditions). Same requirements as nonlocal_data in sdmTMB(). The rows of newdata (or the fitted data) still set where and when predictions are made; this argument only supplies the covariate values that are diffused or lagged. If NULL (default), the covariates from nonlocal_data in sdmTMB() are reused if supplied (so newdata need not contain those columns); otherwise they come from newdata.

return_tmb_object
: **[Deprecated]** Logical. If TRUE, include the TMB object in a list-format output. Instead, pass the fitted model and newdata directly to get_index() or get_cog().

return_tmb_report
: Logical: return the TMB report (a list of all reported quantities at the estimated parameters) instead of a data frame? With nsim > 0 or mcmc_samples, a list with one report per draw. Mainly for developers.

return_tmb_data
: Logical: return the data list passed to TMB instead of predicting? Used internally.

...
: Unused.

Value

By default, newdata (or the fitted data) with these columns added:

- est: The prediction on the scale set by type.
- est_se: The standard error of est on the link scale, if se_fit = TRUE.
- est_non_rf: The linear predictor excluding the spatial and spatiotemporal random fields: fixed effects, smoothers, IID random effects, time-varying coefficients, and the offset.
- est_rf: The sum of all random field terms, including spatially varying coefficient fields multiplied by their covariates. On the link scale, est_non_rf + est_rf equals est.
- omega_s: The spatial random field.
- zeta_s_<x>: The spatially varying coefficient field for covariate <x> in spatial_varying: the local deviation from the average coefficient, not multiplied by the covariate.
- epsilon_st: The spatiotemporal random field.
- nl_*: The transformed covariate values for each nonlocal_formula term.

Columns for terms not in the model are left out, as are the random field columns with re_form = NA.

Delta models instead return each column (other than est and est_se) per component, with suffixes 1 and 2 for the first and second components (e.g., est1, est2, omega_s1, omega_s2). The identity above then holds within each component, and est combines the components (or gives one of them; see model).

If nsim > 0 or mcmc_samples is supplied: a matrix with one row per row of newdata (or the fitted data) and one column per draw. Row names are the time values.

If return_tmb_object = TRUE (deprecated): a list with elements data (the data frame above), report (the TMB report), obj (the TMB object from the prediction), and fit_obj (the fitted model).

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 on 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 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 without random fields (fixed effects only here)") +
  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
# value. For these population-level predictions, if time isn't also a fixed
# effect, 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 AR(1) 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 without random fields (fixed effects only here)") +
  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

Project 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,
  sample_fe = TRUE,
  sample_historical_re = TRUE,
  sample_future_re = TRUE,
  future_re = c("include", "zero", "fix"),
  silent = FALSE,
  sims_var = NULL,
  return_tmb_report = FALSE,
  ...
)
```

Arguments

object	A fitted model from sdmTMB().
newdata	A new data frame to predict on. It should contain both the historical time elements and any new time elements to project to. Time values must be numeric and align with the constant cadence in the fitted model. Missing intermediate future time elements are simulated.
nsim	Number of projection simulations.
sample_fe	Logical. Sample uncertainty in the estimated model parameters, including regression coefficients, covariance parameters, and dispersion parameters.
sample_historical_re	Logical. Sample uncertainty in random effects associated with the fitted historical period.
sample_future_re	Logical. Simulate random effects in the projection period. When FALSE, dynamic effects are propagated at their conditional means. Ignored when future_re is "zero" or "fix".
future_re	Treatment of spatiotemporal random effects and time-varying coefficients in the projection period. "include" preserves the usual process dynamics, "zero" sets them to zero, and "fix" holds them at their value in the final historical time step. Historical effects are unchanged.
silent	Logical. Suppress progress messages?
sims_var	Optional single element to extract from the **TMB** simulation report. If NULL, the default, the usual named projection list is returned. If supplied, use a native projection report name such as "proj_eta" or "proj_epsilon_st_A_vec"; the simulation dimension is last. See also return_tmb_report.
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. Prediction-path elements retain their native proj_* names. 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 `sims_var` is supplied, an array containing that report element, with the simulation dimension last. If `return_tmb_report = TRUE`, a list of **TMB** reports returned by `simulate()`. Run `names()` on an element of the output to see the available `sims_var` options. Unprefixed observation-path reports such as `eta_i` describe the historical fitting data; projected `newdata` values are in the corresponding `proj_*` reports.

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

```
if (ggplot2_installed()) {
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)

# Keep parameter and historical-state uncertainty, while propagating future
# dynamic effects at their conditional means:
out_mean_future <- project(
  fit2, newdata = proj_grid, nsim = 20, sample_future_re = FALSE
)

# 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 the documentation site. See the Details section below for notes on residual types.

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 or matrix 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 is the same sample as `obj$env$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 residual type in **sdmTMB** (and were called `mle-laplace`). They are available for backward compatibility and for research purposes but 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 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

Run extra optimization on an already fitted object

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. object is left unchanged.

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 **TMB** (Template Model Builder), by default through the **RTMB** package. Spatial and spatiotemporal Gaussian random fields are approximated with the SPDE (stochastic partial differential equation) approach, which represents them as Gaussian Markov random fields on a mesh. This allows for efficient modelling of data that are correlated in space and/or time. Areal spatial and spatiotemporal models (simultaneous or conditional autoregressive; SAR or CAR) are also available. 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"),
  offset = NULL,
  weights = NULL,
  spatial_model = c("spde", "sar", "car"),
  share_range = TRUE,
  anisotropy = FALSE,
  extra_time = NULL,
  spatial_varying = NULL,
  time_varying = NULL,
  time_varying_type = c("rw", "rw0", "ar1"),
  dispformula = ~1,
  nonlocal_formula = NULL,
  nonlocal_data = NULL,
  knots = NULL,
  censored_upper = NULL,
  distribution_column = NULL,
  silent = TRUE,
  reml = FALSE,
  priors = sdmTMBpriors(),
  bayesian = FALSE,
  control = sdmTMBcontrol(),
  previous_fit = NULL,
  do_fit = TRUE,
  experimental = NULL,
  do_index = FALSE,
  predict_args = NULL,
  index_args = NULL
)
```

Arguments

formula	A model formula. Random intercepts and slopes use **lme4** syntax, e.g., `(1 | g)`, `(0 + depth | g)`, or `(1 + depth | g)`, where `g` is a character or factor column. As in **lme4**, intercepts and slopes within a term are correlated. Penalized smooths use **mgcv** syntax, e.g., `s(depth)`, and threshold terms use `breakpt()` or `logistic()`. For delta models, optionally a list of two formulas. See Details.
data	A data frame. Rows with missing values in the response or any variable the model uses (including `weights` and `offset`) are omitted before fitting, as with `na.action = na.omit` in `stats::glm()`. The rows used are stored in the returned object as `data`.
mesh	An object from `make_mesh()` for `spatial_model = "spde"` or from `make_areal_domain()` for `"sar"` or `"car"`.
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 calculate derived quantities downstream (e.g., `get_index()` or `get_cog()`), then supply the time argument.

`family` A family object specifying the response distribution and link: `gaussian()`, `Gamma()`, `binomial()`, `poisson()`, or one of the additional families listed in Families (e.g., `tweedie()`, `nbinom2()`, `delta_gamma()`). See 'Binomial families', 'Censored families', and 'Delta/hurdle models' in Details. Experimental multi-family models take a named list of families; `distribution_column` then assigns each row to one of them. See the multi-family vignette for supported combinations.

`spatial` Estimate spatial random fields? Options are `'on'` / `'off'` or equivalently `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 spatiotemporal 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'`.

`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 except for Poisson-link delta models, where it also enters the occurrence-probability calculation. Usually a log transformed variable.

`weights` An optional numeric vector (not a column name) of weights on each observation's contribution to the likelihood. As in **glmmTMB**, weights need not sum to one and are not rescaled. For binomial-type families with a proportion response, `weights` gives the number of trials instead; see 'Binomial families' in Details.

`spatial_model` Spatial process model. `"spde"` uses the default continuous-space SPDE approximation. `"sar"` and `"car"` use areal spatial autoregressive models with an areal domain supplied to `mesh`. Capitalization is ignored.

`share_range` Logical: estimate a single range parameter shared by the spatial and spatiotemporal fields (`TRUE`, default) or separate range parameters (`FALSE`)? For delta models, can be a list, e.g., `list(TRUE, FALSE)`. Applies to SPDE models only.

`anisotropy` Logical: allow for anisotropy (spatial correlation that is directionally dependent)? See `plot_anisotropy()`. Applies to SPDE models only and is shared by both components of a delta model.

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

`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. 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 a factor, `~ 0 + f` gives every level its own field, which would duplicate the spatial intercept field, so set `spatial = 'off'` to match. Structure is shared in delta models.

`time_varying` An optional one-sided formula of coefficients that vary through time following the process set by `time_varying_type`. Whether the same covariates should also appear in `formula` depends on that type. Shared by both components of a delta model.

`time_varying_type`

The process for `time_varying` coefficients:

- `'rw'` (default): a random walk with the first value estimated freely. Do not also include these covariates in `formula`, or the model is not identifiable; e.g., with `time_varying = ~ 1`, use `formula = y ~ 0 + ...`.
- `'rw0'`: a random walk whose first value has a mean-zero normal prior.
- `'ar1'`: a stationary first-order autoregressive process with mean zero.

For `'rw0'` and `'ar1'`, include the same covariates in `formula`; the time-varying process then describes deviations from that average effect. Shared by both components of a delta model.

`dispformula` A one-sided formula for the dispersion parameter (e.g., `phi`). The default, `~ 1`, estimates a single value. Ignored for families without a dispersion parameter (e.g., binomial or Poisson). In delta models, applies to the positive component. Not yet available for multi-family models or truncated negative binomial families.

`nonlocal_formula`

An optional one-sided formula of non-local covariate effects: `diffusion()` for an effect of conditions in the surrounding area, and `time_lag()` for an effect of conditions in previous time steps, e.g., `~ diffusion(x) + time_lag(x)`. Using both on the same covariate gives one combined effect that spreads over both space and time; different covariates give separate effects. See `time_lag()` for how the lag starts (`start`). If `time` is `NULL`, a spatial-only covariate is held constant across time slices. See the non-local covariates vignette, which also explains the reported diffusion scales (MSDK and RMSDK).

`nonlocal_data` An optional data frame supplying the `nonlocal_formula` covariate(s) at a different resolution and/or coverage than `data` (e.g., a finer grid, or one spanning `extra_time` slices). Must contain the mesh `xy_cols`, the diffusion covariate columns, and the `time` column if `nonlocal_formula` is time-indexed (`time_lag()` terms, or `diffusion()` terms with `time` specified). In that case, it must cover every fitted (`+ extra_time`) time slice. Defaults to `NULL`, in which case `data` is used.

`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(`

censored_upper Upper bounds for the censored families (see Families): a numeric vector or the name of a column in data. Each response is a count between its observed value and this bound. A bound equal to the response is uncensored, a larger bound is interval-censored, and Inf is right-censored (capped at the number of trials for censored_binomial() and censored_betabinomial()). Non-integer bounds are rounded down. For left censoring (e.g., fewer than 5), use a response of 0 and a bound of 4.

distribution_column For experimental multi-family models, the name of the column in data mapping each row to a family in the named family list. See the multi-family vignette for the supported family and method combinations.

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

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.

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

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.

control Optimization control options via sdmTMBcontrol().

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)?

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

do_index **[Deprecated]** Do index standardization calculations while fitting? Instead, fit the model and then pass the fitted model and newdata directly to get_index(), get_cog(), get_eao(), or get_weighted_average(). If TRUE, then predict_args must have a newdata element supplied and area can be supplied to index_args.

predict_args **[Deprecated]** A list of arguments to pass to predict.sdmTMB() **if** do_index = TRUE.

index_args **[Deprecated]** A list of arguments to pass to get_index() **if** do_index = TRUE. Currently, area and derived_link are supported.

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 four ways: (1) the response may be a factor (success is interpreted as any level other than the first level), (2) the response may be binary (0/1), (3) the response can be a matrix of form `cbind(success, failure)`, and (4) the response may be 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 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. For delta families, the threshold applies to both components, and threshold terms are not available if `formula` is a list.

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; `time_varying` and spatiotemporal random-walk or AR(1) fields include 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_model$tmb_data$X_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 (see the list under `family`). Delta families with `type = "poisson-link"` use a Poisson-link parameterization instead of a classic hurdle model; see the Poisson-link vignette. 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), and `dispformula` applies to the positive component only. Furthermore, there are currently limitations if specifying two formulas as a list: smoothers and random effect terms must be identical between the two formulas, and threshold effects 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.

Censored families

Censored families treat some observations as known only to lie within a range, e.g., catch counts on longlines where competition for hooks hides the true number caught. Supply the bounds with `censored_upper`. All but `censored_poisson()` need the RTMB backend (the default). See Families and the hook competition vignette.

Areal models

For data on areal units (e.g., grid cells or management areas) rather than point locations, build a domain with `make_areal_domain()`, pass it to `mesh`, and set `spatial_model = "sar"` or `"car"`. `share_range` and `anisotropy` do not apply. See the areal 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:

- `model`: output from `stats::nlminb()`
- `sd_report`: output from `TMB::sdreport()` or `RTMB::sdreport()`
- `gradients`: gradients of the marginal log likelihood with respect to each fixed effect
- `data`: the data used in fitting (rows with missing values removed)

- spde: the object supplied to mesh
- family: the family object, including the inverse link function family$linkinv()
- backend: "rtmb" or "tmb"; see sdmTMBcontrol()
- tmb_obj: the objective function object from TMB::MakeADFun() or RTMB::MakeADFun()
- tmb_data, tmb_params, tmb_map: the data, parameter, and map lists passed to 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. 2022. 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. 2023. Evaluating drivers of spatiotemporal variability in individual condition of a bottom-associated marine fish, Atlantic cod (*Gadus morhua*). ICES Journal of Marine Science. 80(5): 1539–1550.

Non-local covariates:

Lindmark, M., Anderson, S.C., and Thorson, J.T. 2026. Estimating scale-dependent covariate responses using two-dimensional diffusion derived from the stochastic partial differential equation method. Methods in Ecology and Evolution 17: 207–218. doi:10.1111/2041210X.70177.

Thorson, J.T., Anderson, S.C., and Lindmark, M. 2026. Temperature carryover effect revealed for marine fishes using spatio-temporal distributed lag models. EcoEvoRxiv. doi:10.32942/X2W95P.

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", 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

# Correlated random intercepts and slopes 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_method = c("auto", "direct"),
  multiphase = TRUE,
  profile = FALSE,
  get_joint_precision = TRUE,
  parallel = getOption("sdmTMB.cores", 1L),
  suppress_nlminb_warnings = TRUE,
  collapse_spatial_variance = FALSE,
  collapse_threshold = deprecated(),
  collapse_spatial_variance_threshold = 0.01,
  collapse_spatiotemporal_ar1 = FALSE,
  collapse_ar1_threshold = 0.01,
  sar_weight_style = c("row", "raw"),
  get_rsr = FALSE,
  backend = getOption("sdmTMB.backend", "rtmb"),
  ...
)
```

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_modeltmb_objenv$parList(). Elements of start that are specified will replace the default starting values.
map	A named list with factor NAs 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_method	How censored_betabinomial() sums the likelihood. "auto" (default) sums the interval or its complement, whichever has fewer terms, and checks the precision at the estimate, refitting with the direct sum for any rows that fail. "direct" always sums the interval, which can be much slower. The check only applies at the estimate, so consider "direct" when evaluating the likelihood far from it (e.g., MCMC sampling).
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 to 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_spatial_variance_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 the computational cost of one or more model refits if collapsing occurs but can yield a simpler, more stable model and more reliable inference. Default is FALSE for backward compatibility.

collapse_threshold
: Deprecated alias for collapse_spatial_variance_threshold. It will be removed in a future release.

collapse_spatial_variance_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.

collapse_spatiotemporal_ar1
: Logical: should spatiotemporal AR1 fields be automatically simplified when the estimated correlation (rho) is near zero or one? Fields with abs(rho) <= collapse_ar1_threshold are refitted as "iid"; fields with rho >= 1 - collapse_ar1_threshold are refitted as "rw". This does not apply to time_varying effects. Default is FALSE.

collapse_ar1_threshold
: Numeric: distance from zero or one at which a spatiotemporal AR1 correlation is considered to be collapsing. Must be greater than zero and less than 0.5. Only used when collapse_spatiotemporal_ar1 = TRUE. Default is 0.01.

sar_weight_style
: Weight matrix to use for areal SAR models. "row" uses row-normalized weights and is the default. "raw" uses the raw adjacency/weight matrix for direct comparisons to packages that do so.

get_rsr	Experimental option, whether to calculate the restricted spatial regression (RSR) adjusted estimator for covariate responses. If get_rsr = TRUE, these will be available via tidy(fit, effects = "rsr"). See Hanks et al. (2015) and Diaz and Thorson (2025).
backend	Model backend. "rtmb" is the default; set options(sdmTMB.backend = "tmb") to use the C++ TMB template by default. The "rtmb" backend currently supports all families, including delta and multi-family models, with SPDE (isotropic, anisotropic, or barrier) and areal SAR/CAR spatial and spatiotemporal fields, spatially and time-varying coefficients, IID random effects, smoothers, threshold terms, covariate diffusion, restricted spatial regression, priors, and derived indices.
...	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.

References

Restricted Spatial Regression (get_rsr):

Diaz, R.R., and Thorson, J.T. 2025. When and How to Use Restricted Spatial Regression to Separate Environmental Effects from Spatial Confounding. EcoEvoRxiv. doi:10.32942/X28351.

Hanks, E.M., Schliep, E.M., Hooten, M.B., and Hoeting, J.A. 2015. Restricted spatial regression in practice: geostatistical models, confounding, and robustness under model misspecification. Environmetrics 26(4): 243–254. doi:10.1002/env.2331.

Examples

```
sdmTMBcontrol()
```

---

sdmTMBpriors *Prior distributions*

Description

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. In other words, if 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 Matérn 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),
  custom = NULL,
  custom_log_jacobian = NULL
)

normal(location = 0, scale = 1)

halfnormal(location = 0, scale = 1)

gamma_cv(location, cv)

lognormal_prior(meanlog, sdlog)

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_range = 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() or lognormal_prior() priors for any time-varying parameter SDs. Supply a single prior to apply it to all time-varying coefficients or a vector with

one element per coefficient (NA for no prior). In delta models, the priors apply to both components.

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.

custom
: Optional function of (par, theta) returning one or more log density contributions (RTMB backend only; see **Custom priors** below).

custom_log_jacobian
: Optional function of (par, theta) returning the log absolute Jacobian for custom. Only used if bayesian = TRUE. Requires custom.

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).

meanlog
: Mean of the distribution on the log scale.

sdlog
: Standard deviation of the distribution on the log scale.

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

Pass these objects to the priors argument in sdmTMB().

normal() and halfnormal() define normal and half-normal priors that, for now, 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.

`lognormal_prior()` defines a lognormal prior with the same parameterization as `stats::dlnorm()`. The median is `exp(meanlog)`.

Value

A named list with values for the specified priors.

Custom priors

With the RTMB backend (the default), `custom` adds arbitrary log densities to the joint objective. The function is called as `custom(par, theta)`, where `par` is the list of raw (internal) parameters and `theta` is a list of natural-scale versions of them. See `get_prior_parameters()` for their names, shapes, and labels. It must return a numeric scalar, vector, or array of log densities (names are optional), which are summed and subtracted from the negative log likelihood. Custom terms are evaluated inside the objective, so they are included in gradients, Hessians, the Laplace approximation, and standard errors, and may involve random effects. They supplement, rather than replace, the built-in priors and random effect distributions. Use `get_prior_densities()` to see the contributions at the estimated parameters.

The function must:

- use operations and densities that **RTMB** can differentiate, such as `RTMB::dnorm(x, mean, sd, log = TRUE)`; `stats::dnorm()` can't take AD values;
- return *log* densities (remember `log = TRUE`, which can't be checked);
- not branch on parameter values (e.g., `if (par$b_j[1] > 0)`);
- be deterministic and free of side effects;
- also work on ordinary numeric inputs.

Fixed hyperparameters can be defined in the function's enclosing environment. The functions are saved with the fitted model, so keep their environments small (e.g., avoid defining them inside a function holding large data).

Jacobians. With `bayesian = FALSE`, custom priors are penalties: the optimum is the posterior mode on the scale the prior is written on, and no Jacobian is needed. With `bayesian = TRUE` (e.g., for sampling with **tmbstan**), the result of `custom_log_jacobian(par, theta)` is also added. No Jacobian adjustment is ever inferred: without `custom_log_jacobian`, a prior on a transformed parameter (e.g., `theta$phi`) is not a valid prior density on that parameter's natural scale for sampling. For a prior on `phi = exp(ln_phi)`, the log Jacobian is `par$ln_phi`.

Matérn range and SDs. The Matérn `range` and the field SDs (`sigma_O`, `sigma_E`) are more complex than most parameters because they depend on each other: each SD is a function of both `ln_tau_*` and `ln_kappa`. A Jacobian for one of them alone isn't well defined; it needs the joint transformation from (`ln_kappa`, `ln_tau_*`) to (`range`, `sigma_*`), whose log Jacobian is `log(range) + log(sigma_*)`. With a shared range (the default), add `log(range)` once plus `log(sigma_*)` for each field. For priors on these parameters we suggest the built-in PC priors, `pc_matern()` via the `matern_s` and `matern_st` arguments, which apply the Jacobian for you if `bayesian = TRUE`.

Limitations. Custom terms are not evaluated in the first phase of `sdmTMBcontrol(multiphase = TRUE)`, which only finds starting values with random fields turned off. Custom priors do not define a sampler: `simulate.sdmTMB()` and other simulation methods ignore any change that custom terms make to the distribution of random effects. Held-out log likelihoods in `sdmTMB_cv()` exclude all priors. As with the built-in priors, custom terms are part of the objective, so they are included in `stats::logLik()` and `stats::AIC()`.

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)
lognormal_prior(log(0.2), 0.5)
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. To judge whether differences between models are meaningful, pass the output to `loo::elpd()` and compare with `loo::loo_compare()`, which reports the difference and its standard error (see `elpd.sdmTMB_cv()`). Models must use the same folds (`fold_ids`) and the same `predictive` argument. 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,
  predictive = c("mle-mvn", "mle-eb", "joint"),
  nsim = 100L,
  ...
)
```

Arguments

formula	Model formula.
data	A data frame. Rows sdmTMB() omits for missing values are left out of the validation results.
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 of user-supplied fold IDs. Can also be a single string, e.g. "fold_id", naming a variable in data. Ignored if lfo = 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. To save space, the fold models are fit without the joint precision matrix unless control is supplied (see get_joint_precision in sdmTMBcontrol()); predict(..., nsim = ...) recomputes it when needed.
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 `future.globals` in `future.apply::future_lapply()`. Useful if global objects are used to specify arguments such as priors or families.

`predictive` Which predictive density to score held-out data with. `"mle-mvn"` (default) integrates over the random effects' approximate (Laplace) posterior with fixed effects at their maximum likelihood estimates (MLEs). `"mle-eb"` (the previous default) evaluates the likelihood with fixed effects at their MLEs and random effects at their empirical Bayes (EB) estimates. These match `type` in `simulate.sdmTMB()` and `residuals.sdmTMB()`. `"joint"` integrates over the joint approximate posterior of all parameters, as with `nsim` in `predict.sdmTMB()`. `"mle-eb"` ignores uncertainty in the random effects and can be overconfident when random fields are flexible (e.g., fine meshes) or forecast (LFOCV). For models without random effects, `"mle-mvn"` is identical to `"mle-eb"`. See Details.

`nsim` Number of posterior draws per fold if `predictive` is `"mle-mvn"` or `"joint"`. The Monte Carlo log-mean-exp estimate is biased low by $O(1/\texttt{nsim})$; check that results are stable to increasing `nsim`.

`...` 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

Plug-in vs. integrated predictive densities

With `predictive = "mle-eb"`, held-out observations are scored at the conditional mode of the random effects, $\log p(y \mid \hat{u}, \hat{\theta})$, which treats the random fields as known (a plug-in predictive density). With `"mle-mvn"`, the score is $\log E[p(y \mid u, \hat{\theta})]$ over $u \sim N(\hat{u}, H^{-1})$, where H is the inner (Laplace) Hessian of the training fit. With `"joint"`, fixed effects and variance parameters (on their unconstrained scale) are also drawn from the joint precision matrix. Each is averaged over `nsim` draws per fold separately for each held-out observation, so `fold_loglik` is $\sum_i \log E[p(y_i \mid u)]$. If `reml = TRUE`, fixed effects are treated as random effects and are therefore also drawn under `"mle-mvn"`. For Gaussian observations, `"mle-mvn"` is exact given $\hat{\theta}$; for other families it relies on the Laplace approximation's Gaussian approximation to the random effects' posterior.

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 that these are time steps in the order they appear in the data. For example, in the pcod data example below, steps between observations are not always one year, so `lfo_forecast = 2` forecasts two observed time steps ahead, not necessarily two calendar 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`), `"mle-eb"` CV log likelihood (`cv_loglik_mle_eb`; identical to `cv_loglik` if `predictive = "mle-eb"`), 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 `cv_loglik` per fold (sum of pointwise log predictive densities of the held-out data). More positive values indicate better out-of-sample prediction.
- `sum_loglik`: Sum of `fold_loglik` across all folds (total pointwise log predictive density). Use this to compare models; larger values are better.
- `sum_loglik_mle_eb`: Sum of `cv_loglik_mle_eb`.
- `predictive`: The `predictive` argument used for `cv_loglik`.
- `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.

See Also

`elpd.sdmTMB_cv()`, `loo::loo_compare()`, `loo::elpd()`

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

# Compare models fit to the same folds with the loo package:
m_cv3_null <- sdmTMB_cv(
  density ~ 1,
  data = pcod, mesh = mesh,
  family = tweedie(link = "log"),
  fold_ids = m_cv3$data$cv_fold
)
if (requireNamespace("loo", quietly = TRUE)) {
  loo::loo_compare(list(
    depth = loo::elpd(m_cv3),
    null = loo::elpd(m_cv3_null)
  ))
}
```

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

Description

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,
  simulate_re = NULL,
  mle_mvn_samples = c("single", "multiple"),
  mcmc_samples = NULL,
  return_tmb_report = FALSE,
  observation_error = TRUE,
  size = NULL,
  silent = FALSE,
  ...
)
```

Arguments

`object`	An `sdmTMB` model.
`nsim`	Number of response lists to simulate. Defaults to 1.
`seed`	Random number seed. Simulations use the backend the model was fit with; TMB and RTMB fits give different values for the same 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 model.
`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 effects (smoothers, which internally are random effects, will not be simulated as new).

simulate_re — Optional character vector naming the random effects to simulate as new, instead of re_form. Any of "spatial", "spatiotemporal", "spatial_varying", and "time_varying" (matching the corresponding sdmTMB() arguments) and "group_re" (random intercepts and slopes from formula terms such as (1 + x | g)). Random effects not named are held at their values in the parameter vector: the empirical Bayes estimates with type = "mle-eb" or the approximate posterior draw with type = "mle-mvn". For example, simulate_re = "spatiotemporal" with type = "mle-mvn" and mle_mvn_samples = "single" keeps the spatial fields at one posterior draw shared across all nsim while drawing new spatiotemporal fields for each simulation. NULL (the default) uses re_form instead.

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 sdmTMBextra::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, type = "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 backward compatibility. We recommend using `simulate_new()` because 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),
  seed = sample.int(1e+06, 1),
  time_varying = NULL,
  time_varying_type = c("rw", "rw0", "ar1"),
  sigma_V = NULL,
  rho_time = NULL,
  nonlocal_formula = NULL,
  lags_kappaS = NULL,
  lags_rhoT = NULL,
  rho_sar = NULL,
  alpha_car = 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),
  seed = sample.int(1e+06, 1),
  time_varying = NULL,
  time_varying_type = c("rw", "rw0", "ar1"),
  sigma_V = NULL,
  rho_time = NULL,
  nonlocal_formula = NULL,
  lags_kappaS = NULL,
  lags_rhoT = NULL,
  rho_sar = NULL,
  alpha_car = 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() or, for areal SAR/CAR models, make_areal_domain() (with spatial_model = "sar" or "car" passed via ...).
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). If nonlocal_formula is used, include regular fixed-effect coefficients followed by transformed-covariate coefficients in operator order. Same-covariate diffusion() and time_lag() wrappers contribute one joint coefficient.
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. Not used for areal models.
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). A single value.
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. NULL values indicate that the corresponding random effects should be simulated.
seed	Seed number. The TMB and RTMB backends (see the backend argument of sdmTMBcontrol(), passed via control) draw random numbers differently, so the same seed gives different simulated values with each.
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.
nonlocal_formula	An optional one-sided formula describing covariate-diffusion terms to pass to sdmTMB(). Supported wrappers are diffusion() and time_lag(). Same-covariate wrappers select one joint operator and coefficient.
lags_kappaS	Spatial diffusion scale for diffusion() terms. Must be positive and finite. Supply a single value or one value per covariate needing a spatial scale.
lags_rhoT	Temporal diffusion persistence for time_lag() terms. Must satisfy 0 < rhoT < 1. Supply a single value or one value per covariate needing temporal diffusion.
rho_sar	Spatial dependence parameter for spatial_model = "sar"; between -1 and 1.
alpha_car	Spatial dependence parameter for spatial_model = "car"; between 0 and 1.
...	Any other arguments to pass to sdmTMB().

Value

A data frame where:

- The first column is the time variable (if present).
- The second and third columns are the spatial coordinates (or, for areal models, a single column of areal unit IDs).
- omega_s represents the simulated spatial random effects (only if present).
- zeta_s represents the simulated spatially 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", "rsr", "dispersion"),
  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 and time-varying coefficients, if included; behaves like ranef(); see Details), "ran_vcov" (list of variance covariance matrices for the random effects, by model and group), or "rsr" (Restricted Spatial Regression fixed-effect coefficients adjusted for spatial confounding with the random fields; Hanks et al. 2015; Diaz and Thorson 2025). To access RSR coefficients the model must be fitted with control = sdmTMBcontrol(get_rsr = TRUE). "dispersion" returns dispersion-model coefficients when dispformula is used.
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 data frame with a model column).
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) only when dispersion is scalar. With dispformula, use effects = "dispersion" to extract dispersion-model coefficients.

Time-varying coefficients (from the time_varying argument in sdmTMB()) are returned with effects = "ran_vals", one row per coefficient and time slice, with terms named "<coefficient>:<time>" (e.g., "(Intercept):2011"). These are on the link scale. Their standard deviation (sigma_V) and, for time_varying_type = "ar1", correlation (rho_time) are returned with effects = "ran_pars".

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

References

Restricted Spatial Regression (effects = "rsr"):

Diaz, R.R., and Thorson, J.T. 2025. When and How to Use Restricted Spatial Regression to Separate Environmental Effects from Spatial Confounding. EcoEvoRxiv. doi:10.32942/X28351.

Hanks, E.M., Schliep, E.M., Hooten, M.B., and Hoeting, J.A. 2015. Restricted spatial regression in practice: geostatistical models, confounding, and robustness under model misspecification. Environmetrics 26(4): 243–254. doi:10.1002/env.2331.

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")

# time-varying coefficients:
fit <- sdmTMB(density ~ 0, time_varying = ~ 1, time = "year",
  data = pcod_2011, mesh = pcod_mesh_2011, family = tweedie(),
  spatiotemporal = "off"
)
tidy(fit, "ran_vals", conf.int = TRUE)
tidy(fit, "ran_pars")
```

---

update.sdmTMB *Update an sdmTMB model*

Description

This method updates an sdmTMB model with new arguments while 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.

Details

Unless `control` is supplied, the updated model reuses the fitted model's `sdmTMBcontrol()` settings, including its backend, regardless of the current `sdmTMB.backend` option. A supplied `control` replaces these settings, including the backend (e.g., `update(fit, control = sdmTMBcontrol(backend = "rtmb"))`).

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()`.
`...`	Arguments passed to `visreg::visreg()` or `visreg::visreg2d()`.
`model`	First or second delta-model component.

Details

Note that the residuals are currently randomized quantile residuals, *not* the deviance residuals usually used for GLMs with **visreg**.

Value

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

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)
  visreg_delta(fit_dg, xvar = "depth_scaled", model = 2)
  visreg_delta(fit_dg,
    xvar = "depth_scaled", model = 1,
    scale = "response"
  )
  visreg_delta(fit_dg,
    xvar = "depth_scaled", model = 2,
    scale = "response"
  )
  visreg_delta(fit_dg,
    xvar = "depth_scaled", model = 2,
    scale = "response", rug = FALSE
  )
  visreg2d_delta(fit_dg,
    xvar = "depth_scaled", yvar = "year",
    model = 2, scale = "response"
  )
  v2d <- visreg2d_delta(fit_dg,
    xvar = "depth_scaled", yvar = "year",
    model = 1, scale = "response", plot = FALSE
  )
  if (!is.null(utils::getS3method("persp", "visreg2d", optional = TRUE))) {
    persp(v2d)
  }
  visreg2d_delta(fit_dg,
    xvar = "depth_scaled", yvar = "year",
```

```
    model = 2, scale = "response"
  )

}
```

Index

∗ **datasets**
    ohio_df, 41
    pcod, 42

add_utm_columns, 4
add_utm_columns(), *36*
AIC(), *5*, *8*

Beta *(*Families*)*, 15
Beta(), *16*
betabinomial *(*Families*)*, 15
betabinomial(), *16*, *24*
binomial(), *7*, *16*, *24*, *66*

cAIC, 5
cAIC(), *8*
censored_betabinomial *(*Families*)*, 15
censored_betabinomial(), *16*, *21*, *24*, *68*, *77*
censored_binomial *(*Families*)*, 15
censored_binomial(), *16*, *21*, *24*, *68*
censored_nbinom1 *(*Families*)*, 15
censored_nbinom1(), *16*
censored_nbinom2 *(*Families*)*, 15
censored_nbinom2(), *16*
censored_poisson *(*Families*)*, 15
censored_poisson(), *16*, *21*, *70*
coef.sdmTMB, 6
compare_deviance, 7
compare_deviance(), *14*
cv_to_waywiser, 9
cv_to_waywiser(), *85*

delta_beta *(*Families*)*, 15
delta_beta(), *16*
delta_gamma *(*Families*)*, 15
delta_gamma(), *16*, *66*, *90*
delta_gamma_mix *(*Families*)*, 15
delta_gamma_mix(), *16*
delta_gengamma *(*Families*)*, 15
delta_gengamma(), *16*
delta_lognormal *(*Families*)*, 15
delta_lognormal(), *16*
delta_lognormal_mix *(*Families*)*, 15
delta_lognormal_mix(), *16*
delta_truncated_nbinom1 *(*Families*)*, 15
delta_truncated_nbinom1(), *16*
delta_truncated_nbinom2 *(*Families*)*, 15
delta_truncated_nbinom2(), *16*
DHARMa::createDHARMa(), *10*, *11*
DHARMa::plotQQunif(), *11*
dharma_residuals, 10
dharma_residuals(), *62*
diffusion *(*nonlocal_terms*)*, 40
diffusion(), *67*, *97*
dogfish *(*pcod*)*, 42

Effect.sdmTMB, 13
effects::effect(), *13*
elpd.sdmTMB_cv, 13
elpd.sdmTMB_cv(), *84*, *87*
emmeans.sdmTMB, 14

Families, 15, *66*, *68*, *70*
fmesher::fm_mesh_2d_inla(), *37*
fmesher::fm_nonconvex_hull(), *37*
fmesher::fm_rcdt_2d_inla(), *37*
future.apply::future_lapply(), *86*
future::plan(), *85*

Gamma(), *7*, *16*, *66*
gamma_cv *(*sdmTMBpriors*)*, 79
gamma_mix *(*Families*)*, 15
gamma_mix(), *16*
gather_sims *(*spread_sims*)*, 98
gaussian(), *7*, *16*, *66*
gengamma *(*Families*)*, 15
gengamma(), *7*, *16*
get_censored_upper, 20
get_cog *(*get_index*)*, 22

get_cog(), *52*, *66*, *68*
get_crs *(*add_utm_columns*)*, 4
get_crs(), *4*, *9*, *10*
get_eao *(*get_index*)*, 22
get_eao(), *68*
get_index, 22
get_index(), *27*, *28*, *51*, *52*, *66*, *68*, *70*
get_index_sims, 27
get_index_sims(), *26*, *78*
get_index_split *(*get_index*)*, 22
get_pars, 29
get_prior_densities
*(*get_prior_parameters*)*, 29
get_prior_densities(), *82*
get_prior_parameters, 29
get_prior_parameters(), *82*
get_range_edge, 31
get_weighted_average *(*get_index*)*, 22
get_weighted_average(), *68*
ggeffects::ggpredict(), *90*
glmmTMB::simulate.glmmTMB(), *92*
graphics::image(), *48*
graphics::plot(), *37*

halfnormal *(*sdmTMBpriors*)*, 79
hbll_s_grid *(*pcod*)*, 42

lme4::simulate.merMod(), *92*
lognormal *(*Families*)*, 15
lognormal(), *7*, *16*
lognormal_mix *(*Families*)*, 15
lognormal_mix(), *16*
lognormal_prior *(*sdmTMBpriors*)*, 79
loo::elpd(), *14*, *84*, *87*
loo::loo_compare(), *13*, *14*, *84*, *87*

make_areal_domain, 33
make_areal_domain(), *65*, *70*, *96*
make_category_svc, 34
make_mesh, 36
make_mesh(), *37*, *65*, *85*, *96*
make_zero_one_map, 38
mgcv::gam(), *67*
mgcv::gamm(), *67*
mgcv::s(), *69*
mgcv::smooth2random(), *69*
mgcv::t2(), *69*
mvnormal *(*sdmTMBpriors*)*, 79

nbinom1 *(*Families*)*, 15
nbinom1(), *16*
nbinom2 *(*Families*)*, 15
nbinom2(), *16*, *66*
nbinom2_mix *(*Families*)*, 15
nbinom2_mix(), *16*
nonlocal_terms, 40
normal *(*sdmTMBpriors*)*, 79

ohio_df, 41
ohio_sf *(*ohio_df*)*, 41
ordbeta *(*Families*)*, 15
ordbeta(), *16*

pc_matern *(*sdmTMBpriors*)*, 79
pc_matern(), *48*, *82*
pcod, 42
pcod_2011 *(*pcod*)*, 42
pcod_mesh_2011 *(*pcod*)*, 42
plot.sdmTMBmesh *(*make_mesh*)*, 36
plot_anisotropy, 43
plot_anisotropy(), *66*
plot_anisotropy2 *(*plot_anisotropy*)*, 43
plot_nonlocal_covariate, 44
plot_nonlocal_covariate(), *40*
plot_nonlocal_kernel
*(*plot_nonlocal_covariate*)*, 44
plot_pc_matern, 47
plot_pc_matern(), *83*
plot_smooth, 48
poisson(), *7*, *16*, *66*
predict.sdmTMB, 49
predict.sdmTMB(), *23*, *24*, *27*, *31*, *56*, *68*, *78*, *86*, *91*, *93*
project, 55
project(), *51*

qcs_grid *(*pcod*)*, 42

replicate_df, 59
replicate_df(), *23*
residuals.sdmTMB, 59
residuals.sdmTMB(), *11*, *12*, *86*
RTMB::MakeADFun(), *71*
RTMB::sdreport(), *24*, *70*
run_extra_optimization, 62

sanity, 63
sdmTMB, 64
sdmTMB(), *5*, *7*, *10*, *11*, *16*, *20*, *22*, *23*, *28–30*, *32*, *33*, *35*, *38–40*, *43*, *44*, *49–52*, *56*,

60, *63*, *69*, *76*, *79*, *81*, *85, 86*, *90*, *93*, *96–98*, *100*, *102*
sdmTMB_cv, 84
sdmTMB_cv(), *8–10*, *13, 14*, *83*, *89*
sdmTMB_simulate *(*simulate_new*)*, 94
sdmTMB_simulate(), *40*
sdmTMB_stacking, 89
sdmTMBcontrol, 76
sdmTMBcontrol(), *18*, *35*, *38, 39*, *68*, *71*, *85*, *96*, *102*
sdmTMBpriors, 79
sdmTMBpriors(), *29, 30*, *68*, *70*
set_delta_model, 90
sf::sf(), *9, 10*
sf::st_as_sf(), *4*
sf::st_coordinates(), *4*
sf::st_crs(), *9*
sf::st_transform(), *4*
sigma.sdmTMB, 91
simulate.sdmTMB, 92
simulate.sdmTMB(), *10–12*, *62*, *83*, *86*, *94*, *97*
simulate_new, 94
simulate_new(), *93*
spread_sims, 98
stats::AIC(), *83*
stats::deviance(), *8*
stats::dlnorm(), *82*
stats::glm(), *65*, *69*
stats::kmeans(), *36*
stats::logLik(), *83*
stats::nlminb(), *63*, *70*, *76–79*
stats::optimHess(), *63*, *77*
student *(*Families*)*, 15
student(), *16*

tidy(), *85*
tidy.sdmTMB, 99
tidy.sdmTMB_cv *(*tidy.sdmTMB*)*, 99
time_lag *(*nonlocal_terms*)*, 40
time_lag(), *67*, *97*
TMB::MakeADFun(), *71*, *77, 78*
TMB::normalize(), *77*
TMB::sdreport(), *24*, *70*, *77, 78*
truncated_nbinom1 *(*Families*)*, 15
truncated_nbinom1(), *16*
truncated_nbinom2 *(*Families*)*, 15
truncated_nbinom2(), *16*
try(), *64*
tryCatch(), *64*

tweedie *(*Families*)*, 15
tweedie(), *7*, *16*, *66*

update.sdmTMB, 101
update.sdmTMB(), *78*

visreg2d_delta *(*visreg_delta*)*, 102
visreg::visreg(), *102*
visreg::visreg2d(), *102*
visreg_delta, 102
visreg_delta(), *48*

waywiser::ww_multi_scale(), *9*
wcvi_grid *(*pcod*)*, 42

yelloweye *(*pcod*)*, 42