Package 'sdmTMB'

January 28, 2023

Type Package

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

Version 0.3.0

Description Implements spatial and spatiotemporal predictive-process GLMMs (Generalized Linear Mixed Effect Models) using 'TMB', 'INLA', and the SPDE (Stochastic Partial Differential Equation) approximation to Gaussian random fields. One common application is for spatially explicit (and optionally dynamic) species distribution models (SDMs). See Anderson et al. (2022) <doi:10.1101/2022.03.24.485545>.

License GPL-3

Copyright inst/COPYRIGHTS

URL https://pbs-assess.github.io/sdmTMB/index.html, https://pbs-assess.github.io/sdmTMB/

BugReports https://github.com/pbs-assess/sdmTMB/issues

Depends R (>= 3.5.0)

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

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

LinkingTo RcppEigen, TMB

VignetteBuilder knitr

Additional_repositories https://inla.r-inla-download.org/R/stable

ByteCompile true

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

SystemRequirements GNU make, C++17

NeedsCompilation yes

Author Sean C. Anderson [aut, cre] (<https://orcid.org/0000-0001-9563-1937>),
Eric J. Ward [aut] (<https://orcid.org/0000-0002-4359-0296>),
Lewis A. K. Barnett [aut] (<https://orcid.org/0000-0002-9381-8375>),
Philina A. English [aut] (<https://orcid.org/0000-0003-2992-6782>),
James T. Thorson [ctb, cph] (<https://orcid.org/0000-0001-7415-1010>, VAST author),
Joe Watson [ctb] (Censored Poisson),
Julia Indivero [ctb] (Vignette writing),
Cole C. Monnahan [ctb, cph] (<https://orcid.org/0000-0003-0871-6700>, VAST contributor),
Mollie Brooks [ctb, cph] (<https://orcid.org/0000-0001-6963-8326>, glmmTMB author),
Ben Bolker [ctb, cph] (<https://orcid.org/0000-0002-2127-0443>, glmmTMB author),
Kasper Kristensen [ctb, cph] (TMB/glmmTMB author),
Martin Maechler [ctb, cph] (<https://orcid.org/0000-0002-8685-9910>, glmmTMB author),
Arni Magnusson [ctb, cph] (<https://orcid.org/0000-0003-2769-6741>, glmmTMB author),
Hans J. Skaug [ctb, cph] (glmmTMB author, SPDE barrier),
Anders Nielsen [ctb, cph] (<https://orcid.org/0000-0001-9683-9262>, glmmTMB author),
Casper Berg [ctb, cph] (<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 2023-01-28 07:30:02 UTC

R topics documented:

`add_barrier_mesh` *Transform a mesh object into a mesh with correlation barriers*

Description

Transform a mesh object into a mesh with correlation barriers

Usage

```
add_barrier_mesh(
  spde_obj,
  barrier_sf,
  range_fraction = 0.2,
  proj_scaling = 1,
  plot = FALSE
)
```

Arguments

spde_obj	Output from make_mesh().
barrier_sf	An sf object with polygons defining the barriers. For example, a coastline dataset for ocean data. **Note that this object must have the same projection as the data used to generate the x and y columns in** spde_obj**.**
range_fraction	The fraction of the spatial range that barrier triangles have.
proj_scaling	If spde_obj was created with scaling of the coordinates after the projection (e.g., dividing UTMs by 1000 so the spatial range is on a reasonable scale) the x and y values in spde_obj are multiplied by this scaling factor before applying the projection from barrier_sf.
plot	Logical.

Value

A list similar to make_mesh() but with spde_barrier and a couple other helper list elements added.

If plot = TRUE, then a basic plot will be created as a side effect. Each grey dot represents the center of a "normal" mesh triangle. Each red cross represents the center of a "barrier" mesh triangle.

References

Bakka, H., Vanhatalo, J., Illian, J., Simpson, D., and Rue, H. 2019. Non-stationary Gaussian models with physical barriers. https://arxiv.org/abs/1608.03787

https://sites.google.com/a/r-inla.org/www/barrier-model

https://haakonbakkagit.github.io/btopic107.html

Examples

```
if (require("sf", quietly = TRUE) &&
  require("ggplot2", quietly = TRUE) &&
  require("dplyr", quietly = TRUE) &&
  require("INLA", quietly = TRUE)) {

# First, download coastline data for our region.
# We will use `bc_coast` from the package data,
# but you can recreate it with the following.

# For applied situations on finer scales, you may with to use scale = "large".
# For that, first: remotes::install_github("ropensci/rnaturalearthhires")
# map_data <- rnaturalearth::ne_countries(
#   scale = "medium",
#   returnclass = "sf", country = "canada")
#
# # Crop the polygon for plotting and efficiency:
# st_bbox(map_data)
# bc_coast <- suppressWarnings(suppressMessages(
#   st_crop(map_data,
#     c(xmin = -134, ymin = 46, xmax = -120, ymax = 57))))
```

```
crs_utm9 <- 3156 # Pick a projection, here UTM9

st_crs(bc_coast) <- 4326 # 'WGS84'; necessary on some installs
bc_coast <- st_transform(bc_coast, crs_utm9)

# Project our survey data coordinates:
survey <- pcod %>% select(lon, lat, density) %>%
  st_as_sf(crs = 4326, coords = c("lon", "lat")) %>%
  st_transform(crs_utm9)

# Plot our coast and survey data:
ggplot(bc_coast) +
  geom_sf() +
  geom_sf(data = survey, size = 0.5)

# Note that a barrier mesh won't do much here for this
# example data set, but we nonetheless use it as an example.

# Prepare for making the mesh
# First, we will extract the coordinates:
surv_utm_coords <- st_coordinates(survey)

# Then we will scale coordinates to km so the range parameter
# is on a reasonable scale for estimation:
pcod$X1000 <- surv_utm_coords[,1] / 1000
pcod$Y1000 <- surv_utm_coords[,2] / 1000

spde <- make_mesh(pcod, xy_cols = c("X1000", "Y1000"),
  n_knots = 200, type = "kmeans")
plot(spde)

# Add on the barrier mesh component:
bspde <- add_barrier_mesh(
  spde, bc_coast, range_fraction = 0.1,
  proj_scaling = 1000, plot = TRUE
)

# In the above, the grey dots are the centre of triangles that are in the
# ocean. The red crosses are centres of triangles that are over land. The
# spatial range will be assumed to be 0.1 (`range_fraction`) over land compared
# to over water.

# We can make a more advanced plot if we want:
mesh_df_water <- bspde$mesh_sf[bspde$normal_triangles, ]
mesh_df_land <- bspde$mesh_sf[bspde$barrier_triangles, ]
ggplot(bc_coast) +
  geom_sf() +
  geom_sf(data = mesh_df_water, size = 1, colour = "blue") +
  geom_sf(data = mesh_df_land, size = 1, colour = "green")

# Now, when we fit our model with the new mesh, it will automatically
# include a barrier structure in the spatial correlation:
fit <- sdmTMB(density ~ s(depth, k = 3), data = pcod, mesh = bspde,
```

```
  family = tweedie(link = "log"))
fit
}
```

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

Description

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

Usage

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

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

Arguments

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

Details

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

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

Value

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

Examples

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

---

bc_coast *BC coastline data from ropensci/rnaturalearthhires*

Description

BC coastline data from ropensci/rnaturalearthhires

Usage

```
bc_coast
```

Format

An sf data frame.

Effect.sdmTMB *Calculate effects*

Description

Used by effects package

Usage

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

Arguments

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

Value

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

Examples

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

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

Description

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

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)
```

`Families` *Additional families*

Description

Additional families compatible with `sdmTMB()`.

Usage

```
Beta(link = "logit")

lognormal(link = "log")

gamma_mix(link = "log")

lognormal_mix(link = "log")

nbinom2(link = "log")

nbinom1(link = "log")

truncated_nbinom2(link = "log")

truncated_nbinom1(link = "log")

student(link = "identity", df = 3)

tweedie(link = "log")

censored_poisson(link = "log")

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

delta_gamma_mix(link1 = "logit", link2 = "log")
```

```
delta_lognormal(link1 = "logit", link2 = "log")

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

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

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

delta_poisson_link_gamma(link1 = "log", link2 = "log")

delta_poisson_link_lognormal(link1 = "log", link2 = "log")

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

Arguments

link	Link.
df	Student-t degrees of freedom fixed value parameter.
link1	Link for first part of delta/hurdle model.
link2	Link for second part of delta/hurdle model.

Details

The families ending in `_mix()` are 2-component mixtures where each distribution has its own mean but a shared scale parameter.

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).

For `student()`, the degrees of freedom parameter is currently not estimated and is fixed at `df`.

`delta_poisson_link_gamma()` is the Poisson-link (complementary log-log) delta model (Thorson 2018).

`delta_poisson_link_lognormal()` is the Poisson-link (complementary log-log) delta model (Thorson 2018).

Value

A list with elements common to standard R family objects including `family`, `link`, `linkfun`, and `linkinv`.

References

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

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

```
Beta(link = "logit")
lognormal(link = "log")
gamma_mix(link = "log")
lognormal_mix(link = "log")
nbinom2(link = "log")
nbinom1(link = "log")
truncated_nbinom2(link = "log")
truncated_nbinom1(link = "log")
student(link = "identity")
tweedie(link = "log")
censored_poisson(link = "log")
delta_gamma()
delta_gamma_mix()
delta_lognormal()
delta_lognormal_mix()
delta_truncated_nbinom2()
delta_truncated_nbinom1()
delta_poisson_link_gamma()
delta_poisson_link_lognormal()
delta_beta()
```

---

get_index *Extract a relative biomass/abundance index or a center of gravity*

Description

Extract a relative biomass/abundance index or a center of gravity

Usage

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

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

Arguments

obj	Output from predict.sdmTMB() with return_tmb_object = TRUE.
bias_correct	Should bias correction be implemented TMB::sdreport()?
level	The confidence level.
area	Grid cell area. A vector of length newdata from predict.sdmTMB() or a value of length 1, which will be repeated internally to match.
silent	Silent?
...	Passed to TMB::sdreport().
format	Long or wide.

Value

For get_index(): A data frame with a columns for time, estimate, lower and upper confidence intervals, log estimate, and standard error of the log estimate.

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

References

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

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

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

Geostatistical model-based centre of gravity:

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

Bias correction:

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

See Also

get_index_sims()

Examples

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

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

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

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

if (require("ggplot2", quietly = TRUE)) {
ggplot(ind, aes(year, est)) + geom_line() +
  geom_ribbon(aes(ymin = lwr, ymax = upr), alpha = 0.4)
}

cog <- get_cog(predictions)
cog
```

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

Description

[Experimental]

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

Usage

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

Arguments

obj	predict.sdmTMB() output with nsim > 0.
level	The confidence level.
return_sims	Logical. Return simulation draws? The default (FALSE) is a quantile summary of those simulation draws.
area	A vector of grid cell/polyon areas for each year-grid cell (row of data) in obj. Adjust this if cells are not of unit area or not all the same area (e.g., some cells are partially over land/water). Note that the area vector is added as log(area) to the raw values in obj. In other words, the function assumes a log link, which typically makes sense.
est_function	Function to summarize the estimate (the expected value). mean() would be an alternative to median().
area_function	Function to apply area weighting. Assuming a log link, the function(x, area) x + log(area) default makes sense. If in natural space, function(x, area) x * area makes sense.
agg_function	Function to aggregate samples within each time slice. Assuming a log link, the function(x) sum(exp(x)) default makes sense. If in natural space, function(x) sum(x) makes sense.

Details

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

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

Value

A data frame. If return_sims = FALSE:

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

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

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

See Also

get_index()

Examples

```
if (inla_installed()) {

m <- sdmTMB(density ~ 0 + as.factor(year) + depth_scaled + depth_scaled2,
  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)
```

---

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,
  refine = list(min.angle = 21, max.edge = Inf, max.n.strict = -1, max.n = 1000),
  mesh = NULL
)

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

Arguments

data	A data frame.
xy_cols	A character vector of x and y column names contained in data. These should likely be in an equal distance projection. For a helper function to convert to UTMs, see add_utm_columns().
type	Method to create the mesh. Also see mesh argument to supply your own mesh.
cutoff	An optional cutoff if type is "cutoff". "The minimum allowed distance between points in the mesh". See INLA::inla.mesh.create(). Smaller values create meshes with more knots. Points further apart than this value will receive a separate vertex in the mesh before any mesh refinement.
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".
refine	Logical or list to pass to INLA::inla.mesh.create().

mesh	An optional mesh created via INLA instead of using the above convenience options.
x	Output from make_mesh().
...	Passed to graphics::plot().

Value

make_mesh(): A list of class sdmTMBmesh. The element mesh is the output from INLA::inla.mesh.create() and the element spde is the output from INLA::inla.spde2.matern().

plot.sdmTMB(): A plot of the mesh and data points.

Examples

```
mesh <- make_mesh(pcod, c("X", "Y"), cutoff = 30, type = "cutoff")
plot(mesh)


mesh <- make_mesh(pcod, c("X", "Y"), cutoff = 5, type = "cutoff")
plot(mesh)

mesh <- make_mesh(pcod, c("X", "Y"), n_knots = 50, type = "cutoff_search")
plot(mesh)

mesh <- make_mesh(pcod, c("X", "Y"), n_knots = 50, type = "kmeans")
plot(mesh)

# Defining a mesh directly with INLA:
bnd <- INLA::inla.nonconvex.hull(cbind(pcod$X, pcod$Y), convex = -0.05)
inla_mesh <- INLA::inla.mesh.2d(
  boundary = bnd,
  max.edge = c(20, 50),
  offset = -0.05,
  cutoff = c(2, 5),
  min.angle = 10
)
mesh <- make_mesh(pcod, c("X", "Y"), mesh = inla_mesh)
plot(mesh)
```

---

pcod *Example data for Pacific Cod*

Description

Example data for Pacific Cod

Usage

```
pcod
```

Format

A data frame.

---

pcod_2011 *Example data for Pacific Cod (years 2011 and after)*

Description

Example data for Pacific Cod (years 2011 and after)

Usage

```
pcod_2011
```

Format

A data frame.

---

pcod_mesh_2011 *Example SPDE mesh for the Pacific Cod data (years 2011 and after)*

Description

Example SPDE mesh for the Pacific Cod data (years 2011 and after)

Usage

```
pcod_mesh_2011
```

Format

A list object of class `sdmTMBmesh`.

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**.

`plot_anisotropy2()`: A plot of eigenvectors illustrating the estimated anisotropy. A list of the plotted data is invisibly returned. Uses base graphics.

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,
  time = "year",
  anisotropy = TRUE #<
```

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

`plot_pc_matern` *Plot PC Matérn priors*

Description

Plot PC Matérn priors

Usage

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

Arguments

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

Value

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

See Also

`pc_matern()`

Examples

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

---

plot_smooth *Plot a smooth term from an sdmTMB model*

Description

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

Usage

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

Arguments

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

Details

Note:

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

Value

A plot of a smoother term.

Examples

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

---

predict.sdmTMB *Predict from an sdmTMB model*

Description

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

Usage

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

Arguments

object	A model fitted with sdmTMB().
newdata	A data frame to make predictions on. This should be a data frame with the same predictor columns as in the fitted data and a time column (if this is a spatiotemporal model) with the same name as in the fitted data.
type	Should the est column be in link (default) or response space?
se_fit	Should standard errors on predictions at the new locations given by newdata be calculated? Warning: the current implementation can be slow for large data sets or high-resolution projections unless re_form = NA (omitting random fields). A faster option to approximate point-wise uncertainty may be to use the nsim argument.
re_form	NULL to specify including all spatial/spatiotemporal random effects in predictions. ~0 or NA for population-level predictions. Likely to be used in conjunction with se_fit = TRUE. This does not affect get_index() calculations.
re_form_iid	NULL to specify including all random intercepts in the predictions. ~0 or NA for population-level predictions. No other options (e.g., some but not all random intercepts) are implemented yet. Only affects predictions with newdata. This *does* affects get_index().
nsim	Experimental: If > 0, simulate from the joint precision matrix with nsim draws. Returns a matrix of nrow(data) by nsim representing the estimates of the linear predictor (i.e., in link space). Can be useful for deriving uncertainty on predictions (e.g., apply(x, 1, sd)) or propagating uncertainty. This is currently the fastest way to characterize uncertainty on predictions in space with sdmTMB.
sims_var	Experimental: Which TMB reported variable from the model should be extracted from the joint precision matrix simulation draws? Defaults to the link-space predictions. Options include: "omega_s", "zeta_s", "epsilon_st", and "est_rf" (as described below). Other options will be passed verbatim.
model	Type of prediction if a delta/hurdle model *and* nsim > 0 or mcmc_samples is supplied: NA returns the combined prediction from both components on the link scale for the positive component; 1 or 2 return the first or second model component only on the link or response scale depending on the argument type.
offset	A numeric vector of optional offset values. If left at default NULL, the offset is implicitly left at 0.
tmbstan_model	Deprecated. See mcmc_samples.
mcmc_samples	See extract_mcmc() in the sdmTMBextra package for more details and the Bayesian vignette. If specified, the predict function will return a matrix of a similar form as if nsim > 0 but representing Bayesian posterior samples from the Stan model.
return_tmb_object	Logical. If TRUE, will include the TMB object in a list format output. Necessary for the get_index() or get_cog() functions.
return_tmb_report	Logical: return the output from the TMB report? For regular prediction this is all the reported variables at the MLE parameter values. For nsim > 0 or when mcmc_samples is supplied, this is a list where each element is a sample and the contents of each element is the output of the report for that sample.

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

sims
: **Deprecated**. Please use nsim instead.

area
: **Deprecated**. Please use area in get_index().

...
: Not implemented.

Value

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

A data frame:

- est: Estimate in link space (everything is in link space)
- est_non_rf: Estimate from everything that isn't a random field
- est_rf: Estimate from all random fields combined
- omega_s: Spatial (intercept) random field that is constant through time
- zeta_s: Spatial slope random field
- epsilon_st: Spatiotemporal (intercept) random fields, could be off (zero), IID, AR1, or random walk

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

A list:

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

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

If nsim > 0 or mcmc_samples is not NULL:

A matrix:

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

Examples

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

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

```
predictions <- predict(m)
head(predictions)

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

library(ggplot2)
ggplot(predictions, aes(X, Y, col = resids)) + scale_colour_gradient2() +
  geom_point() + facet_wrap(~year)
hist(predictions$resids)
qqnorm(predictions$resids);abline(a = 0, b = 1)

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

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

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

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

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

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

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

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

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

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

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

```
# Because this is a spatiotemporal model, you'll need at least one time
# element. If time isn't also a fixed effect then it doesn't matter what you pick:
nd$year <- 2011L # L: integer to match original data
p <- predict(m, newdata = nd, se_fit = TRUE, re_form = NA)
ggplot(p, aes(depth_scaled, exp(est),
  ymin = exp(est - 1.96 * est_se), ymax = exp(est + 1.96 * est_se))) +
  geom_line() + geom_ribbon(alpha = 0.4)

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

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

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

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

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

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

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

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

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

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

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

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

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

`qcs_grid` *Example 2x2km prediction grid for Queen Charlotte Sound*

Description

Example 2x2km prediction grid for Queen Charlotte Sound

Usage

```
qcs_grid
```

Format

A data frame.

replicate_df *Replicate a prediction data frame over time*

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

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

residuals.sdmTMB *Residuals method for sdmTMB models*

Description

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

Usage

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

Arguments

object	An sdmTMB() model
type	Type of residual. See details.
model	Which delta/hurdle model component?
mcmc_samples	A vector of MCMC samples of the linear predictor in link space. See the sdmTMBextra package.
...	Passed to residual function. Only n works for binomial.

Details

Types of residuals currently supported:

"mle-laplace" refers to randomized quantile residuals (Dunn & Smyth 1996), which are also known as probability integral transform (PIT) residuals (Smith 1985). Under model assumptions, these should be distributed as standard normal with the following caveat: the Laplace approximation used for the latent/random effects can cause these residuals to deviate from the standard normal assumption even if the model is consistent with the data (Thygesen et al. 2017). Therefore, **these residuals are fast to calculate but can be unreliable.**

"mle-mcmc" refers to randomized quantile residuals where the fixed effects are fixed at their MLE (maximum likelihood estimate) values and the random effects are sampled with MCMC via tmbstan/Stan. As proposed in Thygesen et al. (2017) and used in Rufener et al. (2021). Under model assumptions, these should be distributed as standard normal. **These residuals are theoretically preferred over the regular Laplace approximated randomized-quantile residuals, but will be considerably slower to calculate.**

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. MCMC samples are defined by mcmc_iter - mcmc_warmup. The Stan model can be printed with print_stan_model = TRUE to check. The defaults may not be sufficient for many models.

"mvn-laplace" is the same as "mle-laplace" except the parameters are based on simulations drawn from the assumed multivariate normal distribution (using the joint precision matrix).

"response" refers to response residuals: observed minus predicted.

Value

A vector of 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.

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

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

Examples

```
if (inla_installed()) {

  mesh <- make_mesh(pcod_2011, c("X", "Y"), cutoff = 10)
  fit <- sdmTMB(
    present ~ as.factor(year) + poly(depth, 3),
    data = pcod_2011, mesh = mesh,
    family = binomial()
  )

  # response residuals will be not be normally distributed unless
  # the family is Gaussian:
  r0 <- residuals(fit, type = "response")
  qqnorm(r0)
  qqline(r0)

  # quick but can have issues because of Laplace approximation:
  r1 <- residuals(fit, type = "mle-laplace")
  qqnorm(r1)
  qqline(r1)

  # see also "mle-mcmc" residuals with the help of the sdmTMBextra package
}
```

`run_extra_optimization` *Run extra optimization on an already fitted object*

Description

[Experimental]

Usage

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

Arguments

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

Value

An updated model fit of class sdmTMB.

Examples

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

---

sanity *Sanity check of an sdmTMB model*

Description

Sanity check of an sdmTMB model

Usage

```
sanity(object, big_sd_log10 = 3, gradient_thresh = 0.001)
```

Arguments

object	Fitted model from sdmTMB().
big_sd_log10	Value to check size of standard errors against. A value of 3 would indicate that standard errors greater than 10^3 should be flagged.
gradient_thresh	Gradient threshold to issue warning.

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 Gaussian random field generalized linear mixed effects model (GLMM) with the TMB (Template Model Builder) R package and the SPDE (stochastic partial differential equation) approach. This can be useful for (dynamic) species distribution models and relative abundance index standardization among many other uses.

Usage

```
sdmTMB(
  formula,
  data,
  mesh,
  time = NULL,
  family = gaussian(link = "identity"),
  spatial = c("on", "off"),
  spatiotemporal = c("iid", "ar1", "rw", "off"),
  share_range = TRUE,
  time_varying = NULL,
```

```
  time_varying_type = c("rw", "ar1"),
  spatial_varying = NULL,
  weights = NULL,
  offset = NULL,
  extra_time = NULL,
  reml = FALSE,
  silent = TRUE,
  anisotropy = FALSE,
  control = sdmTMBcontrol(),
  priors = sdmTMBpriors(),
  knots = NULL,
  bayesian = FALSE,
  previous_fit = NULL,
  do_fit = TRUE,
  do_index = FALSE,
  predict_args = NULL,
  index_args = NULL,
  experimental = NULL
)
```

Arguments

formula	Model formula. IID random intercepts are possible using **lme4** syntax, e.g., `+ (1 | g)` where `g` is a column of class character or factor representing groups. Penalized splines are possible via **mgcv** with `s()`. Optionally a list for delta (hurdle) models. See examples and details below.
data	A data frame.
mesh	An object from `make_mesh()`.
time	An optional time column name (as character). Can be left as `NULL` for a model with only spatial random fields; however, if the data are actually spatiotemporal and you wish to use `get_index()` or `get_cog()` downstream, supply the time argument.
family	The family and link. Supports `gaussian()`, `Gamma()`, `binomial()`, `poisson()`, `Beta()`, `nbinom2()`, `truncated_nbinom2()`, `nbinom1()`, `truncated_nbinom1()`, `censored_poisson()`, `gamma_mix()`, `lognormal_mix()`, `student()`, and `tweedie()`. Supports the delta/hurdle models: `delta_beta()`, `delta_gamma()`, `delta_gamma_mix()`, `delta_lognormal_mix()`, `delta_lognormal()`, and `delta_truncated_nbinom2()`, For binomial family options, see 'Binomial families' in the Details section below.
spatial	Estimate spatial random fields? Options are `'on'` / `'off'` or `TRUE` / `FALSE`. Optionally, a list for delta models, e.g. `list('on', 'off')`.
spatiotemporal	Estimate the spatiotemporal random fields as `'iid'` (independent and identically distributed; default), stationary `'ar1'` (first-order autoregressive), a random walk (`'rw'`), or fixed at 0 `'off'`. Will be set to `'off'` if `time = NULL`. If a delta model, can be a list. E.g., `list('off', 'ar1')`. Note that the spatiotemporal standard deviation represents the marginal steady-state standard deviation

of the process in the case of the AR1. I.e., it is scaled according to the correlation. See the TMB documentation. If the AR1 correlation coefficient (rho) is estimated close to 1, say > 0.99, then you may wish to switch to the random walk `'rw'`. Capitalization is ignored. `TRUE` gets converted to `'iid'` and `FALSE` gets converted to `'off'`.

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

`time_varying` An optional one-sided formula describing covariates that should be modelled as a random walk through time. Be careful not to include covariates (including the intercept) in both the main and time-varying formula since the first time step is estimated independently. I.e., at least one should have `~ 0` or `~ -1`. Structure must currently be shared in delta models.

`time_varying_type`
Type of time-varying process to apply to `time_varying` formula.

`spatial_varying`
An optional one-sided formula of coefficients that should vary in space as random fields. Note that you likely want to include a fixed effect for the same variable to improve interpretability since the random field is assumed to have a mean of 0. If a (scaled) time column is used, it will represent a local-time-trend model. See doi:10.1111/ecog.05176 and the spatial trends vignette. Note this predictor should be centered to have mean zero and have a standard deviation of approximately 1 and should likely also be included as a main effect. Structure must currently be shared in delta models.

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

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

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

`reml` Logical: use REML (restricted maximum likelihood) estimation rather than maximum likelihood? Internally, this adds the fixed effects to the list of random effects to integrate over.

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

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

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

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

knots	Optional named list containing knot values to be used for basis construction of smoothing terms. See `mgcv::gam()` and `mgcv::gamm()`. E.g., `s(x, bs = 'cc', k = 4), knots = list(`
bayesian	Logical indicating if the model will be passed to **tmbstan**. If `TRUE`, Jacobian adjustments are applied to account for parameter transformations when priors are applied.
previous_fit	A previously fitted sdmTMB model to initialize the optimization with. Can greatly speed up fitting. Note that the model must be set up *exactly* the same way. However, the data and `weights` arguments can change, which can be useful for cross-validation.
do_fit	Fit the model (`TRUE`) or return the processed data without fitting (`FALSE`)?
do_index	Do index standardization calculations while fitting? Saves memory and time when working with large datasets or projection grids since the TMB object doesn't have to be rebuilt with `predict.sdmTMB()` and `get_index()`. If `TRUE`, then `predict_args` must have a `newdata` element supplied and `area` can be supplied to `index_args`.
predict_args	A list of arguments to pass to `predict.sdmTMB()` if `do_index = TRUE`.
index_args	A list of arguments to pass to `get_index()` if `do_index = TRUE`. Currently, only `area` is supported. Bias correction can be done when calling `get_index()` on the resulting fitted object.
experimental	A named list for esoteric or in-development options. Here be dragons.

Details

Model description

See the model description vignette or the relevant appendix of the preprint on sdmTMB: doi:10.1101/2022.03.24.485545

Binomial families

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

Smooth terms

Smooth terms can be included following GAMs (generalized additive models) using `+ s(x)`, which implements a smooth from `mgcv::s()`. **sdmTMB** uses penalized smooths, constructed via `mgcv::smooth2random()`. This is a similar approach implemented in **gamm4** and **brms**, among other packages. Within these smooths, the same syntax commonly used in `mgcv::s()` or `mgcv::t2()` can be applied, e.g. 2-dimensional smooths may be constructed with `+ s(x, y)` or `+ t2(x, y)`; smooths can be specific to various factor levels, `+ s(x, by = group)`; the basis function dimensions may be specified, e.g. `+ s(x, k = 4)`; and various types of splines may be constructed such as cyclic splines to model seasonality, `+ s(month, bs = "cc", k = 12)` (perhaps with the `knots` argument also be supplied).

Threshold models

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

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

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

Extra time: forecasting or interpolating

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

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

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()`.

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

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

See the delta-model vignette.

Value

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

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

References

Main reference introducing the package to cite when using sdmTMB:

Anderson, S.C., E.J. Ward, P.A. English, L.A.K. Barnett. 2022. sdmTMB: an R package for fast, flexible, and user-friendly generalized linear mixed effects models with spatial and spatiotemporal random fields. bioRxiv 2022.03.24.485545; doi:10.1101/2022.03.24.485545.

Reference for local trends:

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

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

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

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

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

Application to fish body condition:

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

A number of 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:*

Mollie E. Brooks, Kasper Kristensen, Koen J. van Benthem, Arni Magnusson, Casper W. Berg, Anders Nielsen, Hans J. Skaug, Martin Maechler and Benjamin 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., 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.

Examples

```
library(sdmTMB)

# Build an SPDE mesh with INLA:
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` or `cutoff = 15` might make more sense in applied situations
#   for this dataset
# * or build any mesh in INLA and pass it to the `mesh` argument in `make_mesh()`
# * 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)

# Visualize depth effect: (see ?visreg_delta)
```

```
visreg::visreg(fit, xvar = "depth") # link space; randomized quantile residuals
visreg::visreg(fit, xvar = "depth", scale = "response")
visreg::visreg(fit, xvar = "depth", scale = "response", gg = TRUE, rug = FALSE)

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

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

# 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

# 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 = 0L,
  mgcv = deprecated(),
  quadratic_roots = FALSE,
  start = NULL,
  map_rf = deprecated(),
  map = NULL,
  lower = NULL,
  upper = NULL,
  multiphase = TRUE,
  profile = FALSE,
  get_joint_precision = TRUE,
  parallel = getOption("sdmTMB.cores", 1L),
  ...
)
```

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 with `stats::optimHess()` after running `stats::nlminb()`. Sometimes aids convergence.
`mgcv`	**Deprecated** Parse the formula with `mgcv::gam()`?

quadratic_roots
: Experimental feature for internal use right now; may be moved to a branch. Logical: should quadratic roots be calculated? Note: on the sdmTMB side, the first two coefficients are used to generate the quadratic parameters. This means that if you want to generate a quadratic profile for depth, and depth and depth^2 are part of your formula, you need to make sure these are listed first and that an intercept isn't included. For example, `formula = cpue ~ 0 + depth + depth2 + as.factor(year)`.

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

map_rf
: **Deprecated** use `spatial = 'off', spatiotemporal = 'off'` in `sdmTMB()`.

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

lower
: An optional named list of lower bounds within the optimization. Parameter vectors with the same name (e.g., `b_j` or `ln_kappa` in some cases) can be specified as a numeric vector. E.g. `lower = list(b_j = c(-5, -5))`.

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

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

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

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

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

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

Details

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

```
sdmTMB(..., control = sdmTMBcontrol(newton_loops = 1))
```

Value

A list of control arguments

Examples

```
sdmTMBcontrol()
```

sdmTMBpriors *Prior distributions*

Description

[Experimental]

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

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

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

Usage

```
sdmTMBpriors(
  matern_s = pc_matern(range_gt = NA, sigma_lt = NA),
  matern_st = pc_matern(range_gt = NA, sigma_lt = NA),
  phi = halfnormal(NA, NA),
  ar1_rho = normal(NA, NA),
  tweedie_p = normal(NA, NA),
  b = normal(NA, NA)
)

normal(location = 0, scale = 1)

halfnormal(location = 0, scale = 1)

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

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

Arguments

`matern_s`	A PC (Penalized Complexity) prior (`pc_matern()`) on the spatial random field Matérn parameters.
`matern_st`	Same as `matern_s` but for the spatiotemporal random field. Note that you will likely want to set `share_fields = FALSE` if you choose to set both a spatial and spatiotemporal Matérn PC prior since they both include a prior on the spatial range parameter.
`phi`	A `halfnormal()` prior for the dispersion parameter in the observation distribution.
`ar1_rho`	A `normal()` prior for the AR1 random field parameter. Note the parameter has support `-1 < ar1_rho < 1`.

tweedie_p	A normal() prior for the Tweedie power parameter. Note the parameter has support 1 < tweedie_p < 2 so choose a mean appropriately.
b	normal() priors for the main population-level 'beta' effects.
location	Location parameter(s).
scale	Scale parameter. For normal()/halfnormal(): standard deviation(s). For mvnormal(): variance-covariance matrix.
range_gt	A value one expects the spatial or spatiotemporal range is **g**reater **t**han with 1 - range_prob probability.
sigma_lt	A value one expects the spatial or spatiotemporal marginal standard deviation (sigma_O or sigma_E internally) is **l**ess **t**han with 1 - sigma_prob probability.
range_prob	Probability. See description for range_gt.
sigma_prob	Probability. See description for sigma_lt.

Details

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

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

See https://arxiv.org/abs/1503.00256 for a description of the PC prior for Gaussian random fields. Quoting the discussion (and substituting the argument names in pc_matern()): "In the simulation study we observe good coverage of the equal-tailed 95% credible intervals when the prior satisfies P(sigma > sigma_lt) = 0.05 and P(range < range_gt) = 0.05, where sigma_lt is between 2.5 to 40 times the true marginal standard deviation and range_gt is between 1/10 and 1/2.5 of the true range." Also see INLA::inla.spde2.pcmatern().

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

Value

A named list with values for the specified priors.

References

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

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

See Also

plot_pc_matern()

Examples

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

if (inla_installed()) {

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

Facilitates cross validation with sdmTMB models. Returns log likelihood or expected log predictive density (ELPD) across left-out data. Has an option for leave-future-out cross validation. By default creates folds randomly but folds can be manually assigned.

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,
  spde = deprecated(),
  ...
)
```

Arguments

formula	Model formula.
data	A data frame.
mesh_args	Arguments for make_mesh(). If supplied, the mesh will be reconstructed for each fold.
mesh	Output from make_mesh(). If supplied, the mesh will be constant across folds.
time	The name of the time column. Leave as NULL if this is only spatial data.
k_folds	Number of folds.
fold_ids	Optional vector containing user fold IDs. Can also be a single string, e.g. "fold_id" representing the name of the variable in data. Ignored if lfo is TRUE
lfo	Whether to implement leave-future-out (LFO) cross validation where data are used to predict future folds. time argument in sdmTMB() must be specified. See Details section below.

`lfo_forecast` If `lfo = TRUE`, number of time steps to forecast. Time steps 1, ..., T are used to predict T + `lfo_forecast` and the last forecasted time step is used for validation. See Details section below.

`lfo_validations` If `lfo = TRUE`, number of times to step through the LFOCV process. 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.

`spde` **Depreciated.** Use `mesh` instead.

`...` All other arguments required to run `sdmTMB()` model with the exception of `weights`, which are used to define the folds.

Details

Parallel processing

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

For example:

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

Leave-future-out cross validation (LFOCV)

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

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

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

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

See example below.

Value

A list:

- `data`: Original data plus columns for fold ID, CV predicted value, and CV log likelihood.
- `models`: A list of models; one per fold.
- `fold_loglik`: Sum of left-out log likelihoods per fold.
- `fold_elpd`: Expected log predictive density per fold on left-out data.

- sum_loglik: Sum of fold_loglik across all left-out data.
- elpd: Expected log predictive density across all left-out data.
- pdHess: Logical vector: Hessian was invertible each fold?
- converged: Logical: all pdHess TRUE?
- max_gradients: Max gradient per fold.

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,
  family = tweedie(link = "log"), k_folds = 2
)

m_cv$fold_elpd
m_cv$elpd

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,
  mesh = mesh,
```

```
  lfo = TRUE,
  lfo_forecast = 2,
  lfo_validations = 3,
  family = binomial(),
  spatiotemporal = "off",
  time = "year" # must be specified
)

# See how the LFOCV folds were assigned:
example_data <- m_lfocv$models[[1]]$data
table(example_data$cv_fold, example_data$year)
```

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

Description

`sdmTMB_simulate()` 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.

Usage

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

Arguments

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

Value

A data frame where:

- The 1st column is the time variable (if present).
- The 2nd and 3rd columns are the spatial coordinates.
- omega_s represents the simulated spatial random effects (only if present).

- `zeta_s` represents the simulated spatial varying covariate field (only if present).
- `epsilon_st` represents the simulated spatiotemporal random effects (only if present).
- `eta` is the true value in link space
- `mu` is the true value in inverse link space.
- `observed` represents the simulated process with observation error.
- The remaining columns are the fixed-effect model matrix.

See Also

`simulate.sdmTMB()`

Examples

```
if (inla_installed()) {
  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 <- sdmTMB_simulate(
    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
}
```

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

Description

[Experimental]

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

Usage

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

Arguments

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

Value

A vector of model weights.

References

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

Examples

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

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

```
m_cv_2 <- sdmTMB_cv(
  density ~ 0 + depth_scaled,
  data = pcod_2011, mesh = pcod_mesh_2011,
  family = tweedie(link = "log"), k_folds = 2
)

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

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

`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),
  params = c("mle", "mvn"),
  model = c(NA, 1, 2),
  re_form = NULL,
  mcmc_samples = NULL,
  ...
)
```

Arguments

`object`	sdmTMB model
`nsim`	Number of response lists to simulate. Defaults to 1.
`seed`	Random number seed

params	Whether the parameters used in the simulation should come from the Maximum Likelihood Estimate ("mle") or from new draws from the joint precision matrix assuming they are multivariate normal distributed ("mvn").
model	If a delta/hurdle model, which model to simulate from? NA = combined, 1 = first model, 2 = second mdoel.
re_form	NULL to specify a simulation conditional on fitted random effects (this only simulates observation error). ~0 or NA to simulate new random affects (smoothers, which internally are random effects, will not be simulated as new).
mcmc_samples	An optional matrix of MCMC samples. See extract_mcmc() in the sdmTMBextra package.
...	Extra arguments (not used)

Value

Returns a matrix; number of columns is nsim.

See Also

sdmTMB_simulate()

Examples

```
if (inla_installed()) {

# 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 <- sdmTMB_simulate(
  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 the parameters drawn from the joint precision matrix:
s2 <- simulate(fit, nsim = 1, params = "MVN")
```

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

# simulate with new random fields and new parameter draws:
s3 <- simulate(fit, nsim = 1, params = "MVN", re_form = ~ 0)
}
```

`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, n_sims = deprecated())

gather_sims(object, nsim = 200, n_sims = deprecated())
```

Arguments

`object`	Output from `sdmTMB()`.
`nsim`	The number of simulation draws.
`n_sims`	Deprecated: please use `nsim`.

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 ~ 0 + depth_scaled + depth_scaled2,
  data = pcod_2011, mesh = pcod_mesh_2011, family = tweedie(),
  spatiotemporal = "AR1", time = "year")
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"),
  model = 1,
  conf.int = FALSE,
  conf.level = 0.95,
  exponentiate = FALSE,
  silent = FALSE,
  ...
)
```

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), or "ran_vals" (individual random intercepts, if included; behaves like ranef()).
model	Which model to tidy if a delta model (1 or 2).
conf.int	Include a confidence interval?
conf.level	Confidence level for CI.

exponentiate	Whether to exponentiate the fixed-effect coefficient estimates and confidence intervals.
silent	Omit any messages?
...	Extra arguments (not used).

Details

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

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

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

Value

A data frame

Examples

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

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

---

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

Description

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

Usage

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

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

Arguments

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

Details

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

Value

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

Examples

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

  pcod_2011$fyear <- as.factor(pcod_2011$year)
  fit <- sdmTMB(
    density ~ s(depth_scaled) + fyear,
    data = pcod_2011, mesh = pcod_mesh_2011,
    spatial = "off",
    family = tweedie()
  )
  visreg::visreg(fit, xvar = "depth_scaled")
  visreg::visreg(fit, xvar = "fyear")
  visreg::visreg2d(fit, xvar = "fyear", yvar = "depth_scaled")


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

  # Delta model example:
  fit_dg <- sdmTMB(
    density ~ s(depth_scaled, year, k = 8),
    data = pcod_2011, mesh = pcod_mesh_2011,
    spatial = "off",
    family = delta_gamma()
  )
```

```
  visreg_delta(fit_dg, xvar = "depth_scaled", model = 1, gg = TRUE)
  visreg_delta(fit_dg, xvar = "depth_scaled", model = 2, gg = TRUE)
  visreg_delta(fit_dg,
    xvar = "depth_scaled", model = 1,
    scale = "response", gg = TRUE
  )
  visreg_delta(fit_dg,
    xvar = "depth_scaled", model = 2,
    scale = "response"
  )
  visreg_delta(fit_dg,
    xvar = "depth_scaled", model = 2,
    scale = "response", gg = TRUE, rug = FALSE
  )
  visreg2d_delta(fit_dg,
    xvar = "depth_scaled", yvar = "year",
    model = 2, scale = "response"
  )
  visreg2d_delta(fit_dg,
    xvar = "depth_scaled", yvar = "year",
    model = 1, scale = "response", plot.type = "persp"
  )
  visreg2d_delta(fit_dg,
    xvar = "depth_scaled", yvar = "year",
    model = 2, scale = "response", plot.type = "gg"
  )

}
```

Index