Package 'FINN'

August 9, 2026

Type Package

Title Forest Informed Neural Networks

Version 0.1.0

Maintainer Yannek Käber <y.kaeber@posteo.de>

Description A hybrid dynamic forest (gap) model (FINN) that can be configured as a fully mechanistic, process-based model, like classic forest gap models, or with its demographic processes (growth, mortality, regeneration) replaced by deep neural networks (DNNs), or any combination of the two. Provides functions to define a model and its mechanistic or empirical components, calibrate it to forest inventory data, and interpret the calibrated processes. FINN is implemented with the 'torch' package, which supplies GPU support and the automatic differentiation used to calibrate the model by stochastic gradient descent; no knowledge of 'torch' is required. The hybrid modeling approach is described in Pichler and Käber (2026) <doi:10.1111/2041-210x.70347>.

License GPL (>= 3)

Encoding UTF-8

URL https://github.com/FINNverse/FINN, https://finnverse.github.io/FINN/

BugReports https://github.com/FINNverse/FINN/issues

Depends R (>= 4.1.0)

LinkingTo Rcpp

Imports abind, cli, stats, utils, data.table, coro, Rcpp, torch, ggplot2, glue

Suggests testthat (>= 3.0.0), knitr, rmarkdown

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

Author Yannek Käber [aut, cre] (ORCID:
<https://orcid.org/0000-0002-7041-9849>),
Maximilian Pichler [aut] (ORCID:
<https://orcid.org/0000-0003-2252-8327>)

Repository CRAN

Date/Publication 2026-08-09 08:00:08 UTC

Contents

ALE *Accumulated local effect plots*

Description

Calculates accumulated local effects (ALE) for the three processes

Usage

```
ALE(
  model,
  env = NULL,
  init_cohort = NULL,
  env_autoscale = TRUE,
  sim_seed = 42L,
  plot = TRUE,
  process = NULL,
  scale = FALSE,
  ...
)
```

Arguments

model	(finn_class) Model object of class finn_class.
env	(data.table|data.frame) Environmental covariates for which the ALE should be calculated. If NULL (default) the training env cached by fit() is used.
init_cohort	(CohortMat) Initial cohort of class CohortMat. If NULL (default) the training init_cohort cached by fit() is used (or bare ground if none was cached).
env_autoscale	(logical(1)) If TRUE (default) env is assumed to be on the raw (unscaled) scale and the model's stored env_scaling is applied internally before the effects are computed, mirroring how the model was fitted (see fit()'s env_autoscale). Set FALSE if env is already on the model scale, or for a model fitted without autoscaling.
sim_seed	(integer(1)) Seed applied via FINN.seed() before the state-harvesting simulation, so the (stochastic) conditional effects / ALE are reproducible and cacheable. NULL disables seeding.
plot	(logical(1)) If TRUE (default) an ALE plot is drawn via plot.FINNale() (rows = processes, columns = environmental predictors, one coloured line per species).

process	(character|NULL) If given (one of "growth", "mortality", "regeneration") only that process is plotted. NULL (default) plots all three.
scale	(logical(1)) If TRUE each curve is divided by the SD of its process x species rate, yielding dimensionless, comparable effects (the curve's variance then equals the Sobol-style normalised importance). Default FALSE.
...	Not supported yet.

Value

A list with one table per process (e.g. $growth, $mortality, $regeneration). Each table gives the accumulated local effect (ale) of every driver (var) across its observed range (x), per species. When plot = TRUE the effects are also drawn.

---

apply_env_scaling *Apply stored z-standardization to environmental predictors*

Description

Re-applies the constants learned by compute_env_scaling() to env. The transformation uses the *stored* mean/sd only and never recomputes them from env, so calibration and prediction use an identical transformation (the usual pitfall of scale() inside a model formula is avoided).

Usage

```
apply_env_scaling(env, scaling)
```

Arguments

env	A data.frame/data.table of raw environmental data.
scaling	The data.frame returned by compute_env_scaling(), or NULL (in which case env is returned unchanged). For a model fit with env_autoscale = TRUE this is model$env_scaling.

Value

env with the scaled predictor columns, as a data.table.

See Also

compute_env_scaling, fit

Examples

```
# reproduce the standardization a model fit with env_autoscale = TRUE used:
env <- data.frame(siteID = 1:3, year = 1L, temp = c(4, 6, 8), prec = c(700, 850, 1000))
scaling <- compute_env_scaling(env)
apply_env_scaling(env, scaling)
```

`array2obsDF` *Transform Arrays to Observation Data Table*

Description

This function transforms arrays of species, dbh, and trees back into an observation data table.

Usage

```
array2obsDF(obs_array)
```

Arguments

`obs_array`	A list containing three arrays: species, dbh, and trees.

Value

A data.frame with columns siteID, patchID, cohortID, species, dbh, and trees.

Examples

```
obs_array <- list(
  species = array(c("A", "B"), dim = c(2, 2, 2)),
  dbh = array(c(10, 20, 30, 40), dim = c(2, 2, 2)),
  trees = array(c(100, 200, 150, 250), dim = c(2, 2, 2)))
result <- array2obsDF(obs_array)
```

`averageConditionalEffects`
Average conditional effects of a FINN model

Description

Summarises the conditional effects into a per process x species x variable average marginal effect: the mean local derivative (`mean_effect`, an approximate linear effect). Derived cheaply from the cached conditional effects.

Usage

```
averageConditionalEffects(
  model,
  env = NULL,
  init_cohort = NULL,
  env_autoscale = TRUE,
  sim_seed = 42L,
  env_only = TRUE
)
```

Arguments

model	(finn_class) fitted model.
env	(data.table|data.frame|NULL) env covariates; NULL uses the cached training env.
init_cohort	(CohortMat|NULL) initial cohort; NULL uses the cached training init_cohort.
env_autoscale	(logical(1)) see ALE().
sim_seed	(integer(1)) seed for the state-harvesting simulation (see ALE()).
env_only	(logical(1)) if TRUE (default) report only the environmental predictors of each process; if FALSE also include the stand-state inputs (dbh, light, growth) each process depends on.

Value

a named list (one entry per process) of data.frames with columns species, variable, mean_effect.

BA_stand *Calculate the Basal Area of a Stand*

Description

This function calculates the basal area of a stand based on the diameter at breast height (dbh), the number of trees, and the patch size in hectares.

Usage

```
BA_stand(dbh, trees, patch_size_ha)
```

Arguments

dbh	A torch tensor or numeric vector representing the diameter at breast height of the trees in centimeters.
trees	A torch tensor or numeric vector representing the number of trees.
patch_size_ha	A numeric value representing the size of the patch in hectares.

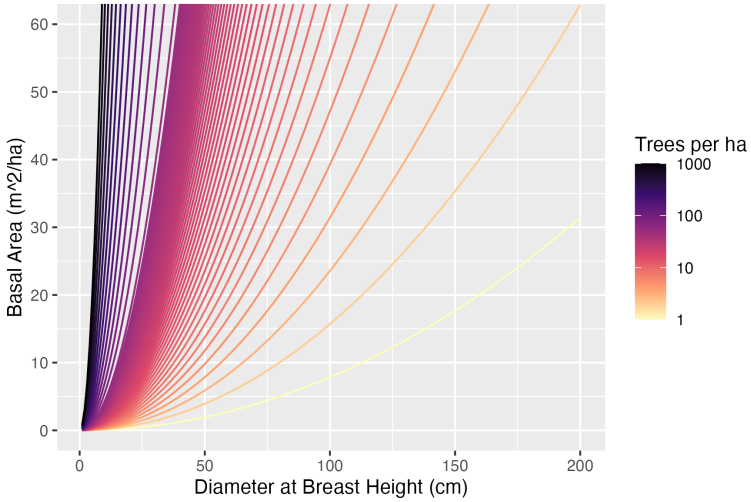
Details

The basal area of a stand is the cross-sectional area of all trees in a stand per unit area. This function calculates the basal area per ha using the formula:

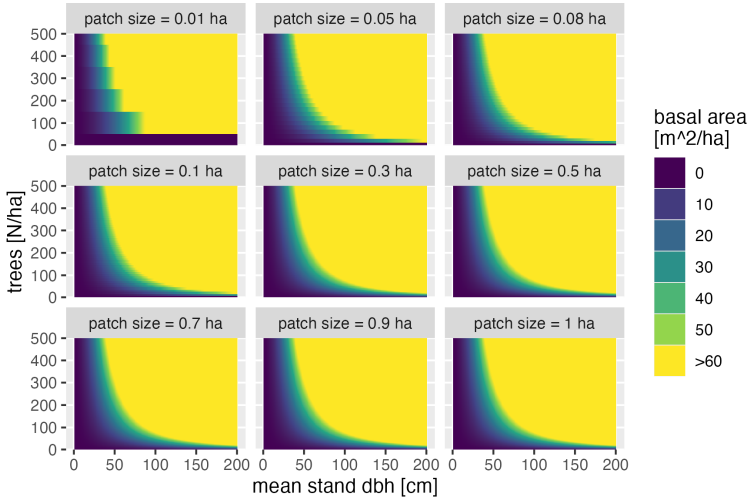
$$BA = \left(\frac{\pi \left(\frac{\text{dbh}}{100} \right)^2}{4} \right) \times \text{trees} \div \text{patch\_size\_ha}$$

The formula takes into account the diameter at breast height (dbh) in centimeters, the number of trees, and the size of the patch in hectares to calculate the basal area in square meters per hectare.

This plot illustrates the basal area for different combinations of dbh and number of trees.



Sensitivity of basal area for different combinations of dbh, number of trees to patch size.



Value

A numeric value representing the basal area of the stand in square meters per hectare.

Examples

```
# BA_stand operates on torch tensors, so this runs only where the torch
# backend (libtorch) is available.
dbh_vec <- seq(1, 200, 1)
trees_vec <- c(0:500, 10^(seq(2, 4, length.out = 20)))

# Generate test data for a patch size of 0.1
patch_size <- 0.1
cohort_df1 <- expand.grid(
  trees_ha = trees_vec,
  patch_size_ha = patch_size,
  dbh = dbh_vec
)

cohort_df1 <- data.frame(
  patchID = 1,
  cohortID = 1,
  species = 1,
  cohort_df1
)

cohort_df1$siteID <- 1:nrow(cohort_df1)
cohort_df1$trees <- round(cohort_df1$trees_ha * patch_size)

cohort <- CohortMat$new(obs_df = cohort_df1)

cohort_df1$basal_area <- torch::as_array(
  BA_stand(cohort$dbh, cohort$trees, patch_size_ha = patch_size))

# View the first few rows of the resulting data frame
head(cohort_df1)
```

`BA_stem` *Calculate the basal area of a tree given the diameter at breast height (dbh)*

Description

This function calculates the basal area of a tree given the diameter at breast height (dbh).

Usage

```
BA_stem(dbh)
```

Arguments

dbh	torch.Tensor The diameter at breast height of the tree.

Value

torch.Tensor The basal area of the tree.

Examples

```
dbh = torch::torch_tensor(50)
basal_area = BA_stem(dbh)
print(basal_area)
```

---

binomial_from_bernoulli *Draw binomial counts from per-trial Bernoulli probabilities*

Description

Draw binomial counts from per-trial Bernoulli probabilities

Usage

```
binomial_from_bernoulli(n, p)
```

Arguments

n	torch.Tensor Number of Bernoulli trials per element.
p	torch.Tensor Success probability per element.

Value

torch.Tensor The number of successes per element.

---

climateDF2array *Convert a climate data frame to a FINN environment array*

Description

Reshapes a long-format climate table into the site x year x variable array FINN uses internally, auto-detecting the time resolution from the columns.

Usage

```
climateDF2array(climate_dt, env_vars)
```

Arguments

climate_dt	A data.frame/data.table of climate values in long format, with siteID, a time column (e.g. year), and one column per variable.
env_vars	character Names of the environmental variable columns to extract.

Value

A numeric array of environmental values indexed by site, time and variable.

CohortMat *Cohort Matrix Class*

Description

Initialize the CohortMat class

Usage

```
CohortMat(
  obs_df = NULL,
  dbh = NULL,
  trees = NULL,
  species = NULL,
  dims = c(50, 30, 10),
  sp = 10,
  device = "cpu"
)
```

Arguments

obs_df	A data frame containing columns "siteID", "patchID", "species", "dbh", and "trees". If provided, it will be used to initialize the tensors.
dbh	A tensor or array representing the diameter at breast height. Defaults to self$dbh.
trees	A tensor or array representing the number of trees. Defaults to self$trees.
species	A tensor or array representing the species. Defaults to self$species.
dims	A numeric vector representing the dimensions of the arrays (sites, patches, cohorts). Defaults to self$dims.
sp	An integer representing the number of species. Defaults to self$sp.
device	A character string specifying the device to use ('cpu' or 'cuda'). Defaults to self$device. @description
	Convert the tensors to a data frame format

Details

An R6 class for managing cohorts of trees in forest models. This class allows for the initialization, transformation, and manipulation of cohorts represented by arrays of dbh, trees, and species.

Value

data.table object

Fields

`dbh` A tensor representing the diameter at breast height for each cohort.

`trees` A tensor representing the number of trees in each cohort.

`species` A tensor representing the species of each cohort.

`dims` A vector representing the dimensions of the arrays (sites, patches, cohorts).

`sp` An integer representing the number of species.

`device` A character string specifying the device to use ('cpu' or 'cuda').

`dbh_r` A numeric array representing the diameter at breast height in R array format.

`trees_r` A numeric array representing the number of trees in R array format.

`species_r` An integer array representing the species in R array format.

`device_r` A character string specifying the device in R format ('cpu' or 'cuda').

`obsDF2arrays` Transforms data.table into array

`competition` *Compute the fraction of available light (light) for each cohort based on the given parameters*

Description

This function calculates the fraction of available light for each cohort of trees based on their diameter at breast height (dbh), species, number of trees, and global parameters.

Usage

```
competition(
  dbh,
  species,
  trees,
  parComp,
  h = NULL,
  patch_size_ha,
  ba = NULL,
  cohortHeights = NULL,
  n_quantiles = 10,
  continuous = FALSE
)
```

Arguments

dbh	torch.Tensor Diameter at breast height for each cohort.
species	torch.Tensor species index for each cohort.
trees	torch.Tensor Number of trees in each cohort.
parComp	torch.Tensor Competition / height-allometry parameters per species.
h	torch.Tensor (Optional) Height of each cohort. Defaults to NULL.
patch_size_ha	numeric Patch size in hectares.
ba	torch.Tensor (Optional) Pre-computed basal area. Defaults to NULL.
cohortHeights	torch.Tensor (Optional) Pre-computed cohort heights. Defaults to NULL.
n_quantiles	integer Number of height quantiles used when continuous = FALSE. Defaults to 10.
continuous	logical Use the continuous competition formulation. Defaults to FALSE.

Value

torch.Tensor Fraction of available light (light) for each cohort.

---

compute_env_scaling *Learn z-standardization for environmental predictors*

Description

Computes the centering and scaling constants (mean and standard deviation) of every numeric environmental predictor in env (all columns except the keys siteID and year), so they can be re-applied unchanged to new data at prediction time. A predictor with (near-)zero standard deviation is given scale = 1 (centred only) to avoid division by zero, mirroring recipes::step_normalize.

Usage

```
compute_env_scaling(env)
```

Arguments

env	A data.frame/data.table with siteID, year and the environmental predictor columns.

Value

A data.frame with columns variable, center, scale; or NULL if there are no numeric predictors. This is the object stored on a fitted model as model$env_scaling when it is fit with env_autoscale = TRUE.

See Also

apply_env_scaling

`conditionalEffects` *Conditional effects of a FINN model*

Description

Computes (and caches on the model) the conditional effects — the local derivatives of each demographic process (growth, mortality, regeneration) with respect to its inputs, per process and per species. This is the shared primitive behind `ALE()`, `averageConditionalEffects()` and the analytical variable importance in `summary.finn_class()`.

Usage

```
conditionalEffects(
  model,
  env = NULL,
  init_cohort = NULL,
  env_autoscale = TRUE,
  sim_seed = 42L
)
```

Arguments

`model`	(`finn_class`) fitted model.
`env`	(`data.table|data.frame|NULL`) env covariates; `NULL` uses the cached training env.
`init_cohort`	(`CohortMat|NULL`) initial cohort; `NULL` uses the cached training init_cohort.
`env_autoscale`	(`logical(1)`) see `ALE()`.
`sim_seed`	(`integer(1)`) seed for the state-harvesting simulation (see `ALE()`).

Value

an object of class `FINNconditionalEffects` (also cached on `model$conditional_effects`).

createHybrid *Define a hybrid (deep-neural-network) demographic process for FINN*

Description

Configures a process (growth, mortality, or regeneration) in which the entire process equation – not just its environmental-response function – is replaced by a deep neural network (DNN), for use as `growth_process`, `mortality_process`, or `regeneration_process` in `finn()`. This is the second ("Level 2") level of hybridization described by Pichler & Käber (2026); to replace only the environmental-response function while keeping the rest of the process mechanistic ("Level 1"), use `createProcess()` instead. `competition_process` must always be created with `createProcess()`, as the competition (light availability) process does not support full replacement by a DNN.

Usage

```
createHybrid(
  formula = NULL,
  optimize = TRUE,
  dispersion_parameter = 1,
  NN = NULL,
  dropout = 0.3,
  encoder_layers = 1L,
  hidden = c(50L, 50L),
  sample_regeneration = TRUE,
  transformer = TRUE,
  emb_dim = 20L,
  dim_feedforward = 256L
)
```

Arguments

`formula` (`formula`)
Environmental predictors passed to the network, evaluated against the `env` data. Default `NULL` uses `~ .` (all available environmental covariates).

`optimize` (`logical(1)`)
Should the network's weights be estimated during `fit()`? Default `TRUE`.

`dispersion_parameter` (`numeric(1)`)
Initial dispersion parameter of the negative binomial distribution used to sample recruits. Only used when this object is the regeneration process.

`NN` (`nn_module`)
Optional custom `torch` module overriding the default network architecture entirely.

`dropout` (`numeric(1)`)
Dropout rate of the network.

`encoder_layers` (integer(1))
: Number of transformer encoder layers. Only used when `transformer = TRUE`.

`hidden` (integer())
: Hidden-layer sizes of the feed-forward network. Only used when `transformer = FALSE`.

`sample_regeneration` (logical(1))
: Should recruits actually be sampled from the negative binomial regeneration distribution (`TRUE`), or only its expectation used (`FALSE`)? Only used when this object is the regeneration process.

`transformer` (logical(1))
: Use a transformer encoder architecture (`TRUE`, default) or a feed-forward network (`FALSE`).

`emb_dim` (integer(1))
: Embedding dimension for the species/cohort/ environment features.

`dim_feedforward` (integer(1))
: Dimension of the feed-forward sublayer within each transformer encoder layer. Only used when `transformer = TRUE`.

Details

The network receives the same cohort- and site-level information that the corresponding mechanistic process equation would use – diameter at breast height, number of trees, available light, species identity and the site's environmental predictors (plus the growth rate, for the mortality process) – and predicts the process output directly. Species identity is passed through a learned embedding layer rather than treated as a categorical covariate with per-species coefficients, so the network can learn low-dimensional "contrasts" between species or plant functional types (PFTs). An inverse-link function appropriate to each process is applied to the network's output: a sigmoid for mortality (a per-tree death probability, as for mortality), and an exponential for growth and regeneration (always-positive rates; for regeneration this is the mean of the negative binomial distribution from which recruits are drawn, as for regeneration).

Two network architectures are available, selected with `transformer`. The default (`transformer = TRUE`) is a small transformer encoder (`encoder_layers` layers, embedding dimension `emb_dim`, feed-forward dimension `dim_feedforward`) that embeds each cohort, its species and the site's environment and attends across the cohorts of a patch. Setting `transformer = FALSE` instead uses a feed-forward network with hidden layers `hidden` (default two layers of 50 units each) and `dropout`, matching the architecture used by Pichler & Käber (2026) for the Barro Colorado Island case study (where dropout was set to 10%).

As with `createProcess()`, hybrid processes are calibrated jointly, end-to-end, with the remaining mechanistic or hybrid processes via `fit()`, rather than pre-trained in isolation and plugged in afterwards. `optimize` controls whether the network's weights are estimated during fitting or kept fixed at their (random) initial values, and `dispersion_parameter`/ `sample_regeneration` have the same meaning as in `createProcess()` and are only used when this object is the regeneration process.

Value

A list of class "hybrid" containing the process definition and associated parameters, to be passed as growth_process, mortality_process, or regeneration_process to finn().

References

Pichler, M., & Käber, Y. (2026). Inferring processes within dynamic forest models using hybrid modelling. *Methods in Ecology and Evolution*. doi:10.1111/2041210x.70347

See Also

createProcess(), finn()

Examples

```
growth_process <- createHybrid(formula = ~temperature + precipitation)
```

---

createProcess *Define a demographic process for FINN*

Description

Configures one demographic process (growth, mortality, regeneration or competition) for use as mortality_process, growth_process, regeneration_process, or competition_process in finn(), either as a fully mechanistic process with an explicit, interpretable functional form, or – if hidden is supplied – as the first ("Level 1") level of hybridization described by Pichler & Käber (2026): only the process' environmental-response function is replaced by a small feed-forward neural network, while the rest of the process equation remains mechanistic. To replace the entire process equation with a neural network ("Level 2"), use createHybrid() instead.

Usage

```
createProcess(
  formula = NULL,
  func,
  initSpecies = NULL,
  initEnv = NULL,
  hidden = NULL,
  optimizeSpecies = FALSE,
  optimizeEnv = TRUE,
  inputNN = NULL,
  outputNN = NULL,
  dispersion_parameter = 1,
  NN = NULL,
  upper = NULL,
  lower = NULL,
```

```
  dropout = 0,
  sample_regeneration = TRUE,
  n_quantiles = 10L,
  continuous = FALSE
)
```

Arguments

formula	(formula) Environmental predictors for this process' environmental-response function, evaluated against the env data. Default NULL uses ~. (all available environmental covariates).
func	(function) Mechanistic process equation, e.g. growth, mortality, regeneration, or competition, or a custom function with the same arguments. Required.
initSpecies	(matrix) Optional custom initial values for the species-specific process parameters. Default NULL draws random initial values.
initEnv	(list) Optional custom initial values for the parameters of the environmental-response network/regression. Default NULL draws random initial values.
hidden	(integer()) Hidden-layer sizes of the feed-forward neural network that replaces the environmental-response function (Level 1 hybrid). Default NULL keeps the environmental response mechanistic (linear/logistic).
optimizeSpecies	(logical(1)) Should the species-specific process parameters be estimated during fit()? Default FALSE.
optimizeEnv	(logical(1)) Should the parameters of the environmental-response function be estimated during fit()? Default TRUE.
inputNN	(integer(1)) Input dimension of the environmental-response network. Default NULL infers it from formula/env.
outputNN	(integer(1)) Output dimension of the environmental-response network. Default NULL uses the number of species.
dispersion_parameter	(numeric(1)) Initial dispersion parameter of the negative binomial distribution used to sample recruits. Only used when this process is the regeneration process.
NN	(nn_module) Optional custom torch module overriding the default environmental-response network architecture.

upper (numeric())
Upper boundaries (natural scale) for the species-specific process parameters.

lower (numeric())
Lower boundaries (natural scale) for the species-specific process parameters.

dropout (numeric(1))
Dropout rate of the environmental-response network. Ignored unless hidden is set.

sample_regeneration
(logical(1))
Should recruits actually be sampled from the negative binomial regeneration distribution (TRUE), or only its expectation used (FALSE)? Only used when this process is the regeneration process.

n_quantiles (integer(1))
Number of height classes used to discretize cohorts when computing shading/light availability. Only used when this process is the competition process and continuous = FALSE.

continuous (logical(1))
Compute shading continuously for every pair of cohorts (TRUE) instead of binning cohorts into n_quantiles height classes (FALSE, default). Only used when this process is the competition process.

Details

Each demographic process in FINN is the product of (i) a process equation func that operates on the cohort state (dbh, number of trees, available light, species) and species-specific process parameters, and (ii) a species- and process-specific environmental-response function that maps site-level environmental predictors to a scalar effect on the process (see finn() for the underlying equations). createProcess() configures both parts:

- func implements the mechanistic process equation itself. The package's default process functions (growth, mortality, regeneration, competition) reproduce the equations described by Pichler & Käber (2026); a custom function with the same arguments can be passed instead to use a different functional form while keeping the process embedded in, and jointly calibrated with, the rest of the model.
- The environmental-response function is, by default (hidden = NULL), a linear/logistic niche function with one coefficient per environmental covariate (named in formula) and per species, comparable to a classic species distribution model. Setting hidden to a vector of hidden-layer sizes (e.g. c(25L)) instead replaces this function with a feed-forward neural network, while func and the species-specific process parameters remain mechanistic – the "Level 1" hybridization described in the paper.

formula selects which columns of the env data (passed to fit() or predict.finn_class()) enter the environmental-response function; initSpecies/initEnv allow supplying custom starting values for the process parameters/environmental-response model instead of the package's random initialization; optimizeSpecies/optimizeEnv control whether these parameters are estimated during fit() or held fixed at their initial values; and upper/lower set box constraints (on the natural process-parameter scale) within which the species-specific process parameters are constrained during optimization.

Value

A list of class "process" containing the process definition and associated parameters, to be passed as mortality_process, growth_process, regeneration_process, or competition_process to finn().

References

Pichler, M., & Käber, Y. (2026). Inferring processes within dynamic forest models using hybrid modelling. *Methods in Ecology and Evolution*. doi:10.1111/2041210x.70347

See Also

createHybrid(), finn()

Examples

```
growth_process <- createProcess(formula = ~temperature + precipitation, func = growth)
```

---

dbh2ba *Convert DBH to basal area*

Description

Computes basal area in m^2 from diameter at breast height in cm:

$$BA = \pi \left(\frac{\text{DBH}}{200} \right)^2$$

Usage

```
dbh2ba(dbh)
```

Arguments

dbh	numeric vector of diameters (cm).

Value

numeric vector of basal areas (m^2).

`feature_importance` *Permutation feature importance for FINN demographic rates*

Description

Permutes each environmental predictor of a process and measures how strongly the permutation shifts that process's predicted rate, **per process and per species**. Unlike the analytical ALE-variance importance, this re-simulates the model, so it captures the full dynamical response (feedback through the stand state) rather than the process response function alone. It does NOT use the conditional-effects cache.

Usage

```
feature_importance(
  model,
  env = NULL,
  init_cohort = NULL,
  nperm = 20L,
  method = c("rmse", "sobol"),
  seed = NULL,
  sim_seed = 42L,
  env_autoscale = TRUE,
  ...
)
```

Arguments

`model`	(`finn_class`) fitted model.
`env`	(`data.table|data.frame|NULL`) env covariates; `NULL` uses the cached training env.
`init_cohort`	(`CohortMat|NULL`) init cohort; `NULL` uses the cached training init_cohort.
`nperm`	(`integer(1)`) number of permutation replicates (default 20).
`method`	(`character(1)`) `"rmse"` or `"sobol"`.
`seed`	(`integer(1)|NULL`) R RNG seed controlling which permutations are drawn.
`sim_seed`	(`integer(1)|NULL`) torch seed for common random numbers across runs.
`env_autoscale`	(`logical(1)`) see `ALE()`. `TRUE` (default) leaves `predict()` to rescale raw `env` internally; `FALSE` treats `env` as already on the model scale.
`...`	passed to `predict()` (e.g. `patches`, `patch_size`).

Details

Two scorings via `method`:

- `"rmse"` — RMSE between the unpermuted and permuted rate, in units of that species' rate SD. Unbounded; larger = more important.
- `"sobol"` — total-effect estimator `0.5 * mean(MSE_shift) / Var(rate)`; dimensionless (only bounded in `[0, 1]` under independent predictors — FINN's climate predictors are usually correlated, so treat it as relative).

Predictors are read per-process from the model formulas, so processes with different formulas get different variable sets. Common random numbers (`sim_seed`, applied to the torch simulation RNG) make the stochastic mortality/regeneration draws shared across the reference and permuted runs, so a driver with no effect returns ~0.

Value

a named list (one per process) of data.frames with columns `species, variable, importance`, sorted within species.

`finn` *Forest Informed Neural Network*

Description

Creates a Forest Informed Neural Network (FINN), a differentiable, cohort-based dynamic forest (gap) model in the tradition of JABOWA/ForClim-style models, in which any of the four demographic processes (growth, mortality, regeneration, competition for light) can either be specified mechanistically or replaced by a deep neural network (DNN). Mechanistic and DNN-based processes are calibrated jointly, end-to-end, via gradient descent (see fit).

Usage

```
finn(
  N_species,
  mortality_process = NULL,
  growth_process = NULL,
  regeneration_process = NULL,
  competition_process = NULL,
  recruits_dbh = 1
)
```

Arguments

`N_species`	`(integer(1))` Number of species.

`mortality_process`
: (`function`)
Mortality process, created by createProcess (mechanistic) or createHybrid (DNN-based). If `NULL` (default), the default mechanistic mortality process is used.

`growth_process` (`function`)
: Growth process, created by createProcess (mechanistic) or createHybrid (DNN-based). If `NULL` (default), the default mechanistic growth process is used.

`regeneration_process`
: (`function`)
Regeneration process, created by createProcess (mechanistic) or createHybrid (DNN-based). If `NULL` (default), the default mechanistic regeneration process is used.

`competition_process`
: (`function`)
Competition (light availability) process, created by createProcess. If `NULL` (default), the default mechanistic competition process is used.

`recruits_dbh` (`numeric(1)`)
: Starting dbh for recruits. Has value 1.0 as default.

Details

FINN represents the forest as cohorts of trees, grouped by site, patch and cohort, each characterized by diameter at breast height (dbh), number of trees, and species identity. Starting from an initial state, FINN simulates the forest forward in discrete annual time steps by sequentially applying the four demographic processes: competition (light availability, based on basal area and species-specific shading), growth (diameter increment as a function of light, size and environment), mortality (binomial death of trees as a function of growth, light, size and environment) and regeneration (recruitment of new cohorts as a function of light and environment, drawn from a negative binomial distribution). Each process additionally depends on a species- and process-specific environmental-response function that maps site-level environmental predictors to a scalar effect on the process.

Every process and its environmental-response function can be configured in one of two ways: (1) mechanistically, with an explicit functional form and interpretable parameters (e.g. light-response thresholds, allometric coefficients), created via createProcess; or (2) as a hybrid process, in which the environmental-response function or the entire process equation is replaced by a DNN, created via createHybrid. The remaining mechanistic processes constrain the DNN to ecologically plausible behaviour, while the DNN absorbs misalignments and structural simplifications that would otherwise bias the mechanistic processes; both are estimated jointly rather than calibrated in isolation and plugged in afterwards. If a process argument is left at its default `NULL`, the corresponding default mechanistic process (as described in Pichler & Käber, 2026, Methods in Ecology and Evolution) is used.

`finn()` only assembles the model architecture (analogous to instantiating a `torch nn_module`); none of the process or environmental-response parameters are estimated yet. Use fit to calibrate the returned object against observed data, and predict.finn_class/simulateForest to simulate forest dynamics from a (fitted) model.

Value

An object of class `finn_class` (a torch::nn_module): an assembled but un-fitted FINN model, ready to pass to `fit()` or `simulateForest()`.

References

Pichler, M., & Käber, Y. (2026). Inferring processes within dynamic forest models using hybrid modelling. *Methods in Ecology and Evolution*. doi:10.1111/2041210x.70347

`FINN.seed` *Set Seed for Reproducibility in R and Torch*

Description

This function sets the seed for both R's random number generator and Torch's random number generator, ensuring reproducibility across operations that involve both R and Torch.

Usage

```
FINN.seed(seed)
```

Arguments

`seed`	An integer value to set as the seed for both R and Torch. This ensures that random operations in both environments produce consistent results.

Details

The function calls `set.seed()` to set the seed for R's random number generator and `torch::torch_manual_seed()` to set the seed for Torch's random number generator. This is useful for ensuring reproducibility in scripts that rely on both R and Torch for random operations.

Value

This function does not return a value. It sets the seed internally for both R and Torch.

Examples

```
FINN.seed(123)
# Now both R and Torch are seeded with 123, ensuring reproducible results
```

`fit` *Fit FINN*

Description

Fit (calibrate) a FINN model end-to-end by gradient-descent optimization. All mechanistic process parameters, environmental-response coefficients and, if present, the weights of any hybrid (DNN-based) processes are estimated jointly in a single optimization, rather than calibrating each process in isolation.

Usage

```
fit(
  model,
  data = NULL,
  env,
  disturbance = NULL,
  patches = 100L,
  patch_size = 0.1,
  init_cohort = NULL,
  epochs = 20L,
  lr = 0.01,
  lr_scheduler = "none",
  lr_scheduler_params = list(),
  loss = c(dbh = "mse", ba = "mse", trees = "poisson", growth = "mse", mortality =
    "binomial", regeneration = "nbinom"),
  weights = "auto",
  optimizer = optim_ignite_adam,
  batchsize = NULL,
  device = c("cpu", "gpu"),
  update_step = 1L,
  start_time = 1L,
  plot_progress = TRUE,
  folder = NULL,
  checkpoints = 100L,
  shuffle = TRUE,
  record_gradients = FALSE,
  env_autoscale = TRUE,
  clip_norm = 2,
  ...
)
```

Arguments

`model`	(`finn_class`) Object of class `finn_class` created by finn.

data (data.table|data.frame)
Data about demographic rates and stand variables must be passed as data.table or data.frame.

Optional columns that change how the responses are scored:

- period_length — how many simulated timesteps each observation's inventory interval spans. The rate responses (growth, mortality) are then compared against the model's **mean** over that interval, and regeneration against its **sum** (recruits accumulate). Absent or NA means each observation is compared against a single timestep. NOTE: this must currently be the SAME for every site — a per-site interval is not supported yet, so inventories with varying remeasurement gaps need filtering to a constant interval first.
- n_at_risk — trees behind each observed mortality proportion; used to weight the binomial term (see weights).
- growth_n — trees behind each observed growth mean; used to weight the squared-error term, since the variance of a mean is σ^2/n.

env (data.table|data.frame)
Data with environmental covariates must be passed as data.table or data.frame.

disturbance (data.table|data.frame)
Data with disturbance rates must be passed as data.table or data.frame.

patches (integer(1))
Number of patches.

patch_size (numeric(1))
Patch size.

init_cohort (CohortMat)
Initial cohort matrix of class CohortMat, created by CohortMat

epochs (integer(1))
Number of iteration steps.

lr (numeric(1))
Learning rate of the optimizer.

lr_scheduler (character(1)|function)
Learning-rate schedule applied on top of the constant lr above. "none" (default) keeps lr constant, reproducing the previous behavior exactly. Built-ins: "step" (decay by gamma every step_size epochs), "exponential" (multiply by gamma every epoch), "cosine" (cosine-anneal to eta_min over T_max epochs), "plateau" (reduce by factor after patience epochs without improvement in the total loss). Advanced: pass a function(optimizer) returning a torch lr scheduler object (must implement $step()).

lr_scheduler_params
(list())
Tuning overrides for the scheduler chosen via lr_scheduler, e.g. list(step_size = 50, gamma = 0.5) for "step", list(T_max = 200) for "cosine", list(factor = 0.5, patience = 20) for "plateau". Unset entries fall back to defaults scaled off epochs/lr.

loss (character(6))
Named vector of the different losses. Names should be dbh, ba, trees, growth, mortality, and regeneration. Supported losses are mse, poisson, nbinom, gaussian, and binomial. binomial is a Bernoulli/binomial negative log-likelihood intended for mortality; it expects both the prediction and the observation to be proportions in [0, 1].

weights ("auto" or numeric(6))
Weights of the six losses, in the order dbh, ba, trees, growth, mortality, regeneration.

Weights must account for the raw scale of each loss, not just its importance. The six terms are summed, and their raw magnitudes differ by orders of magnitude: on the bundled FIA data dbh is an MSE in cm^2 (~430 at the start of training) while growth is an MSE on a ratio (~0.03) — a factor of ~1e4. Weights chosen by importance alone are therefore dominated by whichever response happens to have the largest units, and the rest receive too little gradient to learn from. With equal weights, dbh takes ~87% of the FIA objective and growth ~1%; growth then never improves.

"auto" (the default) fixes this without needing to know the units. Each loss is divided by its **intercept-only baseline** — the loss you would get by predicting the single best constant (the null model). Every term then measures the same thing on the same scale: the fraction of its own null deviance. For a squared-error term the baseline is exactly $var(y)$, so this reduces to scaling by $1/sd(y)^2$; but it generalises to the Poisson, negative-binomial and binomial terms, where a standard deviation is not the right scale. It is the same idea as env_autoscale = TRUE, applied to the responses rather than the predictors.

A useful side effect: the per-response values in m$history become directly interpretable as **"how much better than the mean"** — 1 is no better than the intercept, below 1 is better, above 1 is worse. The baselines are stored in m$loss_baseline, the resulting weights in m$loss_weights, and both are reported once when fitting starts.

Passing a numeric(6) uses those weights as-is and disables the scaling. On the FIA example, balancing the objective is worth ~+0.24 Spearman on held-out growth — far more than tuning lr or epochs. See the "Fitting FINN to forest inventory data" vignette.

optimizer (torch_optimizer_generator)
Optimizer from the torch package.

batchsize (integer(1))
Batch size, model will be trained in random batch sizes of the data to preserve memory and improve convergence.

device (character(1))
Should the model be fitted on the CPU or the GPU (Graphic card). Support is only for NVIDIA GPUs available.

update_step (integer(1))
Number of steps for which the gradient should be calculated. Automatic differentation becomes slow for larger update steps and the risk of vanishing gradients increases.

`start_time` (integer(1))
Starting from which year should the model be fitted. Can be used to use on burn-in.

`plot_progress` (logical(1))
Plot fitting progress (losses) or not.

`folder` (character(1))
Path to folder for saving checkpoint models. If NULL, no models will be saved during the training.

`checkpoints` (integer(1))
Interval size in epochs for saving checkpoint models.

`shuffle` (logical(1))
Shuffle data or not.

`record_gradients` (logical(1))
Record the gradients of all parameters or not. Can get large for many epochs.

`env_autoscale` (logical(1))
If TRUE, FINN z-standardizes the environmental predictors in env internally: the per-variable mean and standard deviation are learned from the training env and stored on the model, then re-applied automatically at every predict()/simulate() call. This lets you pass raw (untransformed) env for both calibration and prediction; FINN guarantees an identical transformation at both stages. Recommended (and the default) for numerical stability when predictors are on different scales. Set FALSE to use env exactly as supplied (e.g. if you have already standardized it yourself). The learned constants are available as model$env_scaling.

`clip_norm` (numeric(1)|list())
Gradient-norm budget passed to torch::nn_utils_clip_grad_norm_(), applied separately to each of three parameter groups (mechanistic per-species rates, env-effect networks, loss-distribution nuisance parameters) rather than once globally across all parameters. A single number (default 2.0) applies the same budget to every group; a named list/vector keyed by "mechanistic"/"nn"/"loss" overrides individual groups, e.g. clip_norm = list(loss = 5, nn = 1).

`...` Additional arguments passed to optimizer.

Details

Calibration relies on the fact that FINN is implemented in torch for R and is therefore fully differentiable: at each simulated time step the predicted stand variables and demographic rates are compared to the observed data through a joint loss function, and gradients of this loss with respect to every model parameter are obtained via automatic differentiation (backpropagation) and used to update the parameters with a torch optimizer (Adam, torch::optim_ignite_adam, by default).

The joint loss is the sum of per-variable losses (akin to negative log-likelihoods), one for each of dbh, ba (basal area), trees (number of trees), growth, mortality and regeneration. Following Pichler & Käber (2026), reasonable choices are mean squared error ("mse", equivalent to a Gaussian likelihood) for dbh and ba, Poisson likelihood for trees, negative binomial ("nbinom") for regeneration, and "mse" for growth as a continuous rate. mortality is an observed *proportion* of trees that died and the model predicts it through a sigmoid, so it defaults to "binomial" — a

Bernoulli/binomial likelihood (binary cross-entropy, which admits fractional targets). This respects the [0, 1] support and the mean-variance link of a proportion, both of which `"mse"` ignores (the model also supports `"gaussian"` and `"poisson"` as alternatives via the `loss` argument; see Appendix B of the paper for details). Each loss can be weighted individually via `weights`, and missing values in the observed data are masked out of the corresponding loss term. The model is trained for `epochs` iterations over the (optionally batched and shuffled) data using `optimizer` with learning rate `lr`.

Backpropagating gradients through a long simulated time series is prone to vanishing gradients and is computationally expensive. FINN therefore uses truncated backpropagation through time: the computational graph is detached every `update_step` simulated years, gradients are accumulated and the parameters are updated, before the simulation continues. Smaller `update_step` values are faster and avoid vanishing gradients but provide a more short-sighted learning signal; larger values let the loss integrate over a longer trajectory at increased computational cost. `start_time` allows discarding an initial burn-in period of the simulation from the loss, and `checkpoints/folder` allow periodically saving the model state during training.

`growth` is compared as a **relative** rate (`dbh/dbh_before - 1`), which is the model's native parameter (`dbh_new = dbh * (1 + g)`). Training against the absolute diameter increment (`dbh * g`) instead was tested over three seeds and was worse on every one — even when scored on the absolute scale — and about four times more seed-variable, because it couples the growth parameter to whichever trees happen to be present.

Value

The fitted `model`, invisibly. `fit()` trains the model in place, so the returned object is the one passed in, now carrying the training results (e.g. `$history`, `$loss_weights`, `$loss_baseline`).

`growth` *Calculate growth*

Description

This function calculates growth based on specified parameters.

Usage

```
growth(
  dbh,
  species,
  parGrowth,
  pred,
  light,
  light_steepness = 10,
  debug = FALSE,
  trees = NULL
)
```

Arguments

dbh	torch.Tensor Diameter at breast height.
species	torch.Tensor species of tree.
parGrowth	torch.Tensor Growth parameters.
pred	torch.Tensor Predicted values.
light	torch.Tensor Accumulated Light.
light_steepness	numeric Steepness of the light-response sigmoid. Defaults to 10.
debug	logical If TRUE, return the intermediate components as a list. Defaults to FALSE.
trees	torch.Tensor (Optional) Number of trees per cohort. Defaults to NULL.

Value

torch.Tensor A tensor representing the forest plot growth.

---

height *Calculate the height of a tree based on its diameter at breast height and an allometry parameter*

Description

Calculate the height of a tree based on its diameter at breast height and an allometry parameter

Usage

```
height(dbh, parHeight)
```

Arguments

dbh	A numeric value representing the diameter at breast height of the tree in cm.
parHeight	A numeric value representing the species height allometry.

Details

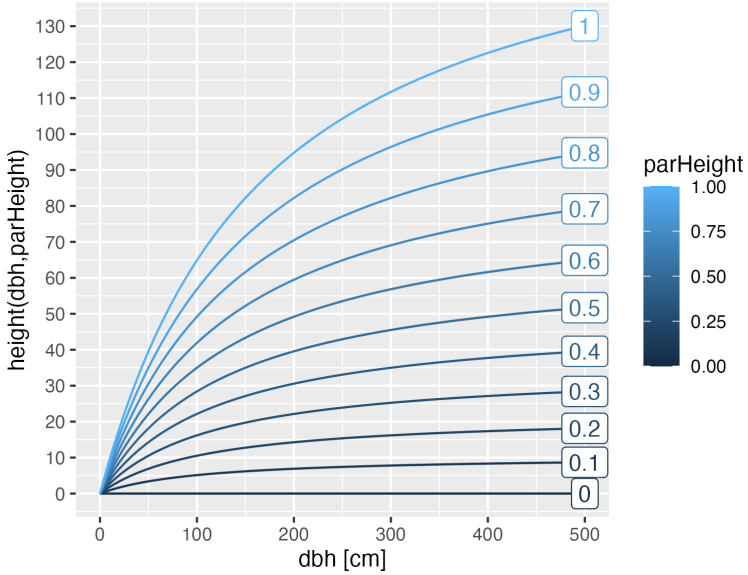
This function calculates the height of a tree based on the diameter at breast height (dbh) and a parameter parHeight.

The height is calculated using the formula:

$$height = \left(\exp\left(\frac{(\text{dbh} \times \text{parHeight})}{(\text{dbh} + 100)}\right) - 1\right) \times 100 + 0.001$$

where dbh is the diameter at breast height of the tree in cm and parHeight is an allometric species specific parameter.

All parameters of parHeight from 0 to 1 result in physiologicaly plausible heights. The range from 0.3 to 0.9 results in realistic tree heights. Values of parHeight close to 1 are physiologically almost impossible, below 0.3 is suitable for small tree species and shrubs.



Value

A numeric value representing the calculated height of the tree.

Examples

```
height(30, 0.5)
height(c(30), c(0.5,0.3))
height(c(30,20), c(0.5))
```

`makeInitCohorts` *Make initial cohorts for FINN*

Description

This function prepares `obs_df` in the exact format expected by `FINN::CohortMat$new()` and calls it. The required long-table schema is:

- **siteID** integer index of sites $1..S$.
- **patchID** integer index of patches within site $1..P_s$.
- **species** integer species code $1..sp$.
- **dbh** numeric DBH in cm (either exact or binned midpoints).
- **trees** integer count of trees in the cohort.

Usage

```
makeInitCohorts(
  init_trees,
  dbh_binsize = NULL,
  min_dbh = NULL,
  Nspecies,
  treeID_table = FALSE,
  singleCohortTreeNames = NULL
)
```

Arguments

init_trees	data.table of initial trees with columns siteID, patchID, species, dbh, treeName; optional trees for pre-counted individuals.
dbh_binsize	numeric or NULL. Bin width in cm. If NULL, keep exact DBH.
min_dbh	numeric or NULL. Lower bound for binning. If NULL, uses min DBH.
Nspecies	integer. Number of species levels passed to sp.
treeID_table	logical. If TRUE, also return the cohort table used.
singleCohortTreeNames	character vector or NULL. Tree names excluded from binning and kept as single-tree cohorts.

Details

Aggregates initial trees into cohorts by DBH bins or exact DBH and constructs a FINN::CohortMat object.

Internally calls:

```
FINN::CohortMat$new(
  obs_df = <data.frame with columns siteID, patchID, species, dbh, trees>,
  dbh    = NULL,
  trees  = NULL,
  species= NULL,
  dims   = c(S, P, K),   # inferred from obs_df
  sp     = Nspecies,     # passed from argument
  device = "cpu"
)
```

Key fields of the resulting R6 object:

- dbh, trees, species: tensors per cohort.
- dims: integer vector c(S, P, K) for sites, patches, cohorts.
- sp: integer number of species.
- device: "cpu" or "cuda".
- dbh_r, trees_r, species_r: R arrays.
- obsDF2arrays: method converting obs_df into arrays.

Value

If `treeID_table = FALSE`, a `FINN::CohortMat` object. If `TRUE`, a list with:

- `initCohort`: the `CohortMat` object.
- `init_trees`: the `obs_df` passed to `CohortMat$new()`.

See Also

`CohortMat`

`makeObsData` *Create observation data from trees*

Description

Derives stand metrics by site/patch/species/year from tree measurements, filters sites, harmonizes years, optionally aggregates by site.

Usage

```
makeObsData(
  tree_dt,
  plotsize,
  aggregate_by_site = TRUE,
  minNyears = 2,
  fix_period_length = NULL,
  dbh_growth_thresh = c(-10, 50),
  Npatches = NULL,
  Nspecies = NULL,
  NspeciesQuantile = NULL
)
```

Arguments

`tree_dt`	data.table of tree records with siteName, patchName, year, treeName, species_name, dbh, `status` (one of "alive", "new", "dead") and `living`. Mortality is derived from `status`; a `mort` column, if present, is ignored.
`plotsize`	numeric plot area used to scale recruitment.
`aggregate_by_site`	logical. Aggregate patches to site level. Default TRUE.
`minNyears`	integer or NULL. Keep sites with at least this many years and equal counts across patches. Default 2.
`fix_period_length`	integer or NULL. If set, drop sites whose inventory interval differs. Default NULL.

dbh_growth_thresh
: length-2 numeric or NULL. Drop sites where any tree dbh change falls outside this range. Default c(-10, 50).

Npatches
: integer or NULL. If set, keep only sites with exactly this many patches.

Nspecies
: integer or NULL. Cap species to the top N (others merged to "other").

NspeciesQuantile
: numeric in (0,1] or NULL. Choose the smallest N covering this fraction of individuals; overrides Nspecies if supplied.

Value

A list with:

- obs_dt: observations at site or patch level. growth is the mean relative diameter increment (dbh/dbh_before - 1), and growth_n is how many trees that mean rests on. Both matter: means are aggregated across patches TREE-WEIGHTED (by growth_n), because that is the quantity the model predicts - it simulates sum(g*trees)/sum(trees) over the whole site, so an unweighted mean of patch means would be a different number. growth_n also travels with the response so fit can weight the likelihood by it: 34% of patch-level growth observations rest on a single tree while others average 30+, and the variance of a mean is σ^2/n. Mortality comes back as a closed-cohort pair of counts, n_at_risk (trees alive at the start of the interval) and n_died (how many of them were dead at the end), plus the derived rate mort = n_died / n_at_risk (NA where no cohort was at risk). The counts are the binomial response; pass them to fit with mortality = "binomial".
- tree_dt: input trees with added growth fields and species recode.

---

mortality *Mortality*

Description

Mortality

Usage

```
mortality(
  dbh,
  species,
  trees,
  parMort,
  pred,
  light,
  base_steepness = 5,
  debug = FALSE,
  growth = NULL
)
```

Arguments

dbh	dbh
species	species
trees	trees
parMort	parMort
pred	predictions
light	available light
base_steepness	numeric Steepness of the shade-response sigmoid. Defaults to 5.
debug	logical If TRUE, return the intermediate components as a list. Defaults to FALSE.
growth	torch.Tensor (Optional) Growth entering the mortality response; defaults to the model's current growth.

Value

A torch tensor of per-cohort mortality probabilities; or, if debug = TRUE, a list of the intermediate components.

---

np_runif *Generate random numbers from a uniform distribution*

Description

This function generates random numbers from a uniform distribution with specified low and high values and size similar to np.random.uniform in Python.

Usage

```
np_runif(low, high, size)
```

Arguments

low	numeric Lower bound of the uniform distribution.
high	numeric Upper bound of the uniform distribution.
size	numeric Size of the output array.

Value

array A numeric array of random numbers.

Examples

```
np_runif(0, 1, c(2, 3))
```

obsDF2arrays *Convert observation data frame to arrays*

Description

Convert observation data frame to arrays

Usage

```
obsDF2arrays(obs_dt, additional_cols = character(0))
```

Arguments

obs_dt	data.frame The observation data table containing siteID, patchID, cohortID, species, dbh, and trees columns.
additional_cols	character vector Optional. Additional columns to be included as arrays.

Value

A list of arrays for species, dbh, trees, and additional columns.

Examples

```
obs_dt <- data.frame(
  siteID = c(1, 1, 2), patchID = c(1, 2, 1), cohortID = c(1, 1, 2),
  species = c(1, 2, 1), dbh = c(10, 20, 30), trees = c(100, 200, 150),
  height = c(5, 10, 15))
result <- obsDF2arrays(obs_dt, additional_cols = c("height"))
```

plot.FINNale *Plot ALE curves of a FINN model*

Description

Plots the accumulated local effect (ALE) curves produced by ALE() as a grid: one row per demographic process (growth, mortality, regeneration) and one column per environmental predictor. When several species are present, their curves are overlaid in the same panel as one coloured line each.

Usage

```
## S3 method for class 'FINNale'
plot(x, process = NULL, scale = FALSE, ...)
```

Arguments

x	an object of class FINNale (returned by ALE()).
process	(character|NULL) restrict the plot to one process ("growth", "mortality" or "regeneration"); NULL (default) plots all three.
scale	(logical(1)) if TRUE, divide each curve by the SD of its process x species rate so the effects are dimensionless and comparable (a shared y-axis is then used); if FALSE (default) the raw ALE is shown with free y-axes.
...	currently ignored.

Value

invisibly, the ggplot object.

---

pred2DF *Convert Prediction Arrays to Data Frames*

Description

This function takes prediction arrays from a model output and converts them into data frames. The data frames can be returned in either 'wide' or 'long' format. The function processes site-level, patch-level, and cohort-level predictions.

Usage

```
pred2DF(pred, format = "wide")
```

Arguments

pred	A list containing prediction arrays for site-level, patch-level, and cohort-level data. Each element of pred should have a structure similar to what is outlined in the details section.
format	A character string indicating the desired format of the output data frames. Must be either "wide" (default) or "long".

Details

The pred argument should be a list containing at least a Predictions element, which itself is a list of arrays. The arrays represent predictions for different metrics such as dbh/ba, tree counts, AL (aboveground live biomass), growth rates, mortality rates, and regeneration rates for sites, patches, or cohorts. The dimensionality of the arrays should correspond to different factors, such as siteID, year, species, and optionally patch or cohortID.

The function first converts each prediction array into a data frame, properly naming and converting the relevant dimensions. It then merges these data frames by common identifiers such as siteID, year, species, patch, and cohortID. Depending on the format parameter, the data frames are returned in either a wide format (one row per site/patch/cohort per year with multiple columns for different metrics) or a long format (one row per site/patch/cohort per year per metric).

Value

A list of data frames. The list may contain up to three elements: `site`, `patch`, and `cohort`, corresponding to the processed site-level, patch-level, and cohort-level predictions, respectively.

Examples

```
# a minimal prediction object of the shape pred2DF() expects
# (normally produced by predict()/simulateForest()):
metrics <- c("dbh", "ba", "trees", "growth", "mort", "reg", "r_mean_ha")
arr <- array(1, dim = c(1, 2, 3), dimnames = list(1, 1:2, 1:3))  # [site, year, species]
pred <- list(Predictions = list(Site = stats::setNames(
  lapply(metrics, function(m) arr), metrics)))
result <- pred2DF(pred, format = "long")
head(result$site)
```

`predict.finn_class` *Predict from a FINN model*

Description

Predict from a FINN model

Usage

```
## S3 method for class 'finn_class'
predict(
  object,
  env,
  disturbance = NULL,
  patches = 100L,
  patch_size = 0.1,
  init_cohort = NULL,
  device = c("cpu", "gpu"),
  return_cohorts = FALSE,
  debug = FALSE,
  ...
)
```

Arguments

`object`	(`finn_class`) Object of class `finn_class` created by finn.
`env`	(`data.table`|`data.frame`) Data with environmental covariates must be passed as `data.table` or `data.frame`.
`disturbance`	(`data.table`|`data.frame`) Data with disturbance rates must be passed as `data.table` or `data.frame`.

patches	(integer(1)) Number of patches.
patch_size	(numeric(1)) Patch size.
init_cohort	(CohortMat) Initial cohort matrix of class CohortMat, created by CohortMat.
device	(character(1)) Should the simulation run on the CPU or the GPU (Graphics card). Support is only available for NVIDIA GPUs.
return_cohorts	Controls whether the raw per-cohort state is returned in addition to the aggregated site output. Storing cohorts every timestep is expensive, so the default is FALSE (none). Use TRUE (or "all") for every timestep, "last" for the final timestep only, or an integer vector of timesteps to store just those. Recorded cohorts appear as $long$cohort / $wide$cohort.
debug	(logical(1)) Debug modus or not. If TRUE, individual tree states are stored.
...	Advanced options forwarded to the internal simulator, chiefly batchsize (split the sites into batches of this many to cap memory for very large runs). The default processes all sites in one batch and is what almost all users want.

Details

Simulate from a (fitted) FINN model. This is an S3 method for the stats::predict generic, so it is dispatched as predict(model, ...).

Value

A named list of predictions. $long$site (and $wide$site) give the site-level results (columns siteID, year, species, variable, value). When return_cohorts is set, $long$cohort / $wide$cohort add the per-cohort state (dbh, trees, species, growth g, mortality m, ...) for the requested timesteps.

---

regeneration *Calculate the regeneration of forest patches based on the input parameters*

Description

This function calculates the regeneration of forest patches based on species information, regeneration parameters, prediction values, and available light.

Usage

```
regeneration(species, parReg, pred, light, debug = FALSE)
```

Arguments

species	torch.Tensor species information.
parReg	torch.Tensor Regeneration parameters. 0 <= parReg <= 1 This parameter denotes the fraction of light needed for a species to regenerate. In general low values for high regeneration and high values for low regeneration.
pred	torch.Tensor Prediction values.
light	torch.Tensor Available light variable for calculation.
debug	logical If TRUE, return the intermediate components as a list. Defaults to FALSE.

Value

torch.Tensor Regeneration values for forest patches.

---

resolveSiteIDs *Resolve site, patch, and year indices for FINN inputs*

Description

Joins tree, environment, and observation tables; assigns integer indices for site, patch, period; standardizes species coding; and returns aligned data plus optional initial cohorts.

Usage

```
resolveSiteIDs(tree_dt, env_dt, obs_dt, createInitCohorts = TRUE)
```

Arguments

tree_dt	data.table with tree-level data including siteName, patchName, year, species_name, dbh, status, living, and optional trees.
env_dt	data.table with environment data including siteName and year.
obs_dt	data.table with observations including siteName, patchName, year, species_name, and stand metrics (ba, dbh, trees, growth, mort, n_at_risk, n_died, reg), as returned by `makeObsData`.
createInitCohorts	logical. If TRUE, build FINN initial cohorts from trees with `living == TRUE` and `year == 1`. Default TRUE.

Value

A list with:

- `siteID_dt`: site/patch/year index map.
- `tree_dt`: tree table with indices and standardized species.
- `env_dt`: environment table with indices.

- obs_dt: site-aggregated observations by species.
- obs_dt_patches: patch-level observations by species.
- species_dt: species lookup (species, species_name).
- initCohorts (optional): FINN CohortMat object.
- initCohort_dt (optional): trees used to build cohorts.

---

rweibull_cohorts *Generate Cohorts Using Weibull Distribution*

Description

This function generates cohort data for tree populations using the Weibull distribution based on specified tree counts, diameter at breast height (DBH) shape, and scale parameters.

Usage

```
rweibull_cohorts(
  trees = NULL,
  dbh_shape = NULL,
  dbh_scale = NULL,
  dbh_class_range = 1,
  siteID = 1,
  patchID = 1,
  species = 1
)
```

Arguments

trees	Integer vector specifying the number of trees for each cohort. If a single integer is provided, it will be replicated for each draw.
dbh_shape	Numeric vector specifying the shape parameters of the Weibull distribution for DBH for each cohort. If a single numeric value is provided, it will be replicated for each draw.
dbh_scale	Numeric vector specifying the scale parameters of the Weibull distribution for DBH for each cohort. If a single numeric value is provided, it will be replicated for each draw.
dbh_class_range	Numeric value specifying the range of DBH classes. Default is 1.
siteID	Integer vector specifying the site ID for each cohort. If a single integer is provided, it will be replicated for each draw. Default is 1.
patchID	Integer vector specifying the patch ID for each cohort. If a single integer is provided, it will be replicated for each draw. Default is 1.
species	Integer vector specifying the species ID for each cohort. If a single integer is provided, it will be replicated for each draw. Default is 1.

Details

The function generates cohort data by drawing samples from the Weibull distribution for each cohort based on the specified shape and scale parameters. The resulting DBH values are binned into classes, and the cohort data is generated accordingly.

Value

A data frame containing the cohort data with columns for site ID, patch ID, cohort ID, species, number of trees, and DBH.

Examples

```
obs_df <- rweibull_cohorts(
  trees = c(300, 10),
  dbh_shape = c(3, 3),
  dbh_scale = c(1, 50),
  dbh_class_range = 0.1,
  siteID = c(1, 1),
  patchID = c(1, 1),
  species = c(3, 4)
)
head(obs_df)
```

---

simulateForest *Simulate*

Description

Simulate

Usage

```
simulateForest(
  model,
  env,
  disturbance = NULL,
  patches = 100L,
  patch_size = 0.1,
  init_cohort = NULL,
  device = c("cpu", "gpu"),
  return_cohorts = FALSE,
  debug = FALSE,
  ...
)
```

Arguments

model	(finn_class) Object of class finn_class created by finn.
env	(data.table|data.frame) Data with environmental covariates must be passed as data.table or data.frame.
disturbance	(data.table|data.frame) Data with disturbance rates must be passed as data.table or data.frame.
patches	(integer(1)) Number of patches.
patch_size	(numeric(1)) Patch size.
init_cohort	(CohortMat) Initial cohort matrix of class CohortMat, created by CohortMat.
device	(character(1)) Should the simulation run on the CPU or the GPU (Graphics card). Support is only available for NVIDIA GPUs.
return_cohorts	Controls whether the raw per-cohort state is returned in addition to the aggregated site output. Storing cohorts every timestep is expensive, so the default is FALSE (none). Use TRUE (or "all") for every timestep, "last" for the final timestep only, or an integer vector of timesteps to store just those. Recorded cohorts appear as $long$cohort / $wide$cohort.
debug	(logical(1)) Debug modus or not. If TRUE, individual tree states are stored.
...	Advanced options forwarded to the internal simulator, chiefly batchsize (split the sites into batches of this many to cap memory for very large runs). The default processes all sites in one batch and is what almost all users want.

Details

Simulate from a fitted FINN model. This is a thin, backwards-compatible alias for predict.finn_class; new code should call predict(model, ...) directly.

Value

A named list of simulation results. The patch-averaged, stand-level state variables and demographic rates are in $long$site / $wide$site (long format: siteID, year, species, variable, value). When return_cohorts is set, $long$cohort / $wide$cohort hold the raw per-cohort state for the requested timesteps.

`summary.finn_class` *Summarise a fitted FINN model*

Description

Prints, per process and per species, the environmental variable importance and the average conditional effects. Both are derived from the model's conditional effects, which are computed once and cached — so if `ALE()` was already run (or a previous `summary()`), the default path needs no further simulation.

Usage

```
## S3 method for class 'finn_class'
summary(
  object,
  env = NULL,
  init_cohort = NULL,
  importance = c("ale", "permutation"),
  env_autoscale = TRUE,
  sim_seed = 42L,
  nperm = 20L,
  scale = TRUE,
  ...
)
```

Arguments

`object`	(`finn_class`) fitted model.
`env`, `init_cohort`	(`NULL` or data) interpretation data; `NULL` uses the cached training data (see `ALE()`).
`importance`	(`character(1)`) `"ale"` (default) = analytical ALE-variance importance (cheap, from the cache); `"permutation"` = re-simulating permutation importance (see `feature_importance()`), cached on the model under the same input key.
`env_autoscale`	(`logical(1)`) see `ALE()`.
`sim_seed`	(`integer(1)`) seed for the simulation (see `ALE()`).
`nperm`	(`integer(1)`) replicates for `importance = "permutation"`.
`scale`	(`logical(1)`) for `importance = "ale"`, divide `Var(ALE)` by the process x species rate variance so the importances are dimensionless and comparable across processes and species (Sobol-style). Default `TRUE`.
`...`	passed through (e.g. to `predict()` for the permutation option).

Value

invisibly, a list with `importance`, `average_conditional_effects`, and `method`.

Index