

Package: sjSDM (via r-universe)

October 17, 2026

Type Package

Title Scalable Joint Species Distribution Modeling

Version 1.1.0

Description A scalable and fast method for estimating joint Species Distribution Models (jSDMs) for big community data, including eDNA data. The package estimates a full (i.e. non-latent) jSDM with different response distributions (including the traditional multivariate probit model). The package allows to perform variation partitioning (VP) / ANOVA on the fitted models to separate the contribution of environmental, spatial, and biotic associations. In addition, the total R-squared can be further partitioned per species and site to reveal the internal metacommunity structure, see Leibold et al., [doi:10.1111/oik.08618](https://doi.org/10.1111/oik.08618). The internal structure can then be regressed against environmental and spatial distinctiveness, richness, and traits to analyze metacommunity assembly processes. The package includes support for accounting for spatial autocorrelation, for grouping-factor random effects in the environmental formula, and for deep neural networks, which can replace the linear predictor or sit beside it as one block of a formula that also holds interpretable linear terms. As described in Pichler & Hartig (2021) [doi:10.1111/2041-210X.13687](https://doi.org/10.1111/2041-210X.13687), scalability is achieved by using a Monte Carlo approximation of the joint likelihood implemented via 'torch', which can be run on CPUs or GPUs.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports torch, reformulas, stats, mvtnorm, utils, rstudioapi, abind, graphics, grDevices, Metrics, parallel, mgcv, cli, crayon, ggplot2, checkmate, mathjaxr, beeswarm, qgam, scales, viridis

Suggests testthat, knitr, rmarkdown, fields

URL <https://github.com/TheoreticalEcology/s-jSDM/>

BugReports <https://github.com/TheoreticalEcology/s-jSDM/issues>

Roxygen list(old_usage = FALSE, markdown = TRUE)

VignetteBuilder knitr

RdMacros mathjaxr

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libuv1-dev zlib1g-dev

Repository <https://theoreticalecology.r-universe.dev>

Date/Publication 2026-10-17 10:06:09 UTC

RemoteUrl <https://github.com/theoreticalecology/s-jSDM>

RemoteRef HEAD

RemoteSha 54a545277d89ea954f3dd26e2d152638e1b48d99

RemoteSubdir sjSDM

Contents

AccSGD	4
AdaBound	4
Adagrad	5
Adam	6
Adamax	6
AdamW	7
anova.sjSDM	8
bioticStruct	11
butterflies	15
check_module	16
checkModel	16
coef.sjSDM	17
DiffGrad	17
DNN	18
eucalypts	22
generateSpatialEV	23
getCor	24
getCov	25
getImportance	26
getRE	26
getSe	27
getWeights	28
importance	29
install_diagnostic	30
install_sjSDM	31
installation_help	31
internalStructure	33
is_torch_available	35

linear	35
logLik.sjSDM	40
madgrad	41
new_image	41
NN	42
plot.sjSDM	43
plot.sjSDM.DNN	44
plot.sjSDM_cv	45
plot.sjSDManova	46
plot.sjSDMimportance	47
plot.sjSDMinternalStructure	48
plotAssemblyEffects	49
plotsjSDMcoef	51
predict.sjSDM	52
print.bioticStruct	54
print.DNN	55
print.linear	55
print.NN	56
print.sjSDM	56
print.sjSDM_cv	57
print.sjSDManova	57
print.sjSDMimportance	58
print.sjSDMinternalStructure	59
re	60
residuals.sjSDM	61
RMSprop	61
Rsquared	62
setWeights	63
SGD	63
simulate.sjSDM	64
simulate_SDM	64
sjSDM	66
sjSDM_cv	73
sjSDMControl	76
summary.sjSDM	77
summary.sjSDM_cv	78
summary.sjSDManova	78
update.sjSDM	80

AccSGD

AccSGD

Description

Deprecated. 'torch' has no AccSGD implementation, so this redirects to [SGD](#).

Usage

```
AccSGD(kappa = 1000, xi = 10, small_const = 0.7, weight_decay = 0)
```

Arguments

kappa	long step, ignored
xi	advantage parameter, ignored
small_const	small constant, ignored
weight_decay	l2 penalty on weights

Value

Anonymous function that returns optimizer when called.

References

Kidambi, R., Netrapalli, P., Jain, P., & Kakade, S. (2018, February). On the insufficiency of existing momentum schemes for stochastic optimization. In 2018 Information Theory and Applications Workshop (ITA) (pp. 1-9). IEEE.

AdaBound

AdaBound

Description

Deprecated. 'torch' has no AdaBound implementation, so this redirects to [Adam](#).

Usage

```
AdaBound(
  betas = c(0.9, 0.999),
  final_lr = 0.1,
  gamma = 0.001,
  eps = 1e-08,
  weight_decay = 0,
  amsbound = TRUE
)
```

Arguments

betas	betas
final_lr	eps, ignored
gamma	small_const, ignored
eps	eps
weight_decay	weight_decay
amsbound	amsbound

Value

Anonymous function that returns optimizer when called.

References

Luo, L., Xiong, Y., Liu, Y., & Sun, X. (2019). Adaptive gradient methods with dynamic bound of learning rate. arXiv preprint arXiv:1902.09843.

Adagrad	<i>Adagrad</i>
---------	----------------

Description

Adaptive subgradient optimizer, see Duchi et al., 2011

Usage

```
Adagrad(  
  lr_decay = 0,  
  weight_decay = 0,  
  initial_accumulator_value = 0,  
  eps = 1e-10  
)
```

Arguments

lr_decay	learning rate decay
weight_decay	l2 penalty on weights
initial_accumulator_value	initial value of the gradient accumulator
eps	fuzz factor

Value

Anonymous function that returns optimizer when called.

References

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7).

Adam

Adam

Description

Adam optimizer, see Kingma and Ba, 2014

Usage

```
Adam(betas = c(0.9, 0.999), eps = 1e-08, weight_decay = 0, amsgrad = FALSE)
```

Arguments

betas	exponential decay rates
eps	fuzz factor
weight_decay	l2 penalty on weights
amsgrad	use the AMSGrad variant

Value

Anonymous function that returns optimizer when called.

References

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Adamax

Adamax

Description

Deprecated. 'torch' has no Adamax implementation, so this redirects to [Adam](#).

Usage

```
Adamax(betas = c(0.9, 0.999), eps = 1e-08, weight_decay = 0.002)
```

Arguments

betas	exponential decay rates
eps	fuzz factor
weight_decay	l2 penalty on weights

Value

Anonymous function that returns optimizer when called.

References

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

AdamW	<i>AdamW</i>
-------	--------------

Description

Adam with decoupled weight decay, see Loshchilov and Hutter, 2019

Usage

```
AdamW(betas = c(0.9, 0.999), eps = 1e-08, weight_decay = 0.01, amsgrad = FALSE)
```

Arguments

betas	exponential decay rates
eps	fuzz factor
weight_decay	decoupled l2 penalty on weights
amsgrad	use the AMSGrad variant

Value

Anonymous function that returns optimizer when called.

References

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. International Conference on Learning Representations.

anova.sjSDM

*Anova / Variation partitioning***Description**

Compute variance explained by the three fractions env, space, associations

Usage

```
## S3 method for class 'sjSDM'
anova(
  object,
  samples = 5000L,
  verbose = TRUE,
  proposal = c("laplace", "prior"),
  ...
)
```

Arguments

object	model of object sjSDM
samples	Number of Monte Carlo samples
verbose	TRUE or FALSE, indicating whether progress should be printed or not
proposal	distribution the latent factors are drawn from. "laplace" (the default) draws from a Gaussian fitted to the per-site mode of the integrand and reweights the draws. "prior" draws from $N(0, I)$, which is what every version before 1.1.0 did. The reweighting keeps the estimator of the likelihood unbiased and raises the effective sample size by orders of magnitude at many species. The log of that estimate, which is what the fractions are built from, keeps a bias of roughly $(1/ess - 1)/(2 * samples)$ per site. See Details.
...	optional arguments which are passed to the calculation of the logLikelihood, including mc_batch, the number of sites whose Monte-Carlo draws are held in memory at once. It defaults to the model's step_size, lowered if that would exceed roughly 2.5 GB; lower it by hand on a device with less memory.

Details

The ANOVA function removes each of the three fractions (Environment, Space, Associations) and measures the drop in variance explained, and thus the importance of the three fractions.

Variance explained is measured by Deviance as well as the pseudo-R2 metrics of Nagelkerke and McFadden

In downstream functions such as [plot.sjSDManova](#) or [plot.sjSDManova](#) with add_shared=TRUE. The anova can get unstable for many species and few occurrences/observations. We recommend using large numbers for 'samples'.

The sub-models without associations, and the null model, carry no latent term at all. Up to 1.0.7 they were fitted with an identity covariance, which for a probit link only rescales the coefficients but for the count, negative binomial and gaussian links adds a fixed unit latent variance; that misfit was then reported as associations.

proposal = "prior" draws the latent factors blind and weights them by the likelihood. At many species almost every draw lands where the observed community is implausible, so the effective sample size collapses – measured at 100 species it is 3 draws out of 1000 – and the negative log-likelihood $-\log(\text{mean}(\exp(lp)))$ is biased upwards by an amount that grows with the variance of the per-draw log-probability. That variance differs between the sub-models anova compares, so it does not cancel in the differences the fractions are built from, and the sub-model that leans hardest on the latent factors is depressed most. proposal = "laplace" reduces this by sampling from a Gaussian centred on the per-site mode with the curvature of the integrand, and correcting each draw by $\log p(z) - \log q(z)$. It is a proposal, not an approximation: a poor fit costs effective sample size, not correctness. The realised fraction is returned in the ess component, where 1 means every draw counted. Read it as a per-dimension efficiency raised to the power of the latent dimension df, not as a per-site quality score. The sub-models without associations have nothing to integrate and report 1.

For the gaussian family the McFadden R-squared is a ratio of log-densities, which change with the measurement scale of the response and turn positive for small residual variances, so it is not bounded by 1 and differs between units. The deviances and the Nagelkerke R-squared do not have this problem.

anova is not defined for a model with random-effect bars in the environmental formula and raises an error. Every refit re-estimates the random blocks, so they absorb the fraction that was removed instead of cancelling from the difference. Use [getRE](#) for the group variances.

The species and sites components are not the printed table restricted to a species or a site, and they are read differently. The table composes the fractions on the log-likelihood scale and forms a single ratio at the end, so it is exact inclusion-exclusion on the joint likelihoods. The per-species and per-site values are allocations: each site's improvement over the null is apportioned across species in proportion to how much each species' conditional likelihood improved, and that share is computed separately for every sub-model. A per-species or per-site fraction is therefore a difference of two allocations computed under different shares, it is unbounded in both directions, and 1 is not a boundary for it. The shares are undefined when a sub-model is about as good as the null overall, so treat per-species and per-site values from such a model with care. The site totals are unaffected: the shares cancel when summed over species.

Value

An S3 class of type 'sjSDManova' including the following components:

results	Data frame of results.
to_print	Data frame, summarized results for type I anova.
N	Number of observations (sites).
spatial	Logical, spatial model or not.
species	individual species R2s.
sites	individual site R2s.
lls	individual site by species negative-log-likelihood values.

proposal	the sampler the latent factors were drawn from.
ess	sites-by-sub-model matrix of effective sample sizes, as a fraction of samples. One column per fitted sub-model, so a low column shows which model's likelihood rests on few draws. <code>attr("newton_decrement")</code> carries the largest Newton decrement the Laplace mode-finder left behind, per sub-model.
model	model

Implemented S3 methods are `print.sjSDManova` and `plot.sjSDManova`

See Also

`plot.sjSDManova`, `print.sjSDManova`, `summary.sjSDManova`, `plot.sjSDMinternalStructure`

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates

# fit model:
model <- sjSDM(Y = Occ,
  env = linear(data = Env, formula = ~X1+X2+X3),
  spatial = linear(data = SP, formula = ~0+X1*X2),
  family=binomial("probit"),
  verbose = FALSE,
  iter = 20) # increase iter for real analysis

# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
```

```

plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
plotAssemblyEffects(int, negatives = "floor")
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)

```

bioticStruct	<i>biotic structure</i>
--------------	-------------------------

Description

define biotic (species-species) association (interaction) structure

Usage

```

bioticStruct(
  df = NULL,
  lambda = 0,
  alpha = 0.5,
  on_diag = FALSE,
  reg_on_Cov = TRUE,
  inverse = FALSE,
  diag = FALSE
)

```

Arguments

df	degree of freedom for covariance parametrization, if NULL df is set to $\text{ncol}(Y)/2$
lambda	lambda penalty, strength of regularization: $\lambda * (\text{lasso} + \text{ridge})$
alpha	weighting between lasso and ridge: $(1 - \alpha) * \text{covariances} + \alpha * \text{covariances} ^2$
on_diag	regularization on diagonals
reg_on_Cov	regularization on covariance matrix
inverse	regularization on the inverse covariance matrix
diag	use diagonal matrix with zeros (internal usage)

Value

An S3 class of type 'bioticStruct' including the following components:

l1_cov	L1 regularization strength.
l2_cov	L2 regularization strength.
inverse	Logical, use inverse covariance matrix or not.

diag	Logical, use diagonal matrix or not.
reg_on_Cov	Logical, regularize covariance matrix or not.
on_diag	Logical, regularize diagonals or not.

Implemented S3 methods include `print.bioticStruct`

See Also

[sjSDM](#)

Examples

```
## Not run:

# Basic workflow:
## simulate community:
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)

## fit model:
model = sjSDM(Y = com$response, env = com$env_weights, iter = 50L,
              verbose = FALSE)
# increase iter for your own data

# Default distribution is binomial("probit"). Alternatively, you can use
# binomial(logit), poisson("log"), "nbinom" (with log, still somewhat
# experimental) and gaussian("identity")

coef(model)
summary(model)
getCov(model)

## plot results
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)
plot(model,group=group)

## calculate post-hoc p-values:
p = getSe(model)
summary(p)

## or turn on the option in the sjSDM function:
model = sjSDM(Y = com$response, env = com$env_weights, se = TRUE,
              family = binomial("probit"),
              iter = 2L,
              verbose = FALSE)
summary(model)

## fit model with interactions:
model = sjSDM(Y = com$response,
              env = linear(data = com$env_weights, formula = ~X1:X2 + X3),
              se = TRUE,
```

```

        iter = 2L,
        verbose = FALSE) # increase iter for your own data
summary(model)

## without intercept:
model = update(model, env_formula = ~0+X1:X2 + X3,
               verbose = FALSE)

summary(model)

## predict with model:
preds = predict(model, newdata = com$env_weights)

## calculate R-squared:
R2 = Rsquared(model)
print(R2)

# With spatial terms:
## linear spatial model
XY = matrix(rnorm(200), 100, 2)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
             spatial = linear(XY, ~0+X1:X2),
             iter = 50L,
             verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = XY)
R2 = Rsquared(model)
print(R2)

## Using spatial eigenvectors as predictors to account
## for spatial autocorrelation is a common approach:
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
             spatial = linear(SP, ~0+., lambda = 0.1),
             iter = 50L,
             verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model,
           verbose = FALSE)

internal = internalStructure(an)
plot(internal)

## Visualize community assembly effects

plotAssemblyEffects(internal)

### see ?anova.sjSDM for more details

```

```

## non-linear(deep neural network) model
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = DNN(SPV,hidden = c(5L, 5L), ~0+.),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

# Regularization
## lambda is the regularization strength
## alpha weights the lasso or ridge penalty:
## - alpha = 0 --> pure lasso
## - alpha = 1.0 --> pure ridge
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = linear(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
coef(model)
getCov(model)

# Anova
com = simulate_SDM(env = 3L, species = 15L, sites = 200L, correlation = TRUE)

XY = matrix(rnorm(400), 200, 2)
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+.),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
result = anova(model, verbose = FALSE)
print(result)
plot(result)

## visualize internal meta-community structure
internal = internalStructure(an)
plot(internal)

# Deep neural networks
## we can fit also a deep neural network instead of a linear model:
model = sjSDM(Y = com$response,
              env = DNN(com$env_weights, hidden = c(10L, 10L, 10L)),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
summary(model)

```

```

getCov(model)
pred = predict(model, newdata = com$env_weights)

## extract weights
weights = getWeights(model)

## we can also assign weights:
setWeights(model, weights)

## with regularization:
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = DNN(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
getCov(model)
getWeights(model)

## End(Not run)

```

butterflies

butterflies

Description

This dataset is from [doi:10.1111/2041210X.13106](https://doi.org/10.1111/2041210X.13106).

Usage

butterflies

Format

A 'list': List of 3.

env data.frame with 4 environmental covariates

PA Presence-absence data for 55 butterfly species

lat_lon Coordinates for the sites

Details

This is a dataset about butterfly communities. It consists of 2609 sites and 55 species.

Author(s)

Maximilian Pichler

Source

doi:10.1111/2041210X.13106

Examples

```
## Not run:
PA = butterflies$PA
E = butterflies$env
LatLon = butterflies$lat_lon

m = sjSDM(PA,
  scale(E),
  spatial = DNN(scale(LatLon), formula = ~0+.),
  se = TRUE,
  iter = 20L, # increase to 100
  step_size = 200L,
  verbose = FALSE)

summary(m)
plot(m)

## End(Not run)
```

check_module	<i>check module</i>
--------------	---------------------

Description

check if module is loaded

Usage

```
check_module()
```

checkModel	<i>check model check model and rebuild if necessary</i>
------------	---

Description

check model check model and rebuild if necessary

Usage

```
checkModel(object)
```

Arguments

object of class sjSDM

coef.sjSDM	<i>Return coefficients from a fitted sjSDM model</i>
------------	--

Description

Return coefficients from a fitted sjSDM model

Usage

```
## S3 method for class 'sjSDM'
coef(object, ...)
```

Arguments

object	a model fitted by sjSDM
...	optional arguments for compatibility with the generic function, no function implemented

Value

Matrix of environmental coefficients or list of environmental and spatial coefficients for spatial models.

DiffGrad	<i>DiffGrad</i>
----------	-----------------

Description

DiffGrad

Usage

```
DiffGrad(betas = c(0.9, 0.999), eps = 1e-08, weight_decay = 0)
```

Arguments

betas	betas
eps	eps
weight_decay	weight_decay

Value

Anonymous function that returns optimizer when called.

DNN

*Non-linear model (deep neural network) of environmental responses***Description**

specify the model to be fitted

Usage

```
DNN(
  data = NULL,
  formula = NULL,
  hidden = c(10L, 10L, 10L),
  activation = "selu",
  bias = TRUE,
  lambda = 0,
  alpha = 0.5,
  dropout = 0
)
```

Arguments

<code>data</code>	matrix of environmental predictors
<code>formula</code>	formula object for predictors
<code>hidden</code>	hidden units in layers, length of hidden corresponds to number of layers
<code>activation</code>	activation functions, can be of length one, or a vector of activation functions for each layer. Currently supported: tanh, relu, leakyrelu, selu, or sigmoid
<code>bias</code>	whether use biases in the layers, can be of length one, or a vector (number of hidden layers including (last layer) but not first layer (intercept in first layer is specified by formula)) of logicals for each layer.
<code>lambda</code>	lambda penalty, strength of regularization: $\lambda * (lasso + ridge)$
<code>alpha</code>	weighting between lasso and ridge: $(1 - \alpha) * weights + \alpha weights ^2$
<code>dropout</code>	probability of dropout rate

Details

DNN replaces the linear predictor by one network over all covariates, so the model has no interpretable coefficients left. Use [linear](#) with an [NN](#) block instead when a flexible term should sit beside interpretable linear terms.

A DNN formula is a plain model formula. [NN](#) blocks and random-effect bars are a [linear](#) formula's business and are rejected here.

Value

An S3 class of type 'DNN' including the following components:

formula	Model matrix formula
X	Model matrix of covariates
data	Raw data
l1_coef	L1 regularization strength, can be -99 if $\lambda = 0.0$
l2_coef	L2 regularization strength, can be -99 if $\lambda = 0.0$
hidden	Integer vector of hidden neurons in the deep neural network. Length of vector corresponds to the number of hidden layers.
activation	Character vector of activation functions.
bias	Logical vector whether to use bias or not in each hidden layer.

Implemented S3 methods include [print.DNN](#)

See Also

[linear](#), [NN](#), [sjSDM](#)

Examples

```
## Not run:

# Basic workflow:
## simulate community:
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)

## fit model:
model = sjSDM(Y = com$response, env = com$env_weights, iter = 50L,
              verbose = FALSE)
# increase iter for your own data

# Default distribution is binomial("probit"). Alternatively, you can use
# binomial(logit), poisson("log"), "nbinom" (with log, still somewhat
# experimental) and gaussian("identity")

coef(model)
summary(model)
getCov(model)

## plot results
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)
plot(model,group=group)

## calculate post-hoc p-values:
p = getSe(model)
summary(p)
```

```

## or turn on the option in the sjSDM function:
model = sjSDM(Y = com$response, env = com$env_weights, se = TRUE,
              family = binomial("probit"),
              iter = 2L,
              verbose = FALSE)
summary(model)

## fit model with interactions:
model = sjSDM(Y = com$response,
              env = linear(data = com$env_weights, formula = ~X1:X2 + X3),
              se = TRUE,
              iter = 2L,
              verbose = FALSE) # increase iter for your own data
summary(model)

## without intercept:
model = update(model, env_formula = ~0+X1:X2 + X3,
              verbose = FALSE)

summary(model)

## predict with model:
preds = predict(model, newdata = com$env_weights)

## calculate R-squared:
R2 = Rsquared(model)
print(R2)

# With spatial terms:
## linear spatial model
XY = matrix(rnorm(200), 100, 2)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(XY, ~0+X1:X2),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = XY)
R2 = Rsquared(model)
print(R2)

## Using spatial eigenvectors as predictors to account
## for spatial autocorrelation is a common approach:
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+., lambda = 0.1),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model,

```

```

        verbose = FALSE)

internal = internalStructure(an)
plot(internal)

## Visualize community assembly effects

plotAssemblyEffects(internal)

### see ?anova.sjSDM for more details

## non-linear(deep neural network) model
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = DNN(SPV,hidden = c(5L, 5L), ~0+.),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

# Regularization
## lambda is the regularization strength
## alpha weights the lasso or ridge penalty:
## - alpha = 0 --> pure lasso
## - alpha = 1.0 --> pure ridge
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = linear(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
coef(model)
getCov(model)

# Anova
com = simulate_SDM(env = 3L, species = 15L, sites = 200L, correlation = TRUE)

XY = matrix(rnorm(400), 200, 2)
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+.),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
result = anova(model, verbose = FALSE)
print(result)
plot(result)

## visualize internal meta-community structure

```

```

internal = internalStructure(an)
plot(internal)

# Deep neural networks
## we can fit also a deep neural network instead of a linear model:
model = sjSDM(Y = com$response,
              env = DNN(com$env_weights, hidden = c(10L, 10L, 10L)),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
summary(model)
getCov(model)
pred = predict(model, newdata = com$env_weights)

## extract weights
weights = getWeights(model)

## we can also assign weights:
setWeights(model, weights)

## with regularization:
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = DNN(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
getCov(model)
getWeights(model)

## End(Not run)

```

eucalypts

eucalypts

Description

This dataset is from [doi:10.1111/2041210x.12180](https://doi.org/10.1111/2041210x.12180).

Usage

```
eucalypts
```

Format

A 'list': List of 3.

env data.frame with 7 environmental covariates

PA Presence-absence data for 12 eucalypts species

lat_lon Coordinates for the sites

Details

This is a dataset about butterfly communities. It consists of 458 sites and 12 species.

Author(s)

Maximilian Pichler

Source

[doi:10.1111/2041210x.12180](https://doi.org/10.1111/2041210x.12180)

Examples

```
## Not run:
PA = eucalypts$PA
E = eucalypts$env
LatLon = eucalypts$lat_lon

m = sjSDM(PA,
  scale(E),
  spatial = DNN(scale(LatLon), formula = ~0+.),
  se = TRUE,
  verbose = FALSE)

summary(m)
plot(m)

## End(Not run)
```

generateSpatialEV	<i>Generate spatial eigenvectors</i>
-------------------	--------------------------------------

Description

Generates a Moran's eigenvector map of the distance matrix. See Dray, Legendre, and Peres-Neto, 2006 for more information.

Usage

```
generateSpatialEV(coords = NULL, threshold = 0)
```

Arguments

coords	matrix or data.frame of coordinates
threshold	ignore distances greater than threshold

Value

Matrix of spatial eigenvectors.

References

Dray, S., Legendre, P., & Peres-Neto, P. R. (2006). Spatial modelling: a comprehensive framework for principal coordinate analysis of neighbour matrices (PCNM). *Ecological modelling*, 196(3-4), 483-493.

getCor	<i>getCor</i>
--------	---------------

Description

get species-species association correlation matrix

Usage

```
getCor(object, which = c("residual", "re"))

## S3 method for class 'sjSDM'
getCor(object, which = c("residual", "re"))
```

Arguments

- object a model fitted by [sjSDM](#), or [sjSDM](#) with [DNN](#) object
- which "residual", the default, for the residual association matrix. "re" for the correlation a random-effect bar induces, one entry per bar; see [getCov](#) for the shape. Not defined for a bar with [re](#)(loading = "shared"), where it is 1 by construction.

Value

Matrix of dimensions species by species corresponding to the covariance (occurrence) matrix, or for which = "re" a named list with one correlation matrix per random-effect bar.

See Also

[sjSDM](#), [DNN](#), [getRE](#)

getCov	<i>getCov</i>
--------	---------------

Description

get species-species association (covariance) matrix

Usage

```
getCov(object, which = c("residual", "re"))

## S3 method for class 'sjSDM'
getCov(object, which = c("residual", "re"))
```

Arguments

object	a model fitted by sjSDM , or sjSDM with DNN object
which	"residual", the default, for the residual association matrix $\sigma\sigma' + I$. "re" for the covariance a random-effect bar induces, $\Lambda\Lambda'$, one entry per bar.

Details

which = "re" returns a list named by the grouping variables. A bar with q design columns gives the full $Jq \times Jq$ covariance of the group effect, flattened species-fast and named <species>_k<column>, so species j 's own $q \times q$ block is [getRE](#)(object)[[b]]\$cov[j, ,]; at $q = 1$ it is the plain $J \times J$ matrix named by species. A bar with [re](#)(loading = "shared") has no species dimension and returns the scalar λ^2 .

Unlike the residual covariance, $\Lambda\Lambda'$ carries no $+I$, so its rank is at most d and it is singular whenever $d < Jq$. [getCor](#) warns there: the group effects of the columns are linearly dependent, and at $d = 1$ every correlation is exactly ± 1 .

Value

Matrix of dimensions species by species corresponding to the covariance (occurrence) matrix, or for which = "re" a named list with one covariance matrix per random-effect bar.

See Also

[sjSDM](#), [DNN](#), [getRE](#)

getImportance	<i>getImportance</i>
---------------	----------------------

Description

variation partitioning with coefficients

Usage

```
getImportance(beta, sp = NULL, association, covX, covSP = NULL)
```

Arguments

beta	abiotic weights
sp	spatial weights
association	species associations
covX	environmental covariance matrix
covSP	spatial covariance matrix

Author(s)

Maximilian Pichler

getRE	<i>Get the random effects</i>
-------	-------------------------------

Description

Group-level latent factors of a model fitted with random-effect bars in the environmental formula, one entry per bar.

Usage

```
getRE(object)

## S3 method for class 'sjSDM'
getRE(object)
```

Arguments

object	a model fitted by sjSDM
--------	---

Details

Λ_b is identified only up to an orthogonal rotation of R^{d_b} , exactly as sigma is, so the quantity to report is cov, the per-species covariance $G_j = \Lambda_j \Lambda_j'$ of the bar's design columns, and mean, the posterior mean group effect $m'_g \Lambda_{jk}$ on the linear predictor. lambda and L are the raw ingredients, for a user who needs them: L is the posterior Cholesky factor, the counterpart of lme4's condVar, and $L_g L'_g$ is the posterior covariance of u_g that error bars on a group effect are built from.

A bar with `re(loading = "shared")` has no species dimension: lambda is a vector of length q and cov is $\lambda\lambda'$, a scalar for $(1 \mid g)$, rather than the species by q by q array.

Value

A list with one element per bar, each holding group (the grouping variable), terms and levels, q (the number of design columns of the bar), nlevels, df, loading, cov (species by q by q), mean (levels by q by species), lambda (species by q by df, the loading Λ) and L (levels by df by df, the posterior Cholesky factor). NULL for a model without random effects.

See Also

[sjSDM](#), [re](#), [getCov](#)

Examples

```
## Not run:
com = simulate_SDM(env = 3L, species = 5L, sites = 60L)
X = data.frame(com$env_weights)
X$plot = factor(rep(1:6, each = 10))

model = sjSDM(com$response, env = linear(X, ~ X1 + re(1 | plot, df = 2)),
              iter = 10L, verbose = FALSE)

rr = getRE(model)
dim(rr[[1]]$cov) # species by q by q
dim(rr[[1]]$mean) # groups by q by species

getCov(model, which = "re")$plot

## End(Not run)
```

getSe

Post hoc calculation of standard errors

Description

Post hoc calculation of standard errors

Usage

```
getSe(object, step_size = NULL, parallel = 0L)
```

Arguments

object	a model fitted by sjSDM
step_size	batch size for stochastic gradient descent
parallel	number of cpu cores for the data loader, only necessary for large datasets

Value

The object passed to this function but the object\$se field contains the standard errors now

getWeights	<i>Get weights</i>
------------	--------------------

Description

return weights of each layer

Usage

```
getWeights(object)

## S3 method for class 'sjSDM'
getWeights(object)
```

Arguments

object	object of class sjSDM with DNN
--------	--

Value

- layers - list of layer weights
- sigma - weight to construct covariance matrix
- state - every parameter and buffer of the model by name, including NN() blocks, random-effect blocks and the dispersion; this is what [setWeights](#) restores

importance	<i>Importance of environmental, spatial and association components</i>
------------	--

Description

Computes standardized variance components with respect to abiotic, biotic, and spatial effect groups.

Usage

```
importance(x, save_memory = TRUE, ...)
```

Arguments

<code>x</code>	object fitted by sjSDM or a list with beta, the association matrix, and the correlation matrix of the predictors, see details below
<code>save_memory</code>	use torch backend to calculate importance with single precision floats
<code>...</code>	additional arguments

Details

This approach is based on Ovaskainen et al., 2017, and also used in Leibold et al., 2021. Unlike the [anova.sjSDM](#) function in the sjSDM package, importance is not calculated by explicitly switching a particular model component of and refitting the model, but essentially by setting it ineffective.

Although we have no hard reasons to discourage the use of this function, we have decided in sjSDM to measure importance mainly based on a traditional ANOVA approach. We therefore recommend users to use the [anova.sjSDM](#).

This function is maintained hidden for comparison / benchmarking purpose, and in case there is a need to use it in the future. If you want to access it, use `sjSDM:::importance`.

Value

An S3 class of type 'sjSDMimportance' including the following components:

<code>names</code>	Character vector, species names.
<code>res</code>	Data frame of results.
<code>spatial</code>	Logical, spatial model or not.

Implemented S3 methods include [print.sjSDMimportance](#) and [plot.sjSDMimportance](#)

Author(s)

Maximilian Pichler

References

Ovaskainen, O., Tikhonov, G., Norberg, A., Guillaume Blanchet, F., Duan, L., Dunson, D., ... & Abrego, N. (2017). How to make more out of community data? A conceptual framework and its implementation as models and software. *Ecology letters*, 20(5), 561-576.

Leibold, M. A., Rudolph, F. J., Blanchet, F. G., De Meester, L., Gravel, D., Hartig, F., ... & Chase, J. M. (2021). The internal structure of metacommunities. *Oikos*.

See Also

[print.sjSDMimportance](#), [plot.sjSDMimportance](#)

install_diagnostic	<i>install diagnostic</i>
--------------------	---------------------------

Description

Print information about the 'torch' installation and the compute devices.

Usage

```
install_diagnostic()
```

Details

If the trouble shooting guide [installation_help](#) did not help with the installation, please create an issue on [issue tracker](#) with the output of this function as a quote.

Value

No return value, called to extract dependency information.

See Also

[installation_help](#), [install_sjSDM](#)

install_sjSDM	<i>Install sjSDM's dependencies</i>
---------------	-------------------------------------

Description

Install sjSDM's dependencies

Usage

```
install_sjSDM(...)
```

Arguments

... ignored, kept for backward compatibility (e.g. version = "gpu")

Details

Since version 1.1.0 sjSDM runs on 'torch' (libtorch) and no longer needs 'python', 'conda' or 'PyTorch'. The only remaining dependency is the libtorch binary, which is downloaded by [install_torch](#). This function is a thin wrapper around it so that existing scripts keep working. Whether the GPU is used is decided by the torch installation, not by an argument here.

Value

No return value, called for side effects (installation of the 'torch' binaries).

See Also

[installation_help](#), [install_diagnostic](#)

installation_help	<i>Installation help</i>
-------------------	--------------------------

Description

Trouble shooting guide for the installation of the sjSDM package

Since version 1.1.0 sjSDM runs on the 'torch' package (libtorch) and no longer needs 'python', 'conda', 'reticulate' or 'PyTorch'. Installing sjSDM from CRAN pulls in 'torch', and 'torch' then downloads the libtorch and liblancet binaries on first use. Almost every installation problem is a problem with that download.

Before you start

Check what is missing:

```
sjSDM::install_diagnostic()
```

If it reports libtorch installed: FALSE, the binaries are missing. Get them with

```
torch::install_torch()
# or, equivalently
sjSDM::install_sjSDM()
```

and restart R afterwards.

Behind a proxy or without an internet connection

'torch' can be installed from local files. Download the libtorch and liblantern archives matching your platform on a machine that has access, then point 'torch' at them before loading it:

```
Sys.setenv(TORCH_URL = "/path/to/libtorch.zip")
Sys.setenv(LANTERN_URL = "/path/to/liblantern.zip")
torch::install_torch()
```

See `?torch::install_torch` for the current variable names and the download URLs.

GPU support

There is no separate 'gpu' version of sjSDM any more, and `install_sjSDM()` has no version argument. Whether the GPU can be used is decided entirely by the 'torch' installation:

- CUDA: install a CUDA enabled 'torch' build, then check `torch::cuda_is_available()`. Pass `device = "gpu"` or `device = 0L` (the CUDA device index) to [sjSDM](#).
- Apple silicon: `torch::backends_mps_is_available()`, then `device = "mps"`.
- otherwise everything runs on the CPU, which is the default.

Migrating from sjSDM 1.0.x

The 'r-sjsdm' conda environment is no longer used and can be deleted. Model code does not change. Two things behave differently:

- results are not bit-identical to the 'PyTorch' backend. The two use different random number streams, so weight initialisation and the Monte-Carlo draws differ. Estimates agree within Monte-Carlo noise, they do not agree to the last digit.
- `sjSDMControl(mixed = TRUE)` (half precision) is accepted but ignored.

Help and bugs

To report bugs or ask for help, post a [reproducible example](#) via the [sjSDM issue tracker](#) with a copy of the [install_diagnostic](#) output as a quote.

Author(s)

Maintainer: Maximilian Pichler <maximilian.pichler@biologie.uni-regensburg.de> ([ORCID](#))

Authors:

- Maximilian Pichler <maximilian.pichler@biologie.uni-regensburg.de> ([ORCID](#))
- Florian Hartig <florian.hartig@biologie.uni-regensburg.de> ([ORCID](#))

Other contributors:

- Wang Cai <caiwang0503@163.com> [contributor]

See Also

Useful links:

- <https://github.com/TheoreticalEcology/s-jSDM/>
- Report bugs at <https://github.com/TheoreticalEcology/s-jSDM/issues>

internalStructure

Plot internal metacommunity structure

Description

Plot internal metacommunity structure

Usage

```
internalStructure(
  object,
  Rsquared = c("McFadden", "Nagelkerke"),
  fractions = c("mvp_proportional", "mvp", "discard", "proportional", "equal"),
  negatives = c("floor", "scale", "raw"),
  plot = FALSE
)
```

Arguments

object	anova object from anova.sjSDM
Rsquared	which R squared should be used, McFadden or Nagelkerke (McFadden is default)
fractions	how to handle shared fractions, one of "mvp_proportional" (the default), "mvp", "discard", "proportional" or "equal". See summary.sjSDManova for what each does.
negatives	how to handle negative R squareds
plot	should the plots be suppressed or not. Plots and returns the internal metacommunity structure of species and sites (see Leibold et al., 2022). Plots were heavily inspired by Leibold et al., 2022

Value

An object of class `sjSDMinternalStructure` consisting of a list of `data.frames` with the internal structure.

References

Leibold, M. A., Rudolph, F. J., Blanchet, F. G., De Meester, L., Gravel, D., Hartig, F., ... & Chase, J. M. (2022). The internal structure of metacommunities. *Oikos*, 2022(1).

See Also

[plot.sjSDMinternalStructure](#), [print.sjSDMinternalStructure](#)

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates

# fit model:
model <- sjSDM(Y = Occ,
               env = linear(data = Env, formula = ~X1+X2+X3),
               spatial = linear(data = SP, formula = ~0+X1*X2),
               family=binomial("probit"),
               verbose = FALSE,
               iter = 20) # increase iter for real analysis

# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
```

```
plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
plotAssemblyEffects(int, negatives = "floor")
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)
```

is_torch_available	<i>is_torch_available</i>
--------------------	---------------------------

Description

is_torch_available

Usage

is_torch_available()

Details

check whether torch is available

Value

Logical, is torch module available or not.

linear	<i>Linear model of environmental response</i>
--------	---

Description

specify the model to be fitted

Usage

linear(data = NULL, formula = NULL, lambda = 0, alpha = 0.5)

Arguments

- | | |
|---------|---|
| data | matrix of environmental predictors |
| formula | formula object for predictors |
| lambda | lambda penalty, strength of regularization: $\lambda * (lasso + ridge)$ |
| alpha | weighting between lasso and ridge: $(1-\alpha)* coefficients + \alpha coefficients ^2$ |

Details

The formula is the usual model formula with two additions, and a formula that uses neither is parsed exactly as before.

`NN(soil + ndvi, hidden = c(50, 50))` marks a group of covariates as a non-linear block. The block is a network from its own covariates to the species and its output is added to the linear predictor, so the parametric terms beside it keep their interpretation. Several blocks are allowed.

A bar adds a grouping-factor random effect with 'lme4' semantics: `(1 | plot)`, `(temp | plot)`, `(0 + temp | plot)`, `(temp || plot)`, `(1 | plot/subplot)`. `re(1 | plot, df = 2)` is the explicit form where a bar needs arguments. The group effect is a design-level latent factor with its own species covariance, see [sjSDM](#).

Rejected while the formula is parsed, each with an error naming the offending term:

- a covariate in both the parametric part and an `NN` block, or in two blocks. A linear output layer represents a linear term exactly, so the likelihood is flat along the split and only the ratio of the two penalties decides it. `~ temp + NN(.)` is an error, not a shorthand.
- `NN` inside an interaction. Put both covariates inside the block instead.
- `df` below the number of columns the bar produces, which would force the intercept-slope correlation to ± 1 , and `df` together with `loading = "shared"`, which has no latent rank to choose.
- `re` wrapping a `||` or a nested `g/h` bar. The formula parser then either drops its arguments or leaves the bar unexpanded, so the bars have to be written out: `re(1 | g, df = 2) + re(0 + x | g, df = 2)`.

Neither blocks nor bars are available in the spatial formula or in `DNN`, and `sjSDM_cv` does not support them.

Value

An S3 class of type 'linear' including the following components:

<code>formula</code>	Model matrix formula
<code>X</code>	Model matrix of covariates
<code>data</code>	Raw data
<code>l1_coef</code>	L1 regularization strength, can be -99 if <code>lambda = 0.0</code>
<code>l2_coef</code>	L2 regularization strength, can be -99 if <code>lambda = 0.0</code>
<code>nn</code>	List of the <code>NN</code> blocks the formula asks for, empty for a purely parametric formula. <code>X</code> stays the parametric design.
<code>re</code>	List of the random-effect blocks the formula's bars ask for, empty when it has none. Each holds the bar's design <code>X</code> , the group index, the levels, <code>df</code> and <code>loading</code> .

Implemented S3 methods include `print.linear`

See Also

[DNN](#), [NN](#), [sjSDM](#)

Examples

```
## Not run:

# Basic workflow:
## simulate community:
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)

## fit model:
model = sjSDM(Y = com$response, env = com$env_weights, iter = 50L,
              verbose = FALSE)
# increase iter for your own data

# Default distribution is binomial("probit"). Alternatively, you can use
# binomial(logit), poisson("log"), "nbinom" (with log, still somewhat
# experimental) and gaussian("identity")

coef(model)
summary(model)
getCov(model)

## plot results
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)
plot(model,group=group)

## calculate post-hoc p-values:
p = getSe(model)
summary(p)

## or turn on the option in the sjSDM function:
model = sjSDM(Y = com$response, env = com$env_weights, se = TRUE,
              family = binomial("probit"),
              iter = 2L,
              verbose = FALSE)
summary(model)

## fit model with interactions:
model = sjSDM(Y = com$response,
              env = linear(data = com$env_weights, formula = ~X1:X2 + X3),
              se = TRUE,
              iter = 2L,
              verbose = FALSE) # increase iter for your own data
summary(model)

## without intercept:
model = update(model, env_formula = ~0+X1:X2 + X3,
              verbose = FALSE)

summary(model)

## predict with model:
```

```

preds = predict(model, newdata = com$env_weights)

## calculate R-squared:
R2 = Rsquared(model)
print(R2)

# With spatial terms:
## linear spatial model
XY = matrix(rnorm(200), 100, 2)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(XY, ~0+X1:X2),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = XY)
R2 = Rsquared(model)
print(R2)

## Using spatial eigenvectors as predictors to account
## for spatial autocorrelation is a common approach:
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+., lambda = 0.1),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model,
            verbose = FALSE)

internal = internalStructure(an)
plot(internal)

## Visualize community assembly effects

plotAssemblyEffects(internal)

### see ?anova.sjSDM for more details

## non-linear(deep neural network) model
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = DNN(SPV,hidden = c(5L, 5L), ~0+.),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

# Regularization
## lambda is the regularization strength

```

```

## alpha weights the lasso or ridge penalty:
## - alpha = 0 --> pure lasso
## - alpha = 1.0 --> pure ridge
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = linear(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              iter = 2L, # increase iter for your own data
              verbose = FALSE)
summary(model)
coef(model)
getCov(model)

# Anova
com = simulate_SDM(env = 3L, species = 15L, sites = 200L, correlation = TRUE)

XY = matrix(rnorm(400), 200, 2)
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+.),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
result = anova(model, verbose = FALSE)
print(result)
plot(result)

## visualize internal meta-community structure
internal = internalStructure(an)
plot(internal)

# Deep neural networks
## we can fit also a deep neural network instead of a linear model:
model = sjSDM(Y = com$response,
              env = DNN(com$env_weights, hidden = c(10L, 10L, 10L)),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
summary(model)
getCov(model)
pred = predict(model, newdata = com$env_weights)

## extract weights
weights = getWeights(model)

## we can also assign weights:
setWeights(model, weights)

## with regularization:
model = sjSDM(Y = com$response,

```

```

      # mix of lasso and ridge
      env = DNN(com$env_weights, lambda = 0.01, alpha = 0.5),
      # we can do the same for the species-species associations
      biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
      verbose = FALSE,
      iter = 2L) # increase iter for your own data
getCov(model)
getWeights(model)

## End(Not run)

```

logLik.sjSDM

Extract negative-log-Likelihood from a fitted sjSDM model

Description

Extract negative-log-Likelihood from a fitted sjSDM model

Usage

```

## S3 method for class 'sjSDM'
logLik(object, individual = FALSE, ...)

```

Arguments

object	a model fitted by sjSDM
individual	returns internal ll structure, mostly for internal useage
...	optional arguments passed to internal logLik function (only used if individual=TRUE)

Details

For a model with random-effect bars this is the conditional log-likelihood at the posterior means of the group effects, not the evidence lower bound and not a marginal likelihood. The Kullback-Leibler term enters the training loss only and is visible through `object$history$train_1`. Values are therefore not comparable with those of a model fitted without a bar, and neither are information criteria built from them.

Value

Numeric value or numeric matrix if individual is true.

madgrad	<i>madgrad</i>
---------	----------------

Description

Deprecated. 'torch' has no MADGRAD implementation, so this redirects to [Adam](#).

Usage

```
madgrad(momentum = 0.9, weight_decay = 0, eps = 1e-06)
```

Arguments

momentum	strength of momentum, ignored
weight_decay	l2 penalty on weights
eps	epsilon

Value

Anonymous function that returns optimizer when called.

References

Defazio, A., & Jelassi, S. (2021). Adaptivity without Compromise: A Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization. arXiv preprint arXiv:2101.11075.

new_image	<i>new_image function</i>
-----------	---------------------------

Description

new_image function

Usage

```
new_image(  
  z,  
  cols = (grDevices::colorRampPalette(c("white", "#24526E"), bias = 1.5))(10),  
  range = c(0.5, 1)  
)
```

Arguments

z	z matrix
cols	cols for gradient
range	rescale to range

NN

*Neural network block inside an environmental formula***Description**

Marks a group of covariates as a non-linear block of the environmental predictor. The block is a feed-forward network from its own covariates to the species, and its output is added to the linear predictor, so the parametric terms beside it keep their interpretation.

Usage

```

NN(
  terms,
  hidden = c(10L, 10L, 10L),
  activation = "selu",
  bias = TRUE,
  lambda = 0,
  alpha = 0.5,
  dropout = 0
)

```

Arguments

terms	the covariates the block is a function of, written as formula terms (a + b + c, or . for every column of the data). Captured unevaluated.
hidden	hidden units per layer, length of hidden corresponds to number of layers
activation	activation functions, of length one or one per hidden layer. Currently supported: tanh, relu, leakyrelu, selu, or sigmoid
bias	whether the hidden layers carry biases. The output layer never does, it would alias with the parametric intercept.
lambda	lambda penalty on this block's weights, strength of regularization: $\lambda * (lasso + ridge)$
alpha	weighting between lasso and ridge: $(1 - \alpha) * weights + \alpha weights ^2$
dropout	probability of dropout rate

Details

A covariate may not appear both in the parametric part of the formula and in an NN() block: a linear output layer can represent a linear term exactly, so the likelihood is flat along the split and only the ratio of the two penalties decides it. `~ temp + NN(.)` is therefore an error, not a shorthand.

The block has no interpretable coefficients, so `coef()` and `summary()` report the parametric part only, and standard errors become conditional on the fitted block.

Value

An S3 class of type 'NN', a specification object holding the unevaluated terms and the architecture. It is evaluated by [linear](#) when the formula is parsed.

See Also

[linear](#), [DNN](#), [sjSDM](#)

Examples

```
## Not run:
com = simulate_SDM(env = 3L, species = 5L, sites = 100L)
X = data.frame(com$env_weights)

model = sjSDM(com$response,
              env = linear(X, ~ X1 + NN(X2 + X3, hidden = c(10L, 10L))),
              iter = 20L, verbose = FALSE)

coef(model) # the parametric part only

## switch the block off and refit
model_without = update(model, env_blocks = 1L, verbose = FALSE)

## End(Not run)
```

plot.sjSDM

Coefficients plot

Description

Plotting coefficients returned by sjSDM model. This function only for model fitted by linear, fitted by DNN is not yet supported.

Usage

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

Arguments

x a model fitted by [sjSDM](#)
 ... Additional arguments to pass to [plotsjSDMcoef](#).

Value

ggplot2 object for linear sjSDM model and nothing for DNN sjSDM model.

Author(s)

CAI Wang

See Also[plotsjSDMcoef](#)**Examples**

```
## Not run:
library(sjSDM)
# simulate community:
com = simulate_SDM(env = 6L, species = 7L, sites = 100L)

# fit model:
model = sjSDM(Y = com$response, env = com$env_weights, iter = 2L, se = TRUE,
              verbose = FALSE)

# normal plot
plot(model)

# colored by groups
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)

plot(model,group=group)

## End(Not run)
```

plot.sjSDM.DNN

Training history

Description

Plot training loss history

Usage

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

Arguments

x a model fitted by [sjSDM](#) with [DNN](#) object

... passed to plot

Value

No return value, called for side effects.

Examples

```
## Not run:
library(sjSDM)
# simulate community:
com = simulate_SDM(env = 6L, species = 7L, sites = 100L)

# fit model:
model = sjSDM(Y = com$response, env = com$env_weights, iter = 2L, se = TRUE,
              verbose = FALSE)

# normal plot
plot(model)

# colored by groups
species=c("sp1", "sp2", "sp3", "sp4", "sp5", "sp6", "sp7")
group=c("mammal", "bird", "fish", "fish", "mammal", "amphibian", "amphibian")
group = data.frame(species=species, group=group)

plot(model, group=group)

## End(Not run)
```

plot.sjSDM_cv

Plot elastic net tuning

Description

Plot elastic net tuning

Usage

```
## S3 method for class 'sjSDM_cv'
plot(x, y, perf = c("logLik", "AUC", "AUC_macro"), resolution = 6, k = 3, ...)
```

Arguments

x	a model fitted by sjSDM_cv
y	unused argument
perf	performance measurement to plot
resolution	resolution of grid
k	number of knots for the gm
...	Additional arguments to pass to plot()

Value

Named vector of optimized regularization parameters.

Without space:

lambda_cov	Regularization strength in the bioticStruct object.
alpha_cov	Weigthing between L1 and L2 in the bioticStruct object.
lambda_coef	Regularization strength in the linear or DNN object.
alpha_coef	Weigthing between L1 and L2 in the linear or DNN object.

With space:

lambda_cov	Regularization strength in the bioticStruct object.
alpha_cov	Weigthing between L1 and L2 in the bioticStruct object.
lambda_coef	Regularization strength in the linear or DNN object.
alpha_coef	Weigthing between L1 and L2 in the linear or DNN object.
lambda_spatial	Regularization strength in the linear or DNN object for the spatial component.
alpha_spatial	Weigthing between L1 and L2 in the linear or DNN object for the spatial component.

plot.sjSDManova	<i>Plot anova results</i>
-----------------	---------------------------

Description

Plot anova results

Usage

```
## S3 method for class 'sjSDManova'
plot(
  x,
  y,
  type = c("McFadden", "Deviance", "Nagelkerke"),
  fractions = c("discard", "proportional", "equal"),
  cols = c("#7FC97F", "#BEAED4", "#FDC086"),
  alpha = 0.15,
  env_deviance = NULL,
  ...
)
```

Arguments

x	anova object from anova.sjSDM
y	unused argument
type	deviance, Nagelkerke or McFadden R-squared
fractions	how to handle shared fractions
cols	colors for the groups
alpha	alpha for colors
env_deviance	environmental deviance
...	Additional arguments to pass to plot()

Value

List with the following components:

VENN	Matrix of shown results.
------	--------------------------

References

Leibold, M. A., Rudolph, F. J., Blanchet, F. G., De Meester, L., Gravel, D., Hartig, F., ... & Chase, J. M. (2022). The internal structure of metacommunities. *Oikos*, 2022(1).

plot.sjSDMimportance *Plot importance*

Description

Plot importance

Usage

```
## S3 method for class 'sjSDMimportance'
plot(x, y, col.points = "#24526e", cex.points = 1.2, ...)
```

Arguments

x	a model fitted by importance
y	unused argument
col.points	point color
cex.points	point size
...	Additional arguments to pass to plot()

Value

The visualized matrix is silently returned.

```
plot.sjSDMinternalStructure
```

Plot internal structure

Description

Creates a ternary diagram of an object of class

Usage

```
## S3 method for class 'sjSDMinternalStructure'
plot(
  x,
  alpha = 0.15,
  env_deviance = NULL,
  negatives = c("floor", "scale", "raw"),
  ...
)
```

Arguments

x	and object of class sjSDMinternalStructure create by anova object from internalStructure
alpha	alpha of points
env_deviance	environmental deviance/gradient (points will be colored)
negatives	how to handle negative R squareds
...	no function

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates

# fit model:
model <- sjSDM(Y = Occ,
  env = linear(data = Env, formula = ~X1+X2+X3),
  spatial = linear(data = SP, formula = ~0+X1*X2),
  family=binomial("probit"),
  verbose = FALSE,
  iter = 20) # increase iter for real analysis
```

```

# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
plotAssemblyEffects(int, negatives = "floor")
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)

```

plotAssemblyEffects *Plot predictors of assembly processes*

Description

The function plots correlations between assembly processes and predictors or traits

Usage

```

plotAssemblyEffects(
  object,
  response = c("sites", "species"),
  pred = NULL,
  cols = c("#A38310", "#B42398", "#20A382"),
  negatives = c("raw", "scale", "floor")
)

```

Arguments

object An sjSDManova object from the [anova.sjSDM](#) function.

response	whether to use sites or species. Default is sites
pred	predictor variable. If NULL, environment uniqueness, spatial uniqueness, and richness is calculated from the fitted object and used as predictor.
cols	Colors for the three assembly processes.
negatives	how to handle negative R squareds

Details

Correlation and plots of the three assembly processes (environment, space, and codist) against environmental and spatial uniqueness and richness. The importance of the three assembly processes is measured by the partial R-squared (shown in the internal structure plots).

Importances are available for species and sites. Custom environmental predictors or traits can be specified. Environmental predictors are plotted against site R-squared and traits are plotted against species R-squared. Regression lines are estimated by 50\

Value

A list with the following components:

env	A list of summary tables for env, space, and codist R-squared.
space	A list of summary tables for env, space, and codist R-squared.
codist	A list of summary tables for env, space, and codist R-squared.

Note

Defaults for negative values are different than for [plot.sjSDMinternalStructure](#)

References

Leibold, M. A., Rudolph, F. J., Blanchet, F. G., De Meester, L., Gravel, D., Hartig, F., ... & Chase, J. M. (2022). The internal structure of metacommunities. *Oikos*, 2022(1).

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates

# fit model:
model <- sjSDM(Y = Occ,
               env = linear(data = Env, formula = ~X1+X2+X3),
               spatial = linear(data = SP, formula = ~0+X1*X2),
               family=binomial("probit"),
```

```

        verbose = FALSE,
        iter = 20) # increase iter for real analysis

# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
plotAssemblyEffects(int, negatives = "floor")
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)

```

plotsjSDMcoef

Internal coefficients plot

Description

Plotting coefficients returned by sjSDM model. This function only for model fitted by linear, fitted by DNN is not yet supported.

Usage

```
plotsjSDMcoef(object, wrap_col = NULL, group = NULL, col = NULL, slist = NULL)
```

Arguments

object	a model fitted by sjSDM
wrap_col	Scales argument passed to wrap_col

group	Define the taxonomic characteristics of a species, you need to provide a dataframe with column1 named “species” and column2 named “group”, default is NULL. For example, group[1,1]== “sp1”, group[1,2]== “Mammal”.
col	Define colors for groups, default is NULL.
slist	Select the species you want to plot, default is all, parameter is not supported yet.

Value

ggplot2 object

Author(s)

CAI Wang

Examples

```
## Not run:
library(sjSDM)
# simulate community:
com = simulate_SDM(env = 6L, species = 7L, sites = 100L)

# fit model:
model = sjSDM(Y = com$response, env = com$env_weights, iter = 2L, se = TRUE,
              verbose = FALSE)

# normal plot
plot(model)

# colored by groups
species=c("sp1", "sp2", "sp3", "sp4", "sp5", "sp6", "sp7")
group=c("mammal", "bird", "fish", "fish", "mammal", "amphibian", "amphibian")
group = data.frame(species=species, group=group)

plot(model, group=group)

## End(Not run)
```

predict.sjSDM

Predict from a fitted sjSDM model

Description

Predict from a fitted sjSDM model

Usage

```
## S3 method for class 'sjSDM'
predict(
  object,
  newdata = NULL,
  SP = NULL,
  Y = NULL,
  type = c("link", "raw"),
  dropout = FALSE,
  marginal = TRUE,
  ...
)
```

Arguments

object	a model fitted by <code>sjSDM</code>
newdata	newdata for predictions
SP	spatial predictors (e.g. X and Y coordinates)
Y	Known occurrences of species, must be a matrix of the original size, species to be predicted must consist of NAs
type	raw or link
dropout	use dropout for predictions or not, only supported for DNNs and NN() blocks
marginal	integrate the latent factor out of the prediction. The species associations are generated by an unobserved latent factor z , so a marginal prediction is $E_z[\text{link}(\mu + z\Sigma')]$ and not $\text{link}(\mu)$, which is the prediction at $z = 0$. The two differ whenever the associations are non-zero: for the probit link the marginal is $\Phi(\mu/\sqrt{\text{diag}(\Sigma\Sigma' + I)})$. The integral is approximated by Monte Carlo, so predictions are stochastic; raise sampling for more stable values. <code>marginal = FALSE</code> restores the behaviour of <code>sjSDM <= 1.1.0</code> and is only there for reproducing older results.
...	optional arguments for compatibility with the generic function, no function implemented

Details

For a model with random-effect bars the prediction for a group the fit saw is taken at the posterior mean of that group. A group it did not see - a level of the grouping factor that was not in the training data, or NA - contributes no mean, and its group effect is integrated over the prior instead. That widens the prediction rather than shifting it, and for the logit and linear links the prior variance is added by sampling, so those predictions carry one more source of Monte-Carlo noise.

Value

Matrix of predictions (sites by species)

Examples

```
## Not run:

## Conditional predictions based on focal species
com = simulate_SDM(sites = 200L)
## first 100 observations are the training data
model = sjSDM(com$response[1:100, ], com$env_weights[1:100,])
## Assume that for the other 100 observations, only the first species is missing
## and we want to use the other 4 species to improve the predictions:
Y_focal = com$response[101:200, ]
Y_focal[,1] = NA # set to NA because occurrences are unknown

pred_conditional = predict(model, newdata = com$env_weights[101:200,], Y = Y_focal)
pred_unconditional = predict(model, newdata = com$env_weights[101:200,])[,1]

## Compare performance:
Metrics::auc(com$response[101:200, 1], pred_conditional)
Metrics::auc(com$response[101:200, 1], pred_unconditional)

## Conditional predictions are better, however, it only works if occurrences of
## other species for new sites are known!

## End(Not run)
```

```
print.bioticStruct      Print a bioticStruct object
```

Description

Print a bioticStruct object

Usage

```
## S3 method for class 'bioticStruct'
print(x, ...)
```

Arguments

<code>x</code>	object created by <code>bioticStruct</code>
<code>...</code>	optional arguments for compatibility with the generic function, no function implemented

print.DNN	<i>Print a DNN object</i>
-----------	---------------------------

Description

Print a DNN object

Usage

```
## S3 method for class 'DNN'  
print(x, ...)
```

Arguments

- x object created by [DNN](#)
- ... optional arguments for compatibility with the generic function, no function implemented

print.linear	<i>Print a linear object</i>
--------------	------------------------------

Description

Print a linear object

Usage

```
## S3 method for class 'linear'  
print(x, ...)
```

Arguments

- x object created by [linear](#)
- ... optional arguments for compatibility with the generic function, no function implemented

Value

Invisible formula object

print.NN	<i>Print an NN object</i>
----------	---------------------------

Description

Print an NN object

Usage

```
## S3 method for class 'NN'  
print(x, ...)
```

Arguments

x	object created by NN
...	optional arguments for compatibility with the generic function, no function implemented

Value

Invisible NN object

print.sjSDM	<i>Print a fitted sjSDM model</i>
-------------	-----------------------------------

Description

Print a fitted sjSDM model

Usage

```
## S3 method for class 'sjSDM'  
print(x, ...)
```

Arguments

x	a model fitted by sjSDM
...	optional arguments for compatibility with the generic function, no function implemented

Value

No return value

print.sjSDM_cv	<i>Print a fitted sjSDM_cv model</i>
----------------	--------------------------------------

Description

Print a fitted sjSDM_cv model

Usage

```
## S3 method for class 'sjSDM_cv'
print(x, ...)
```

Arguments

x	a model fitted by sjSDM_cv
...	optional arguments for compatibility with the generic function, no function implemented

Value

Above data frame is silently returned.

print.sjSDManova	<i>Print sjSDM anova object</i>
------------------	---------------------------------

Description

This is a wrapper for [summary.sjSDManova](#), maintained for backwards compatibility - prefer to use `summary()` instead

Usage

```
## S3 method for class 'sjSDManova'
print(x, ...)
```

Arguments

x	an object of type sjSDManova created by anova.sjSDM
...	additional arguments to summary.sjSDManova

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates

# fit model:
model <- sjSDM(Y = Occ,
               env = linear(data = Env, formula = ~X1+X2+X3),
               spatial = linear(data = SP, formula = ~0+X1*X2),
               family=binomial("probit"),
               verbose = FALSE,
               iter = 20) # increase iter for real analysis

# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
plotAssemblyEffects(int, negatives = "floor")
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)
```

```
print.sjSDMimportance Print importance
```

Description

Print importance

Usage

```
## S3 method for class 'sjSDMimportance'  
print(x, ...)
```

Arguments

x	an object of importance
...	optional arguments for compatibility with the generic function, no function implemented

Value

The matrix above is silently returned

```
print.sjSDMinternalStructure  
  Print internal structure object
```

Description

Print internal structure object

Usage

```
## S3 method for class 'sjSDMinternalStructure'  
print(x, ...)
```

Arguments

x	object of class sjSDMinternalStructure
...	no function

re	<i>Random-effect bar with an explicit rank</i>
----	--

Description

Wraps a random-effect bar so that the rank of its group-level latent factor can be set. `re(x | g, df = 4)` is `(x | g)` with `df` latent dimensions instead of the default.

Usage

```
re(bar, df = NULL, loading = "species")
```

Arguments

bar	a random-effect bar, <code>lhs group</code> , with <code>lme4</code> semantics.
df	number of latent dimensions d of the group effect. Must be at least the number of design columns q the bar produces, otherwise the per-species $q \times q$ covariance $\Lambda_j \Lambda_j'$ is rank deficient and the intercept-slope correlation is forced to ± 1 . Defaults to $\max(q, 5)$.
loading	"species", the default, gives every species its own loading Λ and so its own species covariance for the group effect. "shared" gives the bar one loading vector for all species, i.e. the classical mixed-model row effect in which a group shifts every species by the same amount.

Details

`df` caps the rank of the *cross-species* covariance of the group effect: at `df = 1` every species' group effect is a multiple of one common group score. It does not restrict the per-species covariance, which is unconstrained once `df >= q`.

`loading = "shared"` has one scalar per group rather than a latent vector, so there is no rank to choose and `df` alongside it is an error. The implied species covariance of the group effect is $\lambda^2 \mathbf{1}\mathbf{1}'$, rank one by construction, and with $q > 1$ the bar's columns are likewise perfectly correlated: one group score loaded onto all of them. `getRE` then reports `cov` without the species dimension and `getCor`(which = "re") is not defined for the bar.

The call is read out of the formula by the parser and is never evaluated, so `re()` is syntax rather than a function to call.

Value

A list holding the bar, `df` and `loading`.

See Also

[linear](#), [getRE](#), [sjSDM](#)

residuals.sjSDM	<i>Residuals for a sjSDM model</i>
-----------------	------------------------------------

Description

Returns residuals for a fitted sjSDM model

Usage

```
## S3 method for class 'sjSDM'
residuals(object, type = "raw", ...)
```

Arguments

object	a model fitted by sjSDM
type	residual type. Currently only supports raw
...	further arguments, not supported yet.

Value

residuals in the format of the provided community matrix

RMSprop	<i>RMSprop</i>
---------	----------------

Description

RMSprop optimizer

Usage

```
RMSprop(
  alpha = 0.99,
  eps = 1e-08,
  weight_decay = 1e-04,
  momentum = 0.1,
  centered = FALSE
)
```

Arguments

alpha	decay factor
eps	fuzz factor
weight_decay	l2 penalty on weights
momentum	momentum
centered	centered or not

Value

Anonymous function that returns optimizer when called.

Rsquared	<i>R-squared</i>
----------	------------------

Description

calculate R-squared following McFadden or Nagelkerke

Usage

```
Rsquared(model, method = c("McFadden", "Nagelkerke"), verbose = TRUE)
```

Arguments

model	model
method	McFadden or Nagelkerke
verbose	TRUE or FALSE, indicating whether progress should be printed or not

Details

Calculate R-squared following Nagelkerke or McFadden:

- Nagelkerke: $R^2 = 1 - \exp(2/N \cdot (\log \mathcal{L}_0 - \log \mathcal{L}_1))$
- McFadden: $R^2 = 1 - \log \mathcal{L}_1 / \log \mathcal{L}_0$

For a model with random-effect bars the likelihood is conditional on the posterior group means, so the R-squared is a conditional one: it rises with a bar the data do not need and is not comparable with the R-squared of a model without it. For the gaussian family McFadden's R-squared depends on the measurement scale of the response and is not bounded by 1; use Nagelkerke's.

Value

R-squared as numeric value

Author(s)

Maximilian Pichler

setWeights	<i>Set weights</i>
------------	--------------------

Description

set layer weights and sigma in [sjSDM](#) with [DNN](#) object

Usage

```
setWeights(object, weights)

## S3 method for class 'sjSDM'
setWeights(object, weights = NULL)
```

Arguments

- object object of class [sjSDM](#) with [DNN](#) object
- weights list of layer weights: list(env=list(matrix(...)), spatial=list(matrix(...)), sigma=matrix(...)), see [getWeights](#)

Value

No return value, weights are changed in place.

SGD	<i>SGD</i>
-----	------------

Description

stochastic gradient descent optimizer

Usage

```
SGD(momentum = 0.5, dampening = 0, weight_decay = 0, nesterov = TRUE)
```

Arguments

- momentum strength of momentum
- dampening decay
- weight_decay l2 penalty on weights
- nesterov Nesterov momentum or not

Value

Anonymous function that returns optimizer when called.

simulate.sjSDM	<i>Generates simulations from sjSDM model</i>
----------------	---

Description

Simulate nsim responses from the fitted model following a multivariate probit model. So currently only supported for family = stats::binomial("probit")

Usage

```
## S3 method for class 'sjSDM'
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

object	a model fitted by sjSDM
nsim	number of simulations
seed	seed for random number generator
...	optional arguments for compatibility with the generic function, no functionality implemented

Value

Array of simulated species occurrences of dimension order (nsim, sites, species)

simulate_SDM	<i>Simulate joint Species Distribution Models</i>
--------------	---

Description

Simulate species distributions

Usage

```
simulate_SDM(
  env = 5L,
  sites = 100L,
  species = 5L,
  correlation = TRUE,
  weight_range = c(-1, 1),
  link = "probit",
  response = "pa",
  sparse = NULL,
  tolerance = 0.05,
  iter = 20L,
  seed = NULL
)
```

Arguments

env	number of environment variables
sites	number of sites
species	number of species
correlation	correlated species TRUE or FALSE, can be also a function or a matrix
weight_range	sample true weights from uniform range, default -1,1
link	probit, logit or identical
response	pa (presence-absence) or count
sparse	sparse rate
tolerance	tolerance for sparsity check
iter	tries until sparse rate is achieved
seed	random seed. Default = 42

Details

Probit is not possible for abundance response (response = 'count')

Value

List of simulation results:

env	Number of environmental covariates
species	Number of species
sites	Number of sites
link	Which link
response_type	Which response type
response	Species occurrence matrix
correlation	Species covariance matrix
species_weights	Species-environment coefficients
env_weights	Environmental covariates
corr_acc	Method to calculate sign accuracy

Author(s)

Maximilian Pichler

Description

sjSDM is used to fit joint Species Distribution models (jSDMs) using the central processing unit (CPU) or the graphical processing unit (GPU). The default is a multivariate probit model based on a Monte-Carlo approximation of the joint likelihood. sjSDM can be used to fit linear but also deep neural networks and supports the well known formula syntax. The environmental formula carries structure: it may hold [NN](#) blocks and 'lme4' random-effect bars beside the parametric terms, see the details.

Usage

```
sjSDM(
  Y = NULL,
  env = NULL,
  biotic = bioticStruct(),
  spatial = NULL,
  family = stats::binomial("probit"),
  iter = 100L,
  step_size = NULL,
  learning_rate = 0.01,
  se = FALSE,
  sampling = 100L,
  parallel = 0L,
  control = sjSDMControl(),
  device = "cpu",
  dtype = "float32",
  seed = 758341678,
  verbose = TRUE
)

sjSDM.tune(object)
```

Arguments

Y	matrix of species occurrences/responses in range
env	matrix of environmental predictors, object of type linear or DNN . A linear formula may contain NN blocks and random-effect bars, see details
biotic	defines biotic (species-species associations) structure, object of type bioticStruct
spatial	defines spatial structure, object of type linear or DNN
family	error distribution with link function, see details for supported distributions
iter	number of fitting iterations
step_size	batch size for stochastic gradient descent, if NULL then step_size is set to: <code>step_size = 0.1*nrow(X)</code>

learning_rate	learning rate for the optimizer, see sjSDMControl
se	calculate standard errors for environmental coefficients
sampling	number of sampling steps for Monte Carlo integration
parallel	number of cpu cores for the data loader, only necessary for large datasets
control	control parameters for optimizer, see sjSDMControl
device	which device to be used, "cpu", "gpu", "mps", or the integer index of a CUDA device. A GPU request falls back to the CPU with a warning when no CUDA device is present
dtype	which data type, most GPUs support only 32 bit floats.
seed	seed for random operations
verbose	TRUE or FALSE, indicating whether progress should be printed or not
object	object of type sjSDM_cv

Details

The function fits per default a multivariate probit model via Monte-Carlo integration (see Chen et al., 2018) of the joint likelihood for all species.

Model description:

The most common jSDM structure describes the site ($i = 1, \dots, I$) by species ($j = 1, \dots, J$) matrix Y_{ij} as a function of environmental covariates X_{in} ($n = 1, \dots, N$ covariates), and the species-species covariance matrix Σ accounts for correlations in e_{ij} :

$$g(Z_{ij}) = \beta_{j0} + \sum_{n=1}^N X_{in}\beta_{nj} + e_{ij}$$

with $g(\cdot)$ as link function. For the multivariate probit model, the link function is:

$$Y_{ij} = 1(Z_{ij} > 0)$$

The probability to observe the occurrence vector $\mathbf{Y}_i$ is:

$$Pr(\mathbf{Y}_i | \mathbf{X}_i \beta, \Sigma) = \int_{\mathbf{A}_{iJ}} \dots \int_{\mathbf{A}_{i1}} \phi_J(\mathbf{Y}_i^*; \mathbf{X}_i \beta, \Sigma) d\mathbf{Y}_{i1}^* \dots d\mathbf{Y}_{iJ}^*$$

in the interval A_{ij} with $(-\inf, 0]$ if $Y_{ij} = 0$ and $[0, +\inf)$ if $Y_{ij} = 1$.

and ϕ being the density function of the multivariate normal distribution.

The probability of $\mathbf{Y}_i$ requires to integrate over $\mathbf{Y}_i^*$ which has no closed analytical expression for more than two species which makes the evaluation of the likelihood computationally costly and needs a numerical approximation. The previous equation can be expressed more generally as:

$$\mathcal{L}(\beta, \Sigma; \mathbf{Y}_i, \mathbf{X}_i) = \int_{\Omega} \prod_{j=1}^J Pr(\mathbf{Y}_{ij} | \mathbf{X}_i \beta + \zeta) Pr(\zeta | \Sigma) d\zeta$$

sjSDM approximates this integral by M Monte-Carlo samples from the multivariate normal species-species covariance. After integrating out the covariance term, the remaining part of the likelihood can be calculated as in an univariate case and the average of the M samples are used to get an approximation of the integral:

$$\mathcal{L}(\beta, \Sigma; \mathbf{Y}_i, \mathbf{X}_i) \approx \frac{1}{M} \sum_{m=1}^M \prod_{j=1}^J \Pr(\mathbf{Y}_{ij} | \mathbf{X}_i \beta + \zeta_m)$$

with $\zeta_m \sim MVN(0, \Sigma)$.

sjSDM uses 'torch' (libtorch) and can optionally run the model on the graphical processing unit (GPU), see [installation_help](#) for how to enable that.

See Pichler and Hartig, 2020 for benchmark results.

Supported distributions:

Currently supported distributions and link functions, which are :

- [binomial](#): "probit" or "logit"
- [poisson](#): "log"
- "nbinom": "log"
- [gaussian](#): "identity"

Space:

We can extend the model to account for spatial auto-correlation between the sites by:

$$g(Z_{ij}) = \beta_{j0} + \sum_{n=1}^N X_{in} \beta_{nj} + \sum_{m=1}^M S_{im} \alpha_{mj} + e_{ij}$$

There are two ways to generate spatial predictors S :

- trend surface model - using spatial coordinates in a polynomial:
`linear(data=Coords, ~0+poly(X, Y, degree = 2))`
- eigenvector spatial filtering - using spatial eigenvectors. Spatial eigenvectors can be generated by the [generateSpatialEV](#) function:
`SPV = generateSpatialEV(Coords)`
Then we use, for example, the first 20 spatial eigenvectors:
`linear(data=SPV[, 1:20], ~0+.)`

It is important to set the intercept to 0 in the spatial term (e.g. via `~0+.`) because the intercept is already set in the environmental object.

Structure in the environmental formula:

A [linear](#) formula is not restricted to parametric terms. It may in addition hold [NN](#) blocks and random-effect bars:

`linear(X, ~ temp + NN(soil + ndvi, hidden = c(50, 50)) + (temp | plot))`

Every block is summed on the scale of the linear predictor,

$$g(Z_{ij}) = \mathbf{X}_i \beta_j + \mathbf{f}(\mathbf{X}_i^{\text{NN}})_j + \sum_b \sum_k \mathbf{Z}_{ik}^b \mathbf{u}_{g(i)}^b \Lambda_{jk}^b + \mathbf{e}_{ij}$$

so the parametric part keeps its meaning: [coef.sjSDM](#), [summary.sjSDM](#) and the standard errors describe β and nothing else.

[NN\(soil + ndvi\)](#) is a feed-forward network from its own covariates to the species. It sits beside the interpretable terms rather than replacing them, which is the difference to [DNN](#). The block itself has no interpretable coefficients, and the standard errors are conditional on the fitted block: they ignore its uncertainty, and a warning says so. [update.sjSDM](#) with `env_blocks` refits with a subset of the blocks, which is how a block is switched off.

A bar is written in 'lme4' syntax and carries 'lme4' semantics, through the 'reformulas' package: (1 | plot), (temp | plot) for a correlated intercept and slope, (0 + temp | plot), (temp || plot) and (1 | plot/subplot). [re](#) is the explicit form where a bar needs arguments, `re(1 | plot, df = 2)`.

A bar is a design-level latent factor, not a scalar shift. One group effect $\mathbf{u}_g \sim N(\mathbf{0}, \mathbf{I}_d)$ is drawn per group and loaded onto the species by Λ , so the group effect has its own species-species covariance $\Lambda\Lambda'$ of rank df and species may respond to a plot with different magnitude and sign. df defaults to $\max(q, 5)$, with q the number of columns of the bar, and must not be smaller than q : below that the intercept-slope correlation is forced to ± 1 . `re(loading = "shared")` gives the classical mixed-model case instead, one shift per group identical for every species.

The bars are integrated out by a stochastic variational approximation. Each group gets a Gaussian posterior, drawn with the reparameterisation trick, and the Kullback-Leibler term to the $N(0, \mathbf{I})$ prior supplies the shrinkage; the Monte-Carlo likelihood itself is unchanged. `getRE` reports the group effects, `getCov` and `getCor` with which = "re" the covariance the bar induces.

Limitations of blocks and bars:

- `anova.sjSDM` refuses on a model with a bar. The random block is re-estimated in every refit, so it absorbs the fraction that was removed instead of cancelling from the difference, and the per-species rescaling is recomputed with it. Use `getRE` for the group variances.
- `logLik.sjSDM` for a model with a bar is the conditional value at the posterior means, neither the evidence lower bound nor a marginal likelihood. It is not comparable with the log-likelihood of a model fitted without the bar, and neither are information criteria built from it.
- `sjSDM_cv` supports neither blocks nor bars. It raises an error rather than dropping them.
- Neither is available in the spatial formula.
- The reported group variance grows with df , because the Kullback-Leibler term constrains the posterior and not the loadings. Report the rank alongside the variance, and prefer a small df .

Installation:

`sjSDM` needs the 'torch' binaries, which `install_sjSDM` (a wrapper around `torch::install_torch`) downloads. If `sjSDM` still does not work after restarting R, follow the trouble shooting guide [installation_help](#). If the problem remains, please create an issue on [issue tracker](#) with a copy of the `install_diagnostic` output as a quote.

Value

An S3 class of type 'sjSDM' including the following components:

<code>cl</code>	Model call
<code>formula</code>	Formula object for environmental covariates.
<code>names</code>	Names of environmental covariates.
<code>species</code>	Names of species (can be NULL if columns of Y are not named).
<code>get_model</code>	Method which builds and returns the underlying 'torch' model.
<code>logLik</code>	negative log-Likelihood of the model and the regularization loss.
<code>model</code>	The actual model.

settings	List of model settings, see arguments of sjSDM .
family	Response family.
time	Runtime.
data	List of Y, X (and spatial) model matrices.
sessionInfo	Output of sessionInfo .
weights	List of model coefficients (environmental (and spatial)).
sigma	Lower triangular weight matrix for the covariance matrix.
history	History of iteration losses.
se	Matrix of standard errors, if se = FALSE the field 'se' is NULL.

Implemented S3 methods include [summary.sjSDM](#), [plot.sjSDM](#), [print.sjSDM](#), [predict.sjSDM](#), and [coef.sjSDM](#). For other methods, see section 'See Also'.

[sjSDM.tune](#) returns an S3 object of class 'sjSDM', see above for information about values.

Author(s)

Maximilian Pichler

References

Chen, D., Xue, Y., & Gomes, C. P. (2018). End-to-end learning for the deep multivariate probit model. arXiv preprint arXiv:1803.08591.

Pichler, M., & Hartig, F. (2021). A new joint species distribution model for faster and more accurate inference of species associations from big community data. *Methods in Ecology and Evolution*, 12(11), 2159-2173.

See Also

[getCor](#), [getCov](#), [getRE](#), [update.sjSDM](#), [sjSDM_cv](#), [DNN](#), [NN](#), [re](#), [plot.sjSDM](#), [print.sjSDM](#), [predict.sjSDM](#), [coef.sjSDM](#), [summary.sjSDM](#), [simulate.sjSDM](#), [getSe](#), [anova.sjSDM](#), [importance](#)

Examples

```
## Not run:

# Basic workflow:
## simulate community:
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)

## fit model:
model = sjSDM(Y = com$response, env = com$env_weights, iter = 50L,
              verbose = FALSE)
# increase iter for your own data

# Default distribution is binomial("probit"). Alternatively, you can use
# binomial(logit), poisson("log"), "nbinom" (with log, still somewhat
# experimental) and gaussian("identity")
```

```

coef(model)
summary(model)
getCov(model)

## plot results
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)
plot(model,group=group)

## calculate post-hoc p-values:
p = getSe(model)
summary(p)

## or turn on the option in the sjSDM function:
model = sjSDM(Y = com$response, env = com$env_weights, se = TRUE,
              family = binomial("probit"),
              iter = 2L,
              verbose = FALSE)
summary(model)

## fit model with interactions:
model = sjSDM(Y = com$response,
              env = linear(data = com$env_weights, formula = ~X1:X2 + X3),
              se = TRUE,
              iter = 2L,
              verbose = FALSE) # increase iter for your own data
summary(model)

## without intercept:
model = update(model, env_formula = ~0+X1:X2 + X3,
              verbose = FALSE)

summary(model)

## predict with model:
preds = predict(model, newdata = com$env_weights)

## calculate R-squared:
R2 = Rsquared(model)
print(R2)

# With spatial terms:
## linear spatial model
XY = matrix(rnorm(200), 100, 2)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(XY, ~0+X1:X2),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = XY)
R2 = Rsquared(model)
print(R2)

```

```

## Using spatial eigenvectors as predictors to account
## for spatial autocorrelation is a common approach:
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+., lambda = 0.1),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model,
            verbose = FALSE)

internal = internalStructure(an)
plot(internal)

## Visualize community assembly effects

plotAssemblyEffects(internal)

### see ?anova.sjSDM for more details

## non-linear(deep neural network) model
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = DNN(SPV,hidden = c(5L, 5L), ~0+.),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

# Regularization
## lambda is the regularization strength
## alpha weights the lasso or ridge penalty:
## - alpha = 0 --> pure lasso
## - alpha = 1.0 --> pure ridge
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = linear(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
coef(model)
getCov(model)

# Anova

```

```

com = simulate_SDM(env = 3L, species = 15L, sites = 200L, correlation = TRUE)

XY = matrix(rnorm(400), 200, 2)
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+.),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
result = anova(model, verbose = FALSE)
print(result)
plot(result)

## visualize internal meta-community structure
internal = internalStructure(an)
plot(internal)

# Deep neural networks
## we can fit also a deep neural network instead of a linear model:
model = sjSDM(Y = com$response,
              env = DNN(com$env_weights, hidden = c(10L, 10L, 10L)),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
summary(model)
getCov(model)
pred = predict(model, newdata = com$env_weights)

## extract weights
weights = getWeights(model)

## we can also assign weights:
setWeights(model, weights)

## with regularization:
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = DNN(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
getCov(model)
getWeights(model)

## End(Not run)

```

Description

Cross validation of elastic net tuning

Usage

```
sjSDM_cv(
  Y,
  env = NULL,
  biotic = bioticStruct(),
  spatial = NULL,
  tune = c("random", "grid"),
  CV = 5L,
  tune_steps = 20L,
  alpha_cov = seq(0, 1, 0.1),
  alpha_coef = seq(0, 1, 0.1),
  alpha_spatial = seq(0, 1, 0.1),
  lambda_cov = 2^seq(-10, -1, length.out = 20),
  lambda_coef = 2^seq(-10, -0.5, length.out = 20),
  lambda_spatial = 2^seq(-10, -0.5, length.out = 20),
  device = "cpu",
  n_cores = NULL,
  n_gpu = NULL,
  sampling = 5000L,
  blocks = 1L,
  ...
)
```

Arguments

Y	species occurrence matrix
env	matrix of environmental predictors or object of type linear , or DNN
biotic	defines biotic (species-species associations) structure, object of type bioticStruct . Alpha and lambda have no influence
spatial	defines spatial structure, object of type linear , or DNN
tune	tuning strategy, random or grid search
CV	n-fold cross validation or list of test indices
tune_steps	number of tuning steps
alpha_cov	weighting of l1 and l2 on covariances: $(1 - \alpha) * cov + \alpha cov ^2$
alpha_coef	weighting of l1 and l2 on coefficients: $(1 - \alpha) * coef + \alpha coef ^2$
alpha_spatial	weighting of l1 and l2 on spatial coefficients: $(1 - \alpha) * coef_{sp} + \alpha coef_{sp} ^2$
lambda_cov	overall regularization strength on covariances
lambda_coef	overall regularization strength on coefficients
lambda_spatial	overall regularization strength on spatial coefficients
device	device, default cpu

n_cores	number of cores for parallelization
n_gpu	number of GPUs
sampling	number of sampling steps for Monte Carlo integration
blocks	blocks of parallel tuning steps
...	arguments passed to sjSDM, see sjSDM

Details

Each fold refits on the already expanded design matrix, which the [NN](#) blocks and the random-effect bars of an environmental formula do not survive. A model with either raises an error on the resulting row-count mismatch rather than silently dropping them, so cross-validation of such a model is not available.

Value

An S3 class of type 'sjSDM_cv' including the following components:

tune_results	Data frame with tuning results.
short_summary	Data frame with averaged tuning results.
summary	Data frame with summarized averaged results.
settings	List of tuning settings, see the arguments in DNN .
data	List of Y, env (and spatial) objects.
config	List of sjSDM settings, see arguments of sjSDM .
spatial	Logical, spatial model or not.

Implemented S3 methods include [sjSDM.tune](#), [plot.sjSDM_cv](#), [print.sjSDM_cv](#), and [summary.sjSDM_cv](#)

See Also

[plot.sjSDM_cv](#), [print.sjSDM_cv](#), [summary.sjSDM_cv](#), [sjSDM.tune](#)

Examples

```
## Not run:
# simulate sparse community:
com = simulate_SDM(env = 5L, species = 25L, sites = 50L, sparse = 0.5)

# tune regularization:
tune_results = sjSDM_cv(Y = com$response,
                        env = com$env_weights,
                        tune = "random", # random steps in tune-parameter space
                        CV = 2L, # 3-fold cross validation
                        tune_steps = 2L,
                        alpha_cov = seq(0, 1, 0.1),
                        alpha_coef = seq(0, 1, 0.1),
                        lambda_cov = seq(0, 0.1, 0.001),
                        lambda_coef = seq(0, 0.1, 0.001),
                        n_cores = 2L,
```

```

        sampling = 100L,
        # small models can be also run in parallel on the GPU
        iter = 2L # we can pass arguments to sjSDM via...
    )

# print overall results:
tune_results

# summary (mean values over CV for each tuning step)
summary(tune_results)

# visualize tuning and best points:
# best = plot(tune_results, perf = "logLik")

# fit model with best regularization paramter:
model = sjSDM.tune(tune_results)

summary(model)

## End(Not run)

```

sjSDMControl

sjSDM control object

Description

sjSDM control object

Usage

```

sjSDMControl(
  optimizer = Adam(),
  scheduler = 0,
  lr_reduce_factor = 0.99,
  early_stopping_training = 0,
  mixed = FALSE
)

```

Arguments

optimizer	object of type Adam (the default), AdamW , Adagrad , RMSprop , or SGD . Adamax , AccSGD , madgrad and AdaBound have no 'torch' counterpart, redirect to one of the above with a message, and will be removed in 1.2.0
scheduler	reduce lr on plateau scheduler or not (0 means no scheduler, > 0 number of epochs before reducing learning rate)
lr_reduce_factor	factor to reduce learning rate in scheduler

early_stopping_training number of epochs without decrease in training loss before invoking early stopping (0 means no early stopping).

mixed mixed (half-precision) training. Accepted and ignored, 'torch' has no equivalent

Value

List with the following fields:

optimizer Function which returns an optimizer.

scheduler_boolean Logical, use scheduler or not.

scheduler_patience Integer, number of epochs to wait before applying plateau scheduler.

lr_reduce_factor Numerical, learning rate reduce factor.

mixed Logical, use mixed training or not.

early_stopping_training Numerical, early stopping after n epochs.

summary.sjSDM	<i>Return summary of a fitted sjSDM model</i>
---------------	---

Description

Return summary of a fitted sjSDM model

Usage

```
## S3 method for class 'sjSDM'
summary(object, ...)
```

Arguments

object a model fitted by [sjSDM](#)

... optional arguments for compatibility with the generic function, no functionality implemented

Value

The above matrix is silently returned.

summary.sjSDM_cv	<i>Return summary of a fitted sjSDM_cv model</i>
------------------	--

Description

Return summary of a fitted sjSDM_cv model

Usage

```
## S3 method for class 'sjSDM_cv'
summary(object, ...)
```

Arguments

- object a model fitted by [sjSDM_cv](#)
- ... optional arguments for compatibility with the generic function, no functionality implemented

Value

Above data frame is silently returned.

summary.sjSDManova	<i>Summary table of sjSDM anova</i>
--------------------	-------------------------------------

Description

The function prints and returns invisible a summary table of an sjSDM ANOVA, created by [anova.sjSDM](#)

Usage

```
## S3 method for class 'sjSDManova'
summary(
  object,
  method = c("ANOVA"),
  fractions = c("mvp_proportional", "mvp", "all", "discard", "proportional", "equal"),
  ...
)
```

Arguments

- object an object of [anova.sjSDM](#)
- method method used to calculate the ANOVA
- fractions how to handle the shared fractions. See details
- ... optional arguments for compatibility with the generic function, no function implemented

Details

The function returns a ANOVA table with Deviance as well as the pseudo-R2 metrics of Nagelkerke and McFadden

There are six options to handle shared ANOVA fractions, which is variance that can be explained, typically as a result of collinearity, by several of the fractions:

1. "mvp_proportional" (the default) splits Abiotic and Spatial within the model without associations, distributes their shared part proportionally, and gives the remainder of the full model to the associations. Without a spatial component it is the same as "proportional"
2. "mvp" is the same split without distributing the shared Abiotic and Spatial part. Without a spatial component it is the same as "proportional"
3. "all" returns the shared fractions explicitly
4. "discard" discards the fractions, as typically in a type II Anova
5. "proportional" distributes shared fractions proportional to the unique fractions, weighted by their absolute size, so a negative unique fraction still attracts shared variance
6. "equal" distributes shared fractions equally to the unique fractions

Value

The matrix that is printed out is silently returned

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates

# fit model:
model <- sjSDM(Y = Occ,
               env = linear(data = Env, formula = ~X1+X2+X3),
               spatial = linear(data = SP, formula = ~0+X1*X2),
               family=binomial("probit"),
               verbose = FALSE,
               iter = 20) # increase iter for real analysis

# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
```

```

summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
plotAssemblyEffects(int, negatives = "floor")
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)

```

update.sjSDM

Update and re-fit a model call

Description

Update and re-fit a model call

Usage

```

## S3 method for class 'sjSDM'
update(
  object,
  env_formula = NULL,
  spatial_formula = NULL,
  biotic = NULL,
  env_blocks = NULL,
  ...
)

```

Arguments

object	of class 'sjSDM'
env_formula	new environmental formula
spatial_formula	new spatial formula
biotic	new biotic config

<code>env_blocks</code>	indices of the environmental blocks to keep in the predictor, 1 being the parametric block and the <code>NN</code> blocks following it in formula order. Every other block is removed from the sum and contributes exactly zero. Random-effect bars are not selected here, they are kept in every refit.
<code>...</code>	additional arguments

Value

An S3 class of type 'sjSDM'. See [sjSDM](#) for more information.

Index

* datasets

- butterflies, [15](#)
- eucalypts, [22](#)

AccSGD, [4](#), [76](#)
AdaBound, [4](#), [76](#)
Adagrad, [5](#), [76](#)
Adam, [4](#), [6](#), [6](#), [41](#), [76](#)
Adamax, [6](#), [76](#)
AdamW, [7](#), [76](#)
anova.sjSDM, [8](#), [29](#), [33](#), [47](#), [49](#), [57](#), [69](#), [70](#), [78](#)

binomial, [68](#)
bioticStruct, [11](#), [46](#), [54](#), [66](#), [74](#)
butterflies, [15](#)

check_module, [16](#)
checkModel, [16](#)
coef.sjSDM, [17](#), [68](#), [70](#)

DiffGrad, [17](#)
DNN, [18](#), [24](#), [25](#), [28](#), [36](#), [43](#), [44](#), [46](#), [55](#), [63](#), [66](#), [68](#), [70](#), [74](#), [75](#)

eucalypts, [22](#)

gaussian, [68](#)
generateSpatialEV, [23](#), [68](#)
getCor, [24](#), [25](#), [60](#), [69](#), [70](#)
getCov, [24](#), [25](#), [27](#), [69](#), [70](#)
getImportance, [26](#)
getRE, [9](#), [24](#), [25](#), [26](#), [60](#), [69](#), [70](#)
getSe, [27](#), [70](#)
getWeights, [28](#), [63](#)

importance, [29](#), [47](#), [59](#), [70](#)
install_diagnostic, [30](#), [31](#), [32](#), [69](#)
install_sjSDM, [30](#), [31](#), [69](#)
install_torch, [31](#)
installation_help, [30](#), [31](#), [31](#), [68](#), [69](#)
internalStructure, [33](#), [48](#)

is_torch_available, [35](#)

linear, [18](#), [19](#), [35](#), [43](#), [46](#), [55](#), [60](#), [66](#), [68](#), [74](#)
logLik.sjSDM, [40](#), [69](#)

madgrad, [41](#), [76](#)

new_image, [41](#)
NN, [18](#), [19](#), [36](#), [42](#), [56](#), [66](#), [68](#), [70](#), [75](#), [81](#)

plot.sjSDM, [43](#), [70](#)
plot.sjSDM.DNN, [44](#)
plot.sjSDM_cv, [45](#), [75](#)
plot.sjSDManova, [8](#), [10](#), [46](#)
plot.sjSDMimportance, [29](#), [30](#), [47](#)
plot.sjSDMinternalStructure, [10](#), [34](#), [48](#), [50](#)
plotAssemblyEffects, [49](#)
plotsjSDMcoef, [43](#), [44](#), [51](#)
poisson, [68](#)
predict.sjSDM, [52](#), [70](#)
print.bioticStruct, [12](#), [54](#)
print.DNN, [19](#), [55](#)
print.linear, [36](#), [55](#)
print.NN, [56](#)
print.sjSDM, [56](#), [70](#)
print.sjSDM_cv, [57](#), [75](#)
print.sjSDManova, [10](#), [57](#)
print.sjSDMimportance, [29](#), [30](#), [58](#)
print.sjSDMinternalStructure, [34](#), [59](#)

re, [24](#), [25](#), [27](#), [36](#), [60](#), [69](#), [70](#)
residuals.sjSDM, [61](#)
RMSprop, [61](#), [76](#)
Rsquared, [62](#)

sessionInfo, [70](#)
setWeights, [28](#), [63](#)
SGD, [4](#), [63](#), [76](#)
simulate.sjSDM, [64](#), [70](#)
simulate_SDM, [64](#)

sjSDM, [8](#), [12](#), [17](#), [19](#), [24–29](#), [32](#), [36](#), [40](#), [43](#), [44](#),
[51](#), [53](#), [56](#), [60](#), [61](#), [63](#), [64](#), [66](#), [69](#), [70](#),
[75](#), [77](#), [81](#)
sjSDM-package (installation_help), [31](#)
sjSDM.tune, [70](#), [75](#)
sjSDM_cv, [36](#), [45](#), [57](#), [67](#), [69](#), [70](#), [73](#), [78](#)
sjSDMControl, [67](#), [76](#)
summary.sjSDM, [68](#), [70](#), [77](#)
summary.sjSDM_cv, [75](#), [78](#)
summary.sjSDManova, [10](#), [33](#), [57](#), [78](#)
update.sjSDM, [68](#), [70](#), [80](#)