# Package 'PLNmodels'

March 21, 2025

**Title** Poisson Lognormal Models

**Version** 1.2.2

**Description** The Poisson-lognormal model and variants (Chiquet, Mariadassou and Robin, 2021 <doi:10.3389/fevo.2021.588292>) can be used for a variety of multivariate problems when count data are at play, including principal component analysis for count data, discriminant analysis, model-based clustering and network inference. Implements variational algorithms to fit such models accompanied with a set of functions for visualization and diagnostic.

**URL** `https://pln-team.github.io/PLNmodels/`

**BugReports** `https://github.com/pln-team/PLNmodels/issues`

**License** GPL (>= 3)

**Depends** R (>= 4.1.0)

**Imports** cli, corrplot, dplyr, future, future.apply, ggplot2, glassoFast, grDevices, grid, gridExtra, igraph, magrittr, MASS, Matrix, methods, nloptr, parallel, pscl, purrr, R6, Rcpp, rlang, scales, stats, tidyr, torch

**Suggests** factoextra, knitr, rmarkdown, spelling, testthat

**LinkingTo** nloptr, Rcpp, RcppArmadillo

**VignetteBuilder** knitr

**Encoding** UTF-8

**Language** en-US

**LazyData** true

**RoxygenNote** 7.3.2

**Collate** 'PLNfit-class.R' 'PLN.R' 'PLNLDA.R' 'PLNLDAfit-S3methods.R' 'PLNLDAfit-class.R' 'PLNPCA.R' 'PLNPCAfamily-S3methods.R' 'PLNfamily-class.R' 'PLNPCAfamily-class.R' 'PLNPCAfit-S3methods.R' 'PLNPCAfit-class.R' 'PLNfamily-S3methods.R' 'PLNfit-S3methods.R' 'PLNmixture.R' 'PLNmixturefamily-S3methods.R' 'PLNmixturefamily-class.R' 'PLNmixturefit-S3methods.R' 'PLNmixturefit-class.R'

'PLNmodels-package.R' 'PLNnetwork.R'
'PLNnetworkfamily-S3methods.R' 'PLNnetworkfamily-class.R'
'PLNnetworkfit-S3methods.R' 'PLNnetworkfit-class.R'
'RcppExports.R' 'ZIPLNfit-class.R' 'ZIPLN.R'
'ZIPLNfit-S3methods.R' 'ZIPLNnetwork.R' 'barents.R'
'import_utils.R' 'mollusk.R' 'oaks.R' 'plot_utils.R' 'scRNA.R'
'trichoptera.R' 'utils-pipe.R' 'utils-zipln.R' 'utils.R'
'zzz.R'

**NeedsCompilation** yes

**Author** Julien Chiquet [aut, cre] (<`https://orcid.org/0000-0002-3629-3429`>),
Mahendra Mariadassou [aut] (<`https://orcid.org/0000-0003-2986-354X`>),
Stéphane Robin [aut],
François Gindraud [aut],
Julie Aubert [ctb],
Bastien Batardière [ctb],
Giovanni Poggiato [ctb],
Cole Trapnell [ctb],
Maddy Duran [ctb]

**Maintainer** Julien Chiquet <`julien.chiquet@inrae.fr`>

**Repository** CRAN

**Date/Publication** 2025-03-21 17:40:06 UTC

# Contents

---

## `barents` *Barents fish data set*

### Description

This data set gives the abundance of 30 fish species observed in 89 sites in the Barents sea. For each site, 4 additional covariates are known. Subsample of the original datasets studied by Fossheim et al, 2006.

### Usage

```
barents
```

### Format

A data frame with 6 variables:

- Abundance: A 30 fish species by 89 sites count matrix
- Offset: A 30 fish species by 89 samples offset matrix, measuring the sampling effort in each site
- 4 covariates for latitude, longitude, depth (in meters), temperature (in Celsius degrees).

### Source

Data from M. Fossheim and coauthors.

### References

Fossheim, Maria, Einar M. Nilssen, and Michaela Aschan. "Fish assemblages in the Barents Sea." Marine Biology Research 2.4 (2006). doi:10.1080/17451000600815698

### Examples

```
data(barents)
```

---

## `coef.PLNfit` *Extract model coefficients*

---

### Description

Extracts model coefficients from objects returned by `PLN()` and its variants

### Usage

```
## S3 method for class 'PLNfit'
coef(object, type = c("main", "covariance"), ...)
```

### Arguments

| | |
|---|---|
| `object` | an R6 object with class `PLNfit` |
| `type` | type of parameter that should be extracted. Either "main" (default) for $$B$$ or "covariance" for $$\Sigma$$ |
| `...` | additional parameters for S3 compatibility. Not used |

### Value

A matrix of coefficients extracted from the PLNfit model.

### See Also

`sigma.PLNfit()`, `vcov.PLNfit()`, `standard_error.PLNfit()`

### Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLN(Abundance ~ 1 + offset(log(Offset)), data = trichoptera)
coef(myPLN) ## B
coef(myPLN, type = "covariance") ## Sigma
```

---

## `coef.PLNLDAfit` *Extracts model coefficients from objects returned by* `PLNLDA()`

### Description

The method for objects returned by `PLNLDA()` only returns coefficients associated to the

$$\Theta$$

part of the model (see the PLNLDA vignette for mathematical details).

### Usage

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

### Arguments

| | |
|---|---|
| `object` | an R6 object with class PLNLDAfit |
| `...` | additional parameters for S3 compatibility. Not used |

### Value

Either NULL or a matrix of coefficients extracted from the PLNLDAfit model.

### Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLNLDA <- PLNLDA(Abundance ~ Wind, grouping = Group, data = trichoptera)
coef(myPLNLDA)
```

---

## `coef.PLNmixturefit` *Extract model coefficients*

### Description

Extracts model coefficients from objects returned by `PLN()` and its variants

### Usage

```
## S3 method for class 'PLNmixturefit'
coef(object, type = c("main", "means", "covariance", "mixture"), ...)
```

### Arguments

| | |
|---|---|
| `object` | an R6 object with class `PLNmixturefit` |
| `type` | type of parameter that should be extracted. Either "main" (default) for<br>$$\Theta$$<br>, "means" for<br>$$\mu$$<br>, "mixture" for<br>$$\pi$$<br>or "covariance" for<br>$$\Sigma$$ |
| `...` | additional parameters for S3 compatibility. Not used |

### Value

A matrix of coefficients extracted from the PLNfit model.

### See Also

`sigma.PLNmixturefit()`

### Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLNmixture(Abundance ~ 1 + offset(log(Offset)),
       data = trichoptera, control = PLNmixture_param(smoothing = "none")) %>% getBestModel()
coef(myPLN) ## Theta - empty here
coef(myPLN, type = "mixture") ## pi
coef(myPLN, type = "means") ## mu
coef(myPLN, type = "covariance") ## Sigma
```

---

## `coef.ZIPLNfit` *Extract model coefficients*

### Description

Extracts model coefficients from objects returned by `ZIPLN()` and its variants

### Usage

```
## S3 method for class 'ZIPLNfit'
coef(object, type = c("count", "zero", "precision", "covariance"), ...)
```

### Arguments

| | |
|---|---|
| `object` | an R6 object with class `ZIPLNfit` |
| `type` | type of parameter that should be extracted. Either "count" (default) for $B$, "zero" for $B0$, "precision" for $\Omega$, "covariance" for $\Sigma$ |
| `...` | additional parameters for S3 compatibility. Not used |

### Value

A matrix of coefficients extracted from the ZIPLNfit model.

### See Also

`sigma.ZIPLNfit()`

### Examples

```
data(scRNA)
# data subsample: only 100 random cell and the 50 most varying transcript
subset <- sample.int(nrow(scRNA), 100)
myPLN  <- ZIPLN(counts[, 1:50] ~ 1 + offset(log(total_counts)), subset = subset, data = scRNA)
```

---

## `coefficient_path` *Extract the regularization path of a PLNnetwork fit*

### Description

Extract the regularization path of a PLNnetwork fit

### Usage

```
coefficient_path(Robject, precision = TRUE, corr = TRUE)
```

### Arguments

| | |
|---|---|
| `Robject` | an object with class `Networkfamily`, i.e. an output from `PLNnetwork()` |
| `precision` | a logical, should the coefficients of the precision matrix Omega or the covariance matrix Sigma be sent back. Default is `TRUE`. |
| `corr` | a logical, should the correlation (partial in case `precision = TRUE`) be sent back. Default is `TRUE`. |

### Value

Sends back a tibble/data.frame.

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
fits <- PLNnetwork(Abundance ~ 1, data = trichoptera)
head(coefficient_path(fits))
```

---

# `compute_offset` *Compute offsets from a count data using one of several normalization schemes*

---

## Description

Computes offsets from the count table using one of several normalization schemes (TSS, CSS, RLE, GMPR, Wrench, TMM, etc) described in the literature.

## Usage

```
compute_offset(
  counts,
  offset = c("TSS", "GMPR", "RLE", "CSS", "Wrench", "TMM", "none"),
  scale = c("none", "count"),
  ...
)
```

## Arguments

| | |
|---|---|
| `counts` | Required. An abundance count table, preferably with dimensions names and species as columns. |
| `offset` | Optional. Normalization scheme used to compute scaling factors used as offset during PLN inference. Available schemes are "TSS" (Total Sum Scaling, default), "CSS" (Cumulative Sum Scaling, used in metagenomeSeq), "RLE" (Relative Log Expression, used in DESeq2), "GMPR" (Geometric Mean of Pairwise Ratio, introduced in Chen et al., 2018), Wrench (introduced in Kumar et al., 2018) or "none". Alternatively the user can supply its own vector or matrix of offsets (see note for specification of the user-supplied offsets). |
| `scale` | Either `"none"` (default) or `"count"`. Should the offset be normalized to be on the same scale as the counts ? |
| `...` | Additional parameters passed on to specific methods (for now CSS and RLE) |

## Details

RLE has additional `pseudocounts` and `type` arguments to add pseudocounts to the observed counts (defaults to 0L) and to compute offsets using only positive counts (if `type == "poscounts"`). This mimics the behavior of `DESeq2::DESeq()` when using `sfType == "poscounts"`. CSS has an additional `reference` argument to choose the location function used to compute the reference quantiles (defaults to `median` as in the Nature publication but can be set to `mean` to reproduce behavior of functions cumNormStat* from metagenomeSeq). Wrench has two additional parameters: `groups` to specify sample groups and `type` to either reproduce exactly the default `Wrench::wrench()` behavior (`type = "wrench"`, default) or to use simpler heuristics (`type = "simple"`). Note that (i) CSS normalization fails when the median absolute deviation around quantiles does not become instable for high quantiles (limited count variations both within and across samples) and/or one sample has less than two positive counts, (ii) RLE fails when there are no common species across all samples (unless `type == "poscounts"` has been specified) and (iii) GMPR fails if a sample does not share any species with all other samples. TMM code between two libraries is simplified and adapted from M. Robinson (edgeR:::.calcFactorTMM). The final output is however different from the one produced by edgeR:::.calcFactorTMM as they are intended to be used as such in the model (whereas they need to be multiplied by sequencing depths in edgeR)

## Value

If `offset = "none"`, `NULL` else a vector of length `nrow(counts)` with one offset per sample.

## References

Chen, L., Reeve, J., Zhang, L., Huang, S., Wang, X. and Chen, J. (2018) GMPR: A robust normalization method for zero-inflated count data with application to microbiome sequencing data. PeerJ, 6, e4600 doi:10.7717/peerj.4600

Paulson, J. N., Colin Stine, O., Bravo, H. C. and Pop, M. (2013) Differential abundance analysis for microbial marker-gene surveys. Nature Methods, 10, 1200-1202 doi:10.1038/nmeth.2658

Anders, S. and Huber, W. (2010) Differential expression analysis for sequence count data. Genome Biology, 11, R106 doi:10.1186/gb20101110r106

Kumar, M., Slud, E., Okrah, K. et al. (2018) Analysis and correction of compositional bias in sparse sequencing count data. BMC Genomics 19, 799 doi:10.1186/s1286401851605

Robinson, M.D., Oshlack, A. (2010) A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biol 11, R25 doi:10.1186/gb2010113r25

## Examples

```
data(trichoptera)
counts <- trichoptera$Abundance
compute_offset(counts)
## Other normalization schemes
compute_offset(counts, offset = "RLE", pseudocounts = 1)
compute_offset(counts, offset = "Wrench", groups = trichoptera$Covariate$Group)
compute_offset(counts, offset = "GMPR")
compute_offset(counts, offset = "TMM")
## User supplied offsets
my_offset <- setNames(rep(1, nrow(counts)), rownames(counts))
compute_offset(counts, offset = my_offset)
```

---

# `compute_PLN_starting_point` *Helper function for PLN initialization.*

---

## Description

Barebone function to compute starting points for B, M and S when fitting a PLN. Mostly intended for internal use.

## Usage

```
compute_PLN_starting_point(Y, X, O, w, s = 0.1)
```

## Arguments

| | |
|---|---|
| `Y` | Response count matrix |
| `X` | Covariate matrix |
| `O` | Offset matrix (in log-scale) |
| `w` | Weight vector (defaults to 1) |
| `s` | Scale parameter for S (defaults to 0.1) |

## Details

The default strategy to estimate B and M is to fit a linear model with covariates `X` to the response count matrix (after adding a pseudocount of 1, scaling by the offset and taking the log). The regression matrix is used to initialize `B` and the residuals to initialize `M`. `S` is initialized as a constant conformable matrix with value `s`.

## Value

a named list of starting values for model parameter B and variational parameters M and S used in the iterative optimization algorithm of `PLN()`

## Examples

```
## Not run:
data(barents)
Y <- barents$Abundance
X <- model.matrix(Abundance ~ Latitude + Longitude + Depth + Temperature, data = barents)
O <- log(barents$Offset)
w <-- rep(1, nrow(Y))
compute_PLN_starting_point(Y, X, O, w)

## End(Not run)
```

---

`extract_probs` *Extract edge selection frequency in bootstrap subsamples*

---

## Description

Extracts edge selection frequency in networks reconstructed from bootstrap subsamples during the stars stability selection procedure, as either a matrix or a named vector. In the latter case, edge names follow igraph naming convention.

## Usage

```
extract_probs(
  Robject,
  penalty = NULL,
  index = NULL,
  crit = c("StARS", "BIC", "EBIC"),
  format = c("matrix", "vector"),
  tol = 1e-05
)
```

## Arguments

| | |
|---|---|
| `Robject` | an object with class `PLNnetworkfamily`, i.e. an output from `PLNnetwork()` |
| `penalty` | penalty used for the bootstrap subsamples |
| `index` | Integer index of the model to be returned. Only the first value is taken into account. |
| `crit` | a character for the criterion used to performed the selection. Either "BIC", "ICL", "EBIC", "StARS", "R_squared". Default is `ICL` for `PLNPCA`, and `BIC` for `PLNnetwork`. If StARS (Stability Approach to Regularization Selection) is chosen and stability selection was not yet performed, the function will call the method `stability_selection()` with default argument. |
| `format` | output format. Either a matrix (default) or a named vector. |
| `tol` | tolerance for rounding error when comparing penalties. |

## Value

Either a matrix or named vector of edge-wise probabilities. In the latter case, edge names follow igraph convention.

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
nets <- PLNnetwork(Abundance ~ 1 + offset(log(Offset)), data = trichoptera)
## Not run:
stability_selection(nets)
```

```
probs <- extract_probs(nets, crit = "StARS", format = "vector")
probs

## End(Not run)

## Not run:
## Add edge attributes to graph using igraph
net_stars <- getBestModel(nets, "StARS")
g <- plot(net_stars, type = "partial_cor", plot=F)
library(igraph)
E(g)$prob <- probs[as_ids(E(g))]
g

## End(Not run)
```

---

## `fitted.PLNfit` *Extracts model fitted values from objects returned by* `PLN()` *and its variants*

### Description

Extracts model fitted values from objects returned by `PLN()` and its variants

### Usage

```
## S3 method for class 'PLNfit'
fitted(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | an R6 object with class `PLNfit` |
| `...` | additional parameters for S3 compatibility. Not used |

### Value

A matrix of Fitted values extracted from the object object.

---

`fitted.PLNmixturefit` *Extracts model fitted values from objects returned by* `PLNmixture()` *and its variants*

---

## Description

Extracts model fitted values from objects returned by `PLNmixture()` and its variants

## Usage

```
## S3 method for class 'PLNmixturefit'
fitted(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | an R6 object with class `PLNmixturefit` |
| `...` | additional parameters for S3 compatibility. Not used |

## Value

A matrix of Fitted values extracted from the object object.

---

`fitted.ZIPLNfit` *Extracts model fitted values from objects returned by* `ZIPLN()` *and its variants*

---

## Description

Extracts model fitted values from objects returned by `ZIPLN()` and its variants

## Usage

```
## S3 method for class 'ZIPLNfit'
fitted(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | an R6 object with class `ZIPLNfit` |
| `...` | additional parameters for S3 compatibility. Not used |

## Value

A matrix of Fitted values extracted from the object object.

---

`getBestModel.PLNPCAfamily`

*Best model extraction from a collection of models*

---

## Description

Best model extraction from a collection of models

## Usage

```
## S3 method for class 'PLNPCAfamily'
getBestModel(Robject, crit = c("ICL", "BIC"), ...)

getBestModel(Robject, crit, ...)

## S3 method for class 'PLNmixturefamily'
getBestModel(Robject, crit = c("ICL", "BIC"), ...)

## S3 method for class 'Networkfamily'
getBestModel(Robject, crit = c("BIC", "EBIC", "StARS"), ...)

## S3 method for class 'PLNnetworkfamily'
getBestModel(Robject, crit = c("BIC", "EBIC", "StARS"), ...)

## S3 method for class 'ZIPLNnetworkfamily'
getBestModel(Robject, crit = c("BIC", "EBIC", "StARS"), ...)
```

## Arguments

| | |
|---|---|
| `Robject` | an object with class PLNPCAfamilly ot PLNnetworkfamily |
| `crit` | a character for the criterion used to performed the selection. Either "BIC", "ICL", "EBIC", "StARS", "R_squared". Default is `ICL` for `PLNPCA`, and `BIC` for `PLNnetwork`. If StARS (Stability Approach to Regularization Selection) is chosen and stability selection was not yet performed, the function will call the method `stability_selection()` with default argument. |
| `...` | additional parameters for StARS criterion (only for `PLNnetwork`). `stability`, a scalar indicating the target stability (= 1 - 2 beta) at which the network is selected. Default is `0.9`. |

## Value

Send back an object with class `PLNPCAfit` or `PLNnetworkfit`

## Methods (by class)

- `getBestModel(PLNPCAfamily)`: Model extraction for `PLNPCAfamily`

- `getBestModel(PLNmixturefamily)`: Model extraction for `PLNmixturefamily`
- `getBestModel(Networkfamily)`: Model extraction for `PLNnetworkfamily` or `ZIPLNnetworkfamily`
- `getBestModel(PLNnetworkfamily)`: Model extraction for `PLNnetworkfamily`
- `getBestModel(ZIPLNnetworkfamily)`: Model extraction for `ZIPLNnetworkfamily`

## Examples

```
## Not run:
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPCA <- PLNPCA(Abundance ~ 1 + offset(log(Offset)), data = trichoptera, ranks = 1:4)
myModel <- getBestModel(myPCA)

## End(Not run)
```

---

# `getModel.PLNPCAfamily` *Model extraction from a collection of models*

---

## Description

Model extraction from a collection of models

## Usage

```
## S3 method for class 'PLNPCAfamily'
getModel(Robject, var, index = NULL)

getModel(Robject, var, index)

## S3 method for class 'PLNmixturefamily'
getModel(Robject, var, index = NULL)

## S3 method for class 'Networkfamily'
getModel(Robject, var, index = NULL)

## S3 method for class 'PLNnetworkfamily'
getModel(Robject, var, index = NULL)

## S3 method for class 'ZIPLNnetworkfamily'
getModel(Robject, var, index = NULL)
```

## Arguments

| | |
|---|---|
| `Robject` | an R6 object with class `PLNPCAfamily` or `PLNnetworkfamily` |
| `var` | value of the parameter (`rank` for PLNPCA, `sparsity` for PLNnetwork) that identifies the model to be extracted from the collection. If no exact match is found, the model with closest parameter value is returned with a warning. |
| `index` | Integer index of the model to be returned. Only the first value is taken into account. |

## Value

Sends back an object with class `PLNPCAfit` or `PLNnetworkfit`.

## Methods (by class)

- `getModel(PLNPCAfamily)`: Model extraction for `PLNPCAfamily`
- `getModel(PLNmixturefamily)`: Model extraction for `PLNmixturefamily`
- `getModel(Networkfamily)`: Model extraction for `PLNnetworkfamily` or `ZIPLNnetworkfamily`
- `getModel(PLNnetworkfamily)`: Model extraction for `PLNnetworkfamily`
- `getModel(ZIPLNnetworkfamily)`: Model extraction for `ZIPLNnetworkfamily`

## Examples

```
## Not run:
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPCA <- PLNPCA(Abundance ~ 1 + offset(log(Offset)), data = trichoptera, ranks = 1:5)
myModel <- getModel(myPCA, 2)

## End(Not run)
```

---

# `mollusk` *Mollusk data set*

## Description

This data set gives the abundance of 32 mollusk species in 163 samples. For each sample, 4 additional covariates are known.

## Usage

`mollusk`

## Format

A list with 2 two data frames:

**Abundance** a 163 x 32 data frame of abundancies/counts (163 samples and 32 mollusk species)

**Covariate** a 163 x 4 data frame of covariates:

- **site** a factor with 8 levels indicating the sampling site
- **season** a factor with 4 levels indicating the season
- **method** a factor with 2 levels for the method of sampling - wood or string
- **duration** a numeric with 3 levels for the time of exposure in week

In order to prepare the data for using formula in multivariate analysis (multiple outputs and inputs), use `prepare_data()`. Original data set has been extracted from ade4.

### Source

Data from Richardot-Coulet, Chessel and Bournaud.

### References

Richardot-Coulet, M., Chessel D. and Bournaud M. (1986) Typological value of the benthos of old beds of a large river. Methodological approach. Archiv fùr Hydrobiologie, 107, 363–383.

### See Also

`prepare_data()`

### Examples

```
data(mollusk)
mollusc <- prepare_data(mollusk$Abundance, mollusk$Covariate)
```

---

## `Networkfamily` *An R6 Class to virtually represent a collection of network fits*

---

### Description

The functions `PLNnetwork()` and `ZIPLNnetwork()` both produce an instance of this class, which can be thought of as a vector of `PLNnetworkfit`s `ZIPLNfit_sparse`s (indexed by penalty parameter)

This class comes with a set of methods mostly used to compare network fits (in terms of goodness of fit) or extract one from the family (based on penalty parameter and/or goodness of it). See the documentation for `getBestModel()`, `getModel()` and plot() for the user-facing ones.

### Super class

`PLNmodels::PLNfamily` -> `Networkfamily`

### Active bindings

`penalties` the sparsity level of the network in the successively fitted models

`stability_path` the stability path of each edge as returned by the stars procedure

`stability` mean edge stability along the penalty path

`criteria` a data frame with the values of some criteria (variational log-likelihood, (E)BIC, ICL and R2, stability) for the collection of models / fits BIC, ICL and EBIC are defined so that they are on the same scale as the model log-likelihood, i.e. with the form, loglik - 0.5 penalty

## Methods

### Public methods:

- `Networkfamily$new()`
- `Networkfamily$optimize()`
- `Networkfamily$coefficient_path()`
- `Networkfamily$getBestModel()`
- `Networkfamily$plot()`
- `Networkfamily$plot_stars()`
- `Networkfamily$plot_objective()`
- `Networkfamily$show()`
- `Networkfamily$clone()`

**Method** `new()`**:** Initialize all models in the collection

*Usage:*

```
Networkfamily$new(penalties, data, control)
```

*Arguments:*

`penalties` a vector of positive real number controlling the level of sparsity of the underlying network.

`data` a named list used internally to carry the data matrices

`control` a list for controlling the optimization.

*Returns:* Update all network fits in the family with smart starting values

**Method** `optimize()`**:** Call to the C++ optimizer on all models of the collection

*Usage:*

```
Networkfamily$optimize(data, config)
```

*Arguments:*

`data` a named list used internally to carry the data matrices

`config` a list for controlling the optimization.

**Method** `coefficient_path()`**:** Extract the regularization path of a `Networkfamily`

*Usage:*

```
Networkfamily$coefficient_path(precision = TRUE, corr = TRUE)
```

*Arguments:*

`precision` Logical. Should the regularization path be extracted from the precision matrix Omega (`TRUE`, default) or from the variance matrix Sigma (`FALSE`)

`corr` Logical. Should the matrix be transformed to (partial) correlation matrix before extraction? Defaults to `TRUE`

**Method** `getBestModel()`**:** Extract the best network in the family according to some criteria

*Usage:*

```
Networkfamily$getBestModel(crit = c("BIC", "EBIC", "StARS"), stability = 0.9)
```

*Arguments:*

`crit` character. Criterion used to perform the selection. If "StARS" is chosen but `$stability` field is empty, will compute stability path.

`stability` Only used for "StARS" criterion. A scalar indicating the target stability (= 1 - 2 beta) at which the network is selected. Default is `0.9`.

*Details:* For BIC and EBIC criteria, higher is better.

**Method** `plot()`**:** Display various outputs (goodness-of-fit criteria, robustness, diagnostic) associated with a collection of network fits (a `Networkfamily`)

*Usage:*

```
Networkfamily$plot(
  criteria = c("loglik", "pen_loglik", "BIC", "EBIC"),
  reverse = FALSE,
  log.x = TRUE
)
```

*Arguments:*

`criteria` vector of characters. The criteria to plot in `c("loglik", "pen_loglik", "BIC", "EBIC")`. Defaults to all of them.

`reverse` A logical indicating whether to plot the value of the criteria in the "natural" direction (loglik - 0.5 penalty) or in the "reverse" direction (-2 loglik + penalty). Default to FALSE, i.e use the natural direction, on the same scale as the log-likelihood.

`log.x` logical: should the x-axis be represented in log-scale? Default is `TRUE`.

*Returns:* a `ggplot2::ggplot` graph

**Method** `plot_stars()`**:** Plot stability path

*Usage:*

```
Networkfamily$plot_stars(stability = 0.9, log.x = TRUE)
```

*Arguments:*

`stability` scalar: the targeted level of stability using stability selection. Default is `0.9`.

`log.x` logical: should the x-axis be represented in log-scale? Default is `TRUE`.

*Returns:* a `ggplot2::ggplot` graph

**Method** `plot_objective()`**:** Plot objective value of the optimization problem along the penalty path

*Usage:*

```
Networkfamily$plot_objective()
```

*Returns:* a `ggplot2::ggplot` graph

**Method** `show()`**:** User friendly print method

*Usage:*

```
Networkfamily$show()
```

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
Networkfamily$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

### See Also

The functions `PLNnetwork()`, `ZIPLNnetwork()` and the classes `PLNnetworkfit`, `ZIPLNfit_sparse`

---

## `oaks` *Oaks amplicon data set*

---

### Description

This data set gives the abundance of 114 taxa (66 bacterial OTU, 48 fungal OTUs) in 116 samples. For each sample, 11 additional covariates are known.

### Usage

```
oaks
```

### Format

A data frame with 13 variables:

- Abundance: A 114 taxa by 116 samples count matrix
- Offset: A 114 taxa by 116 samples offset matrix
- Sample: Unique sample id
- tree: Tree status with respect to the pathogen (susceptible, intermediate or resistant)
- branch: Unique branch id in each tree (4 branches were sampled in each tree, with 10 leaves per branch)
- leafNO: Unique leaf id in each tree (40 leaves were sampled in each tree)
- distTObase: Distance of the sampled leaf to the base of the branch
- distTOtrunk: Distance of the sampled leaf to the base of the tree trunk
- distTOground: Distance of the sampled leaf to the base of the ground
- pmInfection: Powdery mildew infection, proportion of the upper leaf area displaying mildew symptoms
- orientation: Orientation of the branch (South-West SW or North-East NE)
- readsTOTfun: Total number of ITS1 reads for that leaf
- readsTOTbac: Total number of 16S reads for that leaf

### Source

Data from B. Jakuschkin and coauthors.

### References

Jakuschkin, B., Fievet, V., Schwaller, L. et al. Deciphering the Pathobiome: Intra- and Interkingdom Interactions Involving the Pathogen Erysiphe alphitoides . Microb Ecol 72, 870–880 (2016). doi:10.1007/s002480160777x

## See Also

`prepare_data()`

## Examples

```
data(oaks)
## Not run:
oaks_networks <- PLNnetwork(formula = Abundance ~ 1 + offset(log(Offset)), data = oaks)

## End(Not run)
```

---

# PLN *Poisson lognormal model*

## Description

Fit the multivariate Poisson lognormal model with a variational algorithm. Use the (g)lm syntax for model specification (covariates, offsets, weights).

## Usage

```
PLN(formula, data, subset, weights, control = PLN_param())
```

## Arguments

| | |
|---|---|
| `formula` | an object of class "formula": a symbolic description of the model to be fitted. |
| `data` | an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which PLN is called. |
| `subset` | an optional vector specifying a subset of observations to be used in the fitting process. |
| `weights` | an optional vector of observation weights to be used in the fitting process. |
| `control` | a list-like structure for controlling the optimization, with default generated by `PLN_param()`. See the associated documentation for details. |

## Value

an R6 object with class `PLNfit`

## See Also

The class `PLNfit` and the configuration function `PLN_param()`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLN(Abundance ~ 1, data = trichoptera)
```

---

## `PLNfamily` *An R6 Class to represent a collection of PLNfit*

---

### Description

super class for `PLNPCAfamily` and `PLNnetworkfamily`.

### Public fields

`responses` the matrix of responses common to every models

`covariates` the matrix of covariates common to every models

`offsets` the matrix of offsets common to every models

`weights` the vector of observation weights

`inception` a PLNfit object, obtained when no sparsifying penalty is applied.

`models` a list of PLNfit object, one per penalty.

### Active bindings

`criteria` a data frame with the values of some criteria (approximated log-likelihood, BIC, ICL, etc.) for the collection of models / fits BIC and ICL are defined so that they are on the same scale as the model log-likelihood, i.e. with the form, loglik - 0.5 penalty

`convergence` sends back a data frame with some convergence diagnostics associated with the optimization process (method, optimal value, etc)

### Methods

**Public methods:**

- `PLNfamily$new()`
- `PLNfamily$postTreatment()`
- `PLNfamily$getModel()`
- `PLNfamily$plot()`
- `PLNfamily$show()`
- `PLNfamily$print()`
- `PLNfamily$clone()`

**Method** `new()`**:** Create a new `PLNfamily` object.

*Usage:*

```
PLNfamily$new(responses, covariates, offsets, weights, control)
```

*Arguments:*

`responses` the matrix of responses common to every models

`covariates` the matrix of covariates common to every models

`offsets` the matrix of offsets common to every models

`weights` the vector of observation weights `control` list controlling the optimization and the model

*Returns:* A new `PLNfamily` object

**Method** `postTreatment()`**:** Update fields after optimization

*Usage:*

```
PLNfamily$postTreatment(config_post, config_optim)
```

*Arguments:*

`config_post` a list for controlling the post-treatments (optional bootstrap, jackknife, R2, etc.).

`config_optim` a list for controlling the optimization parameters used during post_treatments

**Method** `getModel()`**:** Extract a model from a collection of models

*Usage:*

```
PLNfamily$getModel(var, index = NULL)
```

*Arguments:*

`var` value of the parameter (`rank` for PLNPCA, `sparsity` for PLNnetwork) that identifies the model to be extracted from the collection. If no exact match is found, the model with closest parameter value is returned with a warning.

`index` Integer index of the model to be returned. Only the first value is taken into account.

*Returns:* A `PLNfit` object

**Method** `plot()`**:** Lineplot of selected criteria for all models in the collection

*Usage:*

```
PLNfamily$plot(criteria, reverse)
```

*Arguments:*

`criteria` A valid model selection criteria for the collection of models. Includes loglik, BIC (all), ICL (PLNPCA) and pen_loglik, EBIC (PLNnetwork)

`reverse` A logical indicating whether to plot the value of the criteria in the "natural" direction (loglik - penalty) or in the "reverse" direction (-2 loglik + penalty). Default to FALSE, i.e use the natural direction, on the same scale as the log-likelihood.

*Returns:* A `ggplot2::ggplot` object

**Method** `show()`**:** User friendly print method

*Usage:*

```
PLNfamily$show()
```

**Method** `print()`**:** User friendly print method

*Usage:*

```
PLNfamily$print()
```

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNfamily$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

`getModel()`

---

`PLNfit` *An R6 Class to represent a PLNfit in a standard, general framework*

---

## Description

The function `PLN()` fit a model which is an instance of a object with class `PLNfit`. Objects produced by the functions `PLNnetwork()`, `PLNPCA()`, `PLNmixture()` and `PLNLDA()` also enjoy the methods of `PLNfit()` by inheritance.

This class comes with a set of R6 methods, some of them being useful for the user and exported as S3 methods. See the documentation for `coef()`, `sigma()`, `predict()`, `vcov()` and `standard_error()`.

Fields are accessed via active binding and cannot be changed by the user.

## Active bindings

`n` number of samples

`q` number of dimensions of the latent space

`p` number of species

`d` number of covariates

`nb_param` number of parameters in the current PLN model

`model_par` a list with the matrices of the model parameters: B (covariates), Sigma (covariance), Omega (precision matrix), plus some others depending on the variant)

`var_par` a list with the matrices of the variational parameters: M (means) and S2 (variances)

`optim_par` a list with parameters useful for monitoring the optimization

`latent` a matrix: values of the latent vector (Z in the model)

`latent_pos` a matrix: values of the latent position vector (Z) without covariates effects or offset

`fitted` a matrix: fitted values of the observations (A in the model)

`vcov_coef` matrix of sandwich estimator of the variance-covariance of B (need fixed -ie known- covariance at the moment)

`vcov_model` character: the model used for the residual covariance

`weights` observational weights

`loglik` (weighted) variational lower bound of the loglikelihood

`loglik_vec` element-wise variational lower bound of the loglikelihood

`BIC` variational lower bound of the BIC

`entropy` Entropy of the variational distribution

`ICL` variational lower bound of the ICL

`R_squared` approximated goodness-of-fit criterion

`criteria` a vector with loglik, BIC, ICL and number of parameters

## Methods

### Public methods:

- `PLNfit$new()`
- `PLNfit$update()`
- `PLNfit$optimize()`
- `PLNfit$optimize_vestep()`
- `PLNfit$postTreatment()`
- `PLNfit$predict()`
- `PLNfit$predict_cond()`
- `PLNfit$show()`
- `PLNfit$print()`
- `PLNfit$clone()`

**Method** `new()`**:** Initialize a `PLNfit` model

*Usage:*

```
PLNfit$new(responses, covariates, offsets, weights, formula, control)
```

*Arguments:*

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`weights` an optional vector of observation weights to be used in the fitting process.

`formula` model formula used for fitting, extracted from the formula in the upper-level call

`control` a list-like structure for controlling the fit, see `PLN_param()`.

**Method** `update()`**:** Update a `PLNfit` object

*Usage:*

```
PLNfit$update(
  B = NA,
  Sigma = NA,
  Omega = NA,
  M = NA,
  S = NA,
  Ji = NA,
  R2 = NA,
  Z = NA,
  A = NA,
  monitoring = NA
)
```

*Arguments:*

`B` matrix of regression matrix

`Sigma` variance-covariance matrix of the latent variables

`Omega` precision matrix of the latent variables. Inverse of Sigma.

`M` matrix of variational parameters for the mean

`S` matrix of variational parameters for the variance

`Ji` vector of variational lower bounds of the log-likelihoods (one value per sample)

`R2` approximate R^2 goodness-of-fit criterion

`Z` matrix of latent vectors (includes covariates and offset effects)

`A` matrix of fitted values

`monitoring` a list with optimization monitoring quantities

*Returns:* Update the current `PLNfit` object

**Method** `optimize()`**:** Call to the NLopt or TORCH optimizer and update of the relevant fields

*Usage:*

```
PLNfit$optimize(responses, covariates, offsets, weights, config)
```

*Arguments:*

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`weights` an optional vector of observation weights to be used in the fitting process.

`config` part of the `control` argument which configures the optimizer

**Method** `optimize_vestep()`**:** Result of one call to the VE step of the optimization procedure: optimal variational parameters (M, S) and corresponding log likelihood values for fixed model parameters (Sigma, B). Intended to position new data in the latent space.

*Usage:*

```
PLNfit$optimize_vestep(
  covariates,
  offsets,
  responses,
  weights,
  B = self$model_par$B,
  Omega = self$model_par$Omega,
  control = PLN_param(backend = "nlopt")
)
```

*Arguments:*

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`weights` an optional vector of observation weights to be used in the fitting process.

`B` Optional fixed value of the regression parameters

`Omega` precision matrix of the latent variables. Inverse of Sigma.

`control` a list-like structure for controlling the fit, see `PLN_param()`.

`Sigma` variance-covariance matrix of the latent variables

*Returns:* A list with three components:

- the matrix `M` of variational means,
- the matrix `S2` of variational variances
- the vector `log.lik` of (variational) log-likelihood of each new observation

**Method** `postTreatment()`**:** Update R2, fisher and std_err fields after optimization

*Usage:*

```
PLNfit$postTreatment(
  responses,
  covariates,
  offsets,
  weights = rep(1, nrow(responses)),
  config_post,
  config_optim,
  nullModel = NULL
)
```

*Arguments:*

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`weights` an optional vector of observation weights to be used in the fitting process.

`config_post` a list for controlling the post-treatments (optional bootstrap, jackknife, R2, etc.). See details

`config_optim` a list for controlling the optimization (optional bootstrap, jackknife, R2, etc.). See details

`nullModel` null model used for approximate R2 computations. Defaults to a GLM model with same design matrix but not latent variable.

*Details:* The list of parameters `config` controls the post-treatment processing, with the following entries:

- jackknife boolean indicating whether jackknife should be performed to evaluate bias and variance of the model parameters. Default is FALSE.
- bootstrap integer indicating the number of bootstrap resamples generated to evaluate the variance of the model parameters. Default is 0 (inactivated).
- variational_var boolean indicating whether variational Fisher information matrix should be computed to estimate the variance of the model parameters (highly underestimated). Default is FALSE.
- sandwich_var boolean indicating whether sandwich estimator should be computed to estimate the variance of the model parameters (highly underestimated). Default is FALSE.

- trace integer for verbosity. should be > 1 to see output in post-treatments

**Method** `predict()`**:** Predict position, scores or observations of new data.

*Usage:*

```
PLNfit$predict(
  newdata,
  responses = NULL,
  type = c("link", "response"),
  level = 1,
  envir = parent.frame()
)
```

*Arguments:*

`newdata` A data frame in which to look for variables with which to predict. If omitted, the fitted values are used.

`responses` Optional data frame containing the count of the observed variables (matching the names of the provided as data in the PLN function), assuming the interest in in testing the model.

`type` Scale used for the prediction. Either `link` (default, predicted positions in the latent space) or `response` (predicted counts).

`level` Optional integer value the level to be used in obtaining the predictions. Level zero corresponds to the population predictions (default if `responses` is not provided) while level one (default) corresponds to predictions after evaluating the variational parameters for the new data.

`envir` Environment in which the prediction is evaluated

*Details:* Note that `level = 1` can only be used if responses are provided, as the variational parameters can't be estimated otherwise. In the absence of responses, `level` is ignored and the fitted values are returned

*Returns:* A matrix with predictions scores or counts.

**Method** `predict_cond()`**:** Predict position, scores or observations of new data, conditionally on the observation of a (set of) variables

*Usage:*

```
PLNfit$predict_cond(
  newdata,
  cond_responses,
  type = c("link", "response"),
  var_par = FALSE,
  envir = parent.frame()
)
```

*Arguments:*

`newdata` a data frame containing the covariates of the sites where to predict

`cond_responses` a data frame containing the count of the observed variables (matching the names of the provided as data in the PLN function)

`type` Scale used for the prediction. Either `link` (default, predicted positions in the latent space) or `response` (predicted counts).

`var_par` Boolean. Should new estimations of the variational parameters of mean and variance be sent back, as attributes of the matrix of predictions. Default to `FALSE`.

`envir` Environment in which the prediction is evaluated

*Returns:* A matrix with predictions scores or counts.

**Method** `show()`**:** User friendly print method

*Usage:*

```
PLNfit$show(
  model = paste("A multivariate Poisson Lognormal fit with", self$vcov_model,
    "covariance model.\n")
)
```

*Arguments:*

`model` First line of the print output

**Method** `print()`**:** User friendly print method

*Usage:*

```
PLNfit$print()
```

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNfit$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
## Not run:
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLN(Abundance ~ 1, data = trichoptera)
class(myPLN)
print(myPLN)

## End(Not run)
```

---

# `PLNfit_diagonal` *An R6 Class to represent a PLNfit in a standard, general framework, with diagonal residual covariance*

## Description

The function `PLNLDA()` produces an instance of an object with class `PLNLDAfit`.

This class comes with a set of methods, some of them being useful for the user: See the documentation for the methods inherited by `PLNfit()`, the `plot()` method for LDA visualization and `predict()` method for prediction

## Super class

`PLNmodels::PLNfit` -> `PLNfit_diagonal`

## Active bindings

`nb_param` number of parameters in the current PLN model

`vcov_model` character: the model used for the residual covariance

## Methods

### Public methods:

- `PLNfit_diagonal$new()`
- `PLNfit_diagonal$clone()`

**Method** `new()`**:** Initialize a `PLNfit` model

*Usage:*

```
PLNfit_diagonal$new(responses, covariates, offsets, weights, formula, control)
```

*Arguments:*

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`weights` an optional vector of observation weights to be used in the fitting process.

`formula` model formula used for fitting, extracted from the formula in the upper-level call

`control` a list for controlling the optimization. See details.

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNfit_diagonal$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Super classes

`PLNmodels::PLNfit` -> `PLNmodels::PLNLDAfit` -> `PLNLDAfit_spherical`

## Active bindings

`vcov_model` character: the model used for the residual covariance

`nb_param` number of parameters in the current PLN model

## Methods

### Public methods:

- `PLNLDAfit_spherical$new()`
- `PLNLDAfit_spherical$clone()`

**Method** `new()`**:** Initialize a `PLNfit` model

*Usage:*

```
PLNLDAfit_spherical$new(
  grouping,
  responses,
  covariates,
  offsets,
  weights,
  formula,
  control
)
```

*Arguments:*

`grouping` a factor specifying the class of each observation used for discriminant analysis.

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`weights` an optional vector of observation weights to be used in the fitting process.

`formula` model formula used for fitting, extracted from the formula in the upper-level call

`control` a list for controlling the optimization. See details.

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNLDAfit_spherical$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
## Not run:
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLN(Abundance ~ 1, data = trichoptera)
class(myPLN)
print(myPLN)

## End(Not run)
## Not run:
data(trichoptera)
```

```
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLNLDA <- PLNLDA(Abundance ~ 1, data = trichoptera, control = PLN_param(covariance = "spherical"))
class(myPLNLDA)
print(myPLNLDA)

## End(Not run)
```

---

## `PLNfit_fixedcov` *An R6 Class to represent a PLNfit in a standard, general framework, with fixed (inverse) residual covariance*

---

### Description

An R6 Class to represent a PLNfit in a standard, general framework, with fixed (inverse) residual covariance

An R6 Class to represent a PLNfit in a standard, general framework, with fixed (inverse) residual covariance

### Super class

`PLNmodels::PLNfit` -> `PLNfit_fixedcov`

### Active bindings

`nb_param` number of parameters in the current PLN model

`vcov_model` character: the model used for the residual covariance

`vcov_coef` matrix of sandwich estimator of the variance-covariance of B (needs known covariance at the moment)

### Methods

**Public methods:**

- `PLNfit_fixedcov$new()`
- `PLNfit_fixedcov$optimize()`
- `PLNfit_fixedcov$clone()`

**Method** `new()`**:** Initialize a `PLNfit` model

*Usage:*

`PLNfit_fixedcov$new(responses, covariates, offsets, weights, formula, control)`

*Arguments:*

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class `offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`weights` an optional vector of observation weights to be used in the fitting process.

`formula` model formula used for fitting, extracted from the formula in the upper-level call

`control` a list for controlling the optimization. See details.

**Method** `optimize()`**:** Call to the NLopt or TORCH optimizer and update of the relevant fields

*Usage:*

```
PLNfit_fixedcov$optimize(responses, covariates, offsets, weights, config)
```

*Arguments:*

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`weights` an optional vector of observation weights to be used in the fitting process.

`config` part of the `control` argument which configures the optimizer

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNfit_fixedcov$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
## Not run:
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLN(Abundance ~ 1, data = trichoptera)
class(myPLN)
print(myPLN)

## End(Not run)
```

---

# `PLNfit_spherical` *An R6 Class to represent a PLNfit in a standard, general framework, with spherical residual covariance*

## Description

An R6 Class to represent a PLNfit in a standard, general framework, with spherical residual covariance

An R6 Class to represent a PLNfit in a standard, general framework, with spherical residual covariance

## Super class

`PLNmodels::PLNfit` -> `PLNfit_spherical`

## Active bindings

`nb_param` number of parameters in the current PLN model

`vcov_model` character: the model used for the residual covariance

## Methods

### Public methods:

- `PLNfit_spherical$new()`
- `PLNfit_spherical$clone()`

**Method** `new()`**:** Initialize a `PLNfit` model

*Usage:*

```
PLNfit_spherical$new(responses, covariates, offsets, weights, formula, control)
```

*Arguments:*

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`weights` an optional vector of observation weights to be used in the fitting process.

`formula` model formula used for fitting, extracted from the formula in the upper-level call

`control` a list for controlling the optimization. See details.

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNfit_spherical$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
## Not run:
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLN(Abundance ~ 1, data = trichoptera)
class(myPLN)
print(myPLN)

## End(Not run)
```

---

`PLNLDA` *Poisson lognormal model towards Linear Discriminant Analysis*

---

## Description

Fit the Poisson lognormal for LDA with a variational algorithm. Use the (g)lm syntax for model specification (covariates, offsets).

## Usage

```
PLNLDA(formula, data, subset, weights, grouping, control = PLN_param())
```

## Arguments

| | |
|---|---|
| `formula` | an object of class "formula": a symbolic description of the model to be fitted. |
| `data` | an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which lm is called. |
| `subset` | an optional vector specifying a subset of observations to be used in the fitting process. |
| `weights` | an optional vector of observation weights to be used in the fitting process. |
| `grouping` | a factor specifying the class of each observation used for discriminant analysis. |
| `control` | a list-like structure for controlling the optimization, with default generated by `PLN_param()`. See the associated documentation |

## Details

The parameter `control` is a list controlling the optimization with the following entries:

- "covariance" character setting the model for the covariance matrix. Either "full" or "spherical". Default is "full".
- "trace" integer for verbosity.
- "inception" Set up the initialization. By default, the model is initialized with a multivariate linear model applied on log-transformed data. However, the user can provide a PLNfit (typically obtained from a previous fit), which often speed up the inference.
- "ftol_rel" stop when an optimization step changes the objective function by less than ftol multiplied by the absolute value of the parameter. Default is 1e-8
- "ftol_abs" stop when an optimization step changes the objective function by less than ftol multiplied by the absolute value of the parameter. Default is 0
- "xtol_rel" stop when an optimization step changes every parameters by less than xtol multiplied by the absolute value of the parameter. Default is 1e-6
- "xtol_abs" stop when an optimization step changes every parameters by less than xtol multiplied by the absolute value of the parameter. Default is 0

- "maxeval" stop when the number of iteration exceeds maxeval. Default is 10000
- "maxtime" stop when the optimization time (in seconds) exceeds maxtime. Default is -1 (no restriction)
- "algorithm" the optimization method used by NLOPT among LD type, i.e. "CCSAQ", "MMA", "LBFGS", "VAR1", "VAR2". See NLOPT documentation for further details. Default is "CCSAQ".

## Value

an R6 object with class `PLNLDAfit()`

## See Also

The class `PLNLDAfit`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLNLDA <- PLNLDA(Abundance ~ 1, grouping = Group, data = trichoptera)
```

---

# `PLNLDAfit` *An R6 Class to represent a PLNfit in a LDA framework*

## Description

The function `PLNLDA()` produces an instance of an object with class `PLNLDAfit`.

This class comes with a set of methods, some of them being useful for the user: See the documentation for the methods inherited by `PLNfit()`, the `plot()` method for LDA visualization and `predict()` method for prediction

## Super class

`PLNmodels::PLNfit` -> `PLNLDAfit`

## Active bindings

`rank` the dimension of the current model

`nb_param` number of parameters in the current PLN model

`model_par` a list with the matrices associated with the estimated parameters of the PLN model: B (covariates), Sigma (latent covariance), C (latent loadings), P (latent position) and Mu (group means)

`percent_var` the percent of variance explained by each axis

`corr_map` a matrix of correlations to plot the correlation circles

`scores` a matrix of scores to plot the individual factor maps

`group_means` a matrix of group mean vectors in the latent space.

## Methods

### Public methods:

- `PLNLDAfit$new()`
- `PLNLDAfit$optimize()`
- `PLNLDAfit$postTreatment()`
- `PLNLDAfit$setVisualization()`
- `PLNLDAfit$plot_individual_map()`
- `PLNLDAfit$plot_correlation_map()`
- `PLNLDAfit$plot_LDA()`
- `PLNLDAfit$predict()`
- `PLNLDAfit$show()`
- `PLNLDAfit$clone()`

**Method** `new()`**:** Initialize a `PLNLDAfit` object

*Usage:*

```
PLNLDAfit$new(
  grouping,
  responses,
  covariates,
  offsets,
  weights,
  formula,
  control
)
```

*Arguments:*

`grouping` a factor specifying the class of each observation used for discriminant analysis.

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`weights` an optional vector of observation weights to be used in the fitting process.

`formula` model formula used for fitting, extracted from the formula in the upper-level call

`control` list controlling the optimization and the model

**Method** `optimize()`**:** Compute group means and axis of the LDA (noted B in the model) in the latent space, update corresponding fields

*Usage:*

```
PLNLDAfit$optimize(grouping, responses, covariates, offsets, weights, config)
```

*Arguments:*

`grouping` a factor specifying the class of each observation used for discriminant analysis.

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class `covariates` design matrix. Automatically built from the covariates and the formula from the call

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`weights` an optional vector of observation weights to be used in the fitting process.

`config` list controlling the optimization

`X` Abundance matrix.

**Method** `postTreatment()`**:** Update R2, fisher and std_err fields and visualization

*Usage:*

```
PLNLDAfit$postTreatment(
  grouping,
  responses,
  covariates,
  offsets,
  config_post,
  config_optim
)
```

*Arguments:*

`grouping` a factor specifying the class of each observation used for discriminant analysis.

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`config_post` a list for controlling the post-treatments (optional bootstrap, jackknife, R2, etc.).

`config_optim` list controlling the optimization parameters

**Method** `setVisualization()`**:** Compute LDA scores in the latent space and update corresponding fields.

*Usage:*

```
PLNLDAfit$setVisualization(scale.unit = FALSE)
```

*Arguments:*

`scale.unit` Logical. Should LDA scores be rescaled to have unit variance

**Method** `plot_individual_map()`**:** Plot the factorial map of the LDA

*Usage:*

```
PLNLDAfit$plot_individual_map(
  axes = 1:min(2, self$rank),
  main = "Individual Factor Map",
  plot = TRUE
)
```

*Arguments:*

`axes` numeric, the axes to use for the plot when map = "individual" or "variable". Default it c(1,min(rank))

`main` character. A title for the single plot (individual or variable factor map). If NULL (the default), an hopefully appropriate title will be used.

`plot` logical. Should the plot be displayed or sent back as ggplot object

*Returns:* a `ggplot2::ggplot` graphic

**Method** `plot_correlation_map()`**:** Plot the correlation circle of a specified axis for a `PLNLDAfit` object

*Usage:*

```
PLNLDAfit$plot_correlation_map(
  axes = 1:min(2, self$rank),
  main = "Variable Factor Map",
  cols = "default",
  plot = TRUE
)
```

*Arguments:*

`axes` numeric, the axes to use for the plot when map = "individual" or "variable". Default it c(1,min(rank))

`main` character. A title for the single plot (individual or variable factor map). If NULL (the default), an hopefully appropriate title will be used.

`cols` a character, factor or numeric to define the color associated with the variables. By default, all variables receive the default color of the current palette.

`plot` logical. Should the plot be displayed or sent back as ggplot object

*Returns:* a `ggplot2::ggplot` graphic

**Method** `plot_LDA()`**:** Plot a summary of the `PLNLDAfit` object

*Usage:*

```
PLNLDAfit$plot_LDA(
  nb_axes = min(3, self$rank),
  var_cols = "default",
  plot = TRUE
)
```

*Arguments:*

`nb_axes` scalar: the number of axes to be considered when map = "both". The default is min(3,rank).

`var_cols` a character, factor or numeric to define the color associated with the variables. By default, all variables receive the default color of the current palette.

`plot` logical. Should the plot be displayed or sent back as ggplot object

*Returns:* a `grob` object

**Method** `predict()`**:** Predict group of new samples

*Usage:*

```
PLNLDAfit$predict(
  newdata,
  type = c("posterior", "response", "scores"),
  scale = c("log", "prob"),
  prior = NULL,
  control = PLN_param(backend = "nlopt"),
  envir = parent.frame()
)
```

*Arguments:*

`newdata` A data frame in which to look for variables, offsets and counts with which to predict.

`type` The type of prediction required. The default are posterior probabilities for each group (in either unnormalized log-scale or natural probabilities, see "scale" for details), "response" is the group with maximal posterior probability and "scores" is the average score along each separation axis in the latent space, with weights equal to the posterior probabilities.

`scale` The scale used for the posterior probability. Either log-scale ("log", default) or natural probabilities summing up to 1 ("prob").

`prior` User-specified prior group probabilities in the new data. If NULL (default), prior probabilities are computed from the learning set.

`control` a list for controlling the optimization. See `PLN()` for details.

`envir` Environment in which the prediction is evaluated

**Method** `show()`**:** User friendly print method

*Usage:*

```
PLNLDAfit$show()
```

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNLDAfit$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

The function `PLNLDA`.

## Examples

```
## Not run:
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLNLDA <- PLNLDA(Abundance ~ 1, grouping = Group, data = trichoptera)
class(myPLNLDA)
print(myPLNLDA)

## End(Not run)
```

---

`PLNLDAfit_diagonal` *An R6 Class to represent a PLNfit in a LDA framework with diagonal covariance*

---

## Description

The function `PLNLDA()` produces an instance of an object with class `PLNLDAfit`.

This class comes with a set of methods, some of them being useful for the user: See the documentation for the methods inherited by `PLNfit()`, the `plot()` method for LDA visualization and `predict()` method for prediction

## Super classes

`PLNmodels::PLNfit` -> `PLNmodels::PLNLDAfit` -> `PLNLDAfit_diagonal`

## Active bindings

`vcov_model` character: the model used for the residual covariance

`nb_param` number of parameters in the current PLN model

## Methods

### Public methods:

- `PLNLDAfit_diagonal$new()`
- `PLNLDAfit_diagonal$clone()`

**Method** `new()`**:** Initialize a `PLNfit` model

*Usage:*

```
PLNLDAfit_diagonal$new(
  grouping,
  responses,
  covariates,
  offsets,
  weights,
  formula,
  control
)
```

*Arguments:*

`grouping` a factor specifying the class of each observation used for discriminant analysis.

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in PLNfamily-class

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in PLNfamily-class

| | |
|---|---|
| `weights` | an optional vector of observation weights to be used in the fitting process. |
| `formula` | model formula used for fitting, extracted from the formula in the upper-level call |
| `control` | a list for controlling the optimization. See details. |

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNLDAfit_diagonal$clone(deep = FALSE)
```

*Arguments:*

| | |
|---|---|
| `deep` | Whether to make a deep clone. |

## Examples

```
## Not run:
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLNLDA <- PLNLDA(Abundance ~ 1, data = trichoptera, control = PLN_param(covariance = "diagonal"))
class(myPLNLDA)
print(myPLNLDA)

## End(Not run)
```

---

# `PLNLDA_param` *Control of a PLNLDA fit*

## Description

Helper to define list of parameters to control the PLNLDA fit. All arguments have defaults.

## Usage

```
PLNLDA_param(
  backend = c("nlopt", "torch"),
  trace = 1,
  covariance = c("full", "diagonal", "spherical"),
  config_post = list(),
  config_optim = list(),
  inception = NULL
)
```

## Arguments

| | |
|---|---|
| `backend` | optimization back used, either "nlopt" or "torch". Default is "nlopt" |
| `trace` | a integer for verbosity. |
| `covariance` | character setting the model for the covariance matrix. Either "full", "diagonal" or "spherical". Default is "full". |

| | |
|---|---|
| `config_post` | a list for controlling the post-treatments (optional bootstrap, jackknife, R2, etc.). See details |
| `config_optim` | a list for controlling the optimizer (either "nlopt" or "torch" backend). See details |
| `inception` | Set up the parameters initialization: by default, the model is initialized with a multivariate linear model applied on log-transformed data, and with the same formula as the one provided by the user. However, the user can provide a PLNfit (typically obtained from a previous fit), which sometimes speeds up the inference. |

## Details

The list of parameters `config_optim` controls the optimizers. When "nlopt" is chosen the following entries are relevant

- "algorithm" the optimization method used by NLOPT among LD type, e.g. "CCSAQ", "MMA", "LBFGS". See NLOPT documentation for further details. Default is "CCSAQ".
- "maxeval" stop when the number of iteration exceeds maxeval. Default is 10000
- "ftol_rel" stop when an optimization step changes the objective function by less than ftol multiplied by the absolute value of the parameter. Default is 1e-8
- "xtol_rel" stop when an optimization step changes every parameters by less than xtol multiplied by the absolute value of the parameter. Default is 1e-6
- "ftol_abs" stop when an optimization step changes the objective function by less than ftol_abs. Default is 0.0 (disabled)
- "xtol_abs" stop when an optimization step changes every parameters by less than xtol_abs. Default is 0.0 (disabled)
- "maxtime" stop when the optimization time (in seconds) exceeds maxtime. Default is -1 (disabled)

When "torch" backend is used (only for PLN and PLNLDA for now), the following entries are relevant:

- "algorithm" the optimizer used by torch among RPROP (default), RMSPROP, ADAM and ADAGRAD
- "maxeval" stop when the number of iteration exceeds maxeval. Default is 10 000
- "numepoch" stop training once this number of epochs exceeds numepoch. Set to -1 to enable infinite training. Default is 1 000
- "num_batch" number of batches to use during training. Defaults to 1 (use full dataset at each epoch)
- "ftol_rel" stop when an optimization step changes the objective function by less than ftol multiplied by the absolute value of the parameter. Default is 1e-8
- "xtol_rel" stop when an optimization step changes every parameters by less than xtol multiplied by the absolute value of the parameter. Default is 1e-6
- "lr" learning rate. Default is 0.1.
- "momentum" momentum factor. Default is 0 (no momentum). Only used in RMSPROP

- "weight_decay" Weight decay penalty. Default is 0 (no decay). Not used in RPROP
- "step_sizes" pair of minimal (default: 1e-6) and maximal (default: 50) allowed step sizes. Only used in RPROP
- "etas" pair of multiplicative increase and decrease factors. Default is (0.5, 1.2). Only used in RPROP
- "centered" if TRUE, compute the centered RMSProp where the gradient is normalized by an estimation of its variance weight_decay (L2 penalty). Default to FALSE. Only used in RMSPROP

The list of parameters `config_post` controls the post-treatment processing (for most `PLN*()` functions), with the following entries (defaults may vary depending on the specific function, check `config_post_default_*` for defaults values):

- jackknife boolean indicating whether jackknife should be performed to evaluate bias and variance of the model parameters. Default is FALSE.
- bootstrap integer indicating the number of bootstrap resamples generated to evaluate the variance of the model parameters. Default is 0 (inactivated).
- variational_var boolean indicating whether variational Fisher information matrix should be computed to estimate the variance of the model parameters (highly underestimated). Default is FALSE.
- sandwich_var boolean indicating whether sandwich estimation should be used to estimate the variance of the model parameters (highly underestimated). Default is FALSE.
- rsquared boolean indicating whether approximation of R2 based on deviance should be computed. Default is TRUE

## Value

list of parameters configuring the fit.

---

# `PLNmixture` *Poisson lognormal mixture model*

## Description

Fit the mixture variants of the Poisson lognormal with a variational algorithm. Use the (g)lm syntax for model specification (covariates, offsets).

## Usage

```
PLNmixture(formula, data, subset, clusters = 1:5, control = PLNmixture_param())
```

### Arguments

| | |
|---|---|
| `formula` | an object of class "formula": a symbolic description of the model to be fitted. |
| `data` | an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which lm is called. |
| `subset` | an optional vector specifying a subset of observations to be used in the fitting process. |
| `clusters` | a vector of integer containing the successive number of clusters (or components) to be considered |
| `control` | a list-like structure for controlling the optimization, with default generated by `PLNmixture_param()`. See the associated documentation for details. |

### Value

an R6 object with class `PLNmixturefamily`, which contains a collection of models with class `PLNmixturefit`

### See Also

The classes `PLNmixturefamily`, `PLNmixturefit` and `PLNmixture_param()`

### Examples

```
## Use future to dispatch the computations on 2 workers
## Not run:
future::plan("multisession", workers = 2)

## End(Not run)

data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myMixtures <- PLNmixture(Abundance ~ 1 + offset(log(Offset)), clusters = 1:4, data = trichoptera,
                         control = PLNmixture_param(smoothing = 'none'))

# Shut down parallel workers
## Not run:
future::plan("sequential")

## End(Not run)
```

---

## `PLNmixturefamily` *An R6 Class to represent a collection of PLNmixturefit*

## Description

The function `PLNmixture()` produces an instance of this class.

This class comes with a set of methods, some of them being useful for the user: See the documentation for `getBestModel()`, `getModel()` and `plot()`.

## Super class

`PLNmodels::PLNfamily` -> `PLNmixturefamily`

## Active bindings

`clusters` vector indicating the number of clusters considered is the successively fitted models

## Methods

### Public methods:

- `PLNmixturefamily$new()`
- `PLNmixturefamily$optimize()`
- `PLNmixturefamily$smooth()`
- `PLNmixturefamily$plot()`
- `PLNmixturefamily$plot_objective()`
- `PLNmixturefamily$getBestModel()`
- `PLNmixturefamily$show()`
- `PLNmixturefamily$print()`
- `PLNmixturefamily$clone()`

**Method `new()`:** helper function for forward smoothing: split a group

Initialize all models in the collection.

*Usage:*

```
PLNmixturefamily$new(
  clusters,
  responses,
  covariates,
  offsets,
  formula,
  control
)
```

*Arguments:*

`clusters` the dimensions of the successively fitted models

`responses` the matrix of responses common to every models

`covariates` the matrix of covariates common to every models

`offsets` the matrix of offsets common to every models

`formula` model formula used for fitting, extracted from the formula in the upper-level call

`control` a list for controlling the optimization. See details.

`control` a list for controlling the optimization. See details.

**Method** `optimize()`**:** Call to the optimizer on all models of the collection

*Usage:*

```
PLNmixturefamily$optimize(config)
```

*Arguments:*

`config` a list for controlling the optimization

**Method** `smooth()`**:** function to restart clustering to avoid local minima by smoothing the log-likelihood values as a function of the number of clusters

*Usage:*

```
PLNmixturefamily$smooth(control)
```

*Arguments:*

`control` a list to control the smoothing process

**Method** `plot()`**:** Lineplot of selected criteria for all models in the collection

*Usage:*

```
PLNmixturefamily$plot(criteria = c("loglik", "BIC", "ICL"), reverse = FALSE)
```

*Arguments:*

`criteria` A valid model selection criteria for the collection of models. Any of "loglik", "BIC" or "ICL" (all).

`reverse` A logical indicating whether to plot the value of the criteria in the "natural" direction (loglik - 0.5 penalty) or in the "reverse" direction (-2 loglik + penalty). Default to FALSE, i.e use the natural direction, on the same scale as the log-likelihood..

*Returns:* A `ggplot2::ggplot` object

**Method** `plot_objective()`**:** Plot objective value of the optimization problem along the penalty path

*Usage:*

```
PLNmixturefamily$plot_objective()
```

*Returns:* a `ggplot2::ggplot` graph

**Method** `getBestModel()`**:** Extract best model in the collection

*Usage:*

```
PLNmixturefamily$getBestModel(crit = c("BIC", "ICL", "loglik"))
```

*Arguments:*

`crit` a character for the criterion used to performed the selection. Either "BIC", "ICL" or "loglik". Default is `ICL`

*Returns:* a `PLNmixturefit` object

**Method** `show()`**:** User friendly print method

*Usage:*

```
PLNmixturefamily$show()
```

**Method** `print()`**:** User friendly print method

*Usage:*

```
PLNmixturefamily$print()
```

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNmixturefamily$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

The function `PLNmixture`, the class `PLNmixturefit`

---

`PLNmixturefit` *An R6 Class to represent a PLNfit in a mixture framework*

---

## Description

The function `PLNmixture` produces a collection of models which are instances of object with class `PLNmixturefit`. A `PLNmixturefit` (say, with k components) is itself a collection of k `PLNfit`.

This class comes with a set of methods, some of them being useful for the user: See the documentation for ...

## Active bindings

| | |
|---|---|
| `n` | number of samples |
| `p` | number of dimensions of the latent space |
| `k` | number of components |
| `d` | number of covariates |
| `components` | components of the mixture (PLNfits) |
| `latent` | a matrix: values of the latent vector (Z in the model) |
| `latent_pos` | a matrix: values of the latent position vector (Z) without covariates effects or offset |
| `posteriorProb` | matrix ofposterior probability for cluster belonging |
| `memberships` | vector for cluster index |
| `mixtureParam` | vector of cluster proportions |
| `optim_par` | a list with parameters useful for monitoring the optimization |
| `nb_param` | number of parameters in the current PLN model |
| `entropy_clustering` | Entropy of the variational distribution of the cluster (multinomial) |
| `entropy_latent` | Entropy of the variational distribution of the latent vector (Gaussian) |
| `entropy` | Full entropy of the variational distribution (latent vector + clustering) |
| `loglik` | variational lower bound of the loglikelihood |

`loglik_vec` element-wise variational lower bound of the loglikelihood

`BIC` variational lower bound of the BIC

`ICL` variational lower bound of the ICL (include entropy of both the clustering and latent distributions)

`R_squared` approximated goodness-of-fit criterion

`criteria` a vector with loglik, BIC, ICL, and number of parameters

`model_par` a list with the matrices of parameters found in the model (Theta, Sigma, Mu and Pi)

`vcov_model` character: the model used for the covariance (either "spherical", "diagonal" or "full")

`fitted` a matrix: fitted values of the observations (A in the model)

`group_means` a matrix of group mean vectors in the latent space.

## Methods

### Public methods:

- `PLNmixturefit$new()`
- `PLNmixturefit$optimize()`
- `PLNmixturefit$predict()`
- `PLNmixturefit$plot_clustering_data()`
- `PLNmixturefit$plot_clustering_pca()`
- `PLNmixturefit$postTreatment()`
- `PLNmixturefit$show()`
- `PLNmixturefit$print()`
- `PLNmixturefit$clone()`

**Method** `new()`**:** Optimize a the

Initialize a `PLNmixturefit` model

*Usage:*

```
PLNmixturefit$new(
  responses,
  covariates,
  offsets,
  posteriorProb,
  formula,
  control
)
```

*Arguments:*

`responses` the matrix of responses common to every models

`covariates` the matrix of covariates common to every models

`offsets` the matrix of offsets common to every models

`posteriorProb` matrix ofposterior probability for cluster belonging

`formula` model formula used for fitting, extracted from the formula in the upper-level call

`control` a list for controlling the optimization.

**Method** `optimize()`**:** Optimize a `PLNmixturefit` model

*Usage:*

```
PLNmixturefit$optimize(responses, covariates, offsets, config)
```

*Arguments:*

`responses` the matrix of responses common to every models

`covariates` the matrix of covariates common to every models

`offsets` the matrix of offsets common to every models

`config` a list for controlling the optimization

**Method** `predict()`**:** Predict group of new samples

*Usage:*

```
PLNmixturefit$predict(
  newdata,
  type = c("posterior", "response", "position"),
  prior = matrix(rep(1/self$k, self$k), nrow(newdata), self$k, byrow = TRUE),
  control = PLNmixture_param(),
  envir = parent.frame()
)
```

*Arguments:*

`newdata` A data frame in which to look for variables, offsets and counts with which to predict.

`type` The type of prediction required. The default `posterior` are posterior probabilities for each group , `response` is the group with maximal posterior probability and `latent` is the averaged latent coordinate (without offset and nor covariate effects), with weights equal to the posterior probabilities.

`prior` User-specified prior group probabilities in the new data. The default uses a uniform prior.

`control` a list-like structure for controlling the fit. See `PLNmixture_param()` for details.

`envir` Environment in which the prediction is evaluated

**Method** `plot_clustering_data()`**:** Plot the matrix of expected mean counts (without offsets, without covariate effects) reordered according the inferred clustering

*Usage:*

```
PLNmixturefit$plot_clustering_data(
  main = "Expected counts reorder by clustering",
  plot = TRUE,
  log_scale = TRUE
)
```

*Arguments:*

`main` character. A title for the plot. An hopefully appropriate title will be used by default.

`plot` logical. Should the plot be displayed or sent back as `ggplot2::ggplot` object

`log_scale` logical. Should the color scale values be log-transform before plotting? Default is `TRUE`.

*Returns:* a `ggplot2::ggplot` graphic

**Method** `plot_clustering_pca()`: Plot the individual map of a PCA performed on the latent coordinates, where individuals are colored according to the memberships

*Usage:*

```
PLNmixturefit$plot_clustering_pca(
  main = "Clustering labels in Individual Factor Map",
  plot = TRUE
)
```

*Arguments:*

`main` character. A title for the plot. An hopefully appropriate title will be used by default.

`plot` logical. Should the plot be displayed or sent back as `ggplot2::ggplot` object

*Returns:* a `ggplot2::ggplot` graphic

**Method** `postTreatment()`: Update fields after optimization

*Usage:*

```
PLNmixturefit$postTreatment(
  responses,
  covariates,
  offsets,
  weights,
  config_post,
  config_optim,
  nullModel
)
```

*Arguments:*

`responses` the matrix of responses common to every models

`covariates` the matrix of covariates common to every models

`offsets` the matrix of offsets common to every models

`weights` an optional vector of observation weights to be used in the fitting process.

`config_post` a list for controlling the post-treatment

`config_optim` a list for controlling the optimization during the post-treatment computations

`nullModel` null model used for approximate R2 computations. Defaults to a GLM model with same design matrix but not latent variable.

**Method** `show()`: User friendly print method

*Usage:*

```
PLNmixturefit$show()
```

**Method** `print()`: User friendly print method

*Usage:*

```
PLNmixturefit$print()
```

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
PLNmixturefit$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

The function `PLNmixture`, the class `PLNmixturefamily`

---

`PLNmixture_param` *Control of a PLNmixture fit*

## Description

Helper to define list of parameters to control the PLNmixture fit. All arguments have defaults.

## Usage

```
PLNmixture_param(
  backend = "nlopt",
  trace = 1,
  covariance = "spherical",
  init_cl = "kmeans",
  smoothing = "both",
  config_optim = list(),
  config_post = list(),
  inception = NULL
)
```

## Arguments

| | |
|---|---|
| `backend` | optimization back used, either "nlopt" or "torch". Default is "nlopt" |
| `trace` | a integer for verbosity. |
| `covariance` | character setting the model for the covariance matrices of the mixture components. Either "full", "diagonal" or "spherical". Default is "spherical". |
| `init_cl` | The initial clustering to apply. Either, 'kmeans', CAH' or a user defined clustering given as a list of clusterings, the size of which is equal to the number of clusters considered. Default is 'kmeans'. |
| `smoothing` | The smoothing to apply. Either, 'none', forward', 'backward' or 'both'. Default is 'both'. |
| `config_optim` | a list for controlling the optimizer (either "nlopt" or "torch" backend). See details |
| `config_post` | a list for controlling the post-treatments (optional bootstrap, jackknife, R2, etc.). |
| `inception` | Set up the parameters initialization: by default, the model is initialized with a multivariate linear model applied on log-transformed data, and with the same formula as the one provided by the user. However, the user can provide a PLNfit (typically obtained from a previous fit), which sometimes speeds up the inference. |

## Details

See `PLN_param()` for a full description of the generic optimization parameters. PLNmixture_param() also has additional parameters controlling the optimization due the inner-outer loop structure of the optimizer:

- "ftol_out" outer solver stops when an optimization step changes the objective function by less than xtol multiplied by the absolute value of the parameter. Default is 1e-6
- "maxit_out" outer solver stops when the number of iteration exceeds maxit_out. Default is 50
- "it_smoothing" number of the iterations of the smoothing procedure. Default is 1.

## Value

list of parameters configuring the fit.

## See Also

`PLN_param()`

---

# `PLNnetwork` *Sparse Poisson lognormal model for network inference*

## Description

Perform sparse inverse covariance estimation for the Zero Inflated Poisson lognormal model using a variational algorithm. Iterate over a range of logarithmically spaced sparsity parameter values. Use the (g)lm syntax to specify the model (including covariates and offsets).

## Usage

```
PLNnetwork(
  formula,
  data,
  subset,
  weights,
  penalties = NULL,
  control = PLNnetwork_param()
)
```

## Arguments

| | |
|---|---|
| `formula` | an object of class "formula": a symbolic description of the model to be fitted. |
| `data` | an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which lm is called. |

| | |
|---|---|
| `subset` | an optional vector specifying a subset of observations to be used in the fitting process. |
| `weights` | an optional vector of observation weights to be used in the fitting process. |
| `penalties` | an optional vector of positive real number controlling the level of sparsity of the underlying network. if NULL (the default), will be set internally. See `PLNnetwork_param()` for additional tuning of the penalty. |
| `control` | a list-like structure for controlling the optimization, with default generated by `PLNnetwork_param()`. See the corresponding documentation for details; |

## Value

an R6 object with class `PLNnetworkfamily`, which contains a collection of models with class `PLNnetworkfit`

## See Also

The classes `PLNnetworkfamily` and `PLNnetworkfit`, and the and the configuration function `PLNnetwork_param()`.

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
fits <- PLNnetwork(Abundance ~ 1, data = trichoptera)
```

---

# `PLNnetworkfamily` *An R6 Class to represent a collection of* `PLNnetworkfit`*s*

## Description

The function `PLNnetwork()` produces an instance of this class.

This class comes with a set of methods mostly used to compare network fits (in terms of goodness of fit) or extract one from the family (based on penalty parameter and/or goodness of it). See the documentation for `getBestModel()`, `getModel()` and plot() for the user-facing ones.

## Super classes

`PLNmodels::PLNfamily` -> `PLNmodels::Networkfamily` -> `PLNnetworkfamily`

## Methods

### Public methods:

- `PLNnetworkfamily$new()`
- `PLNnetworkfamily$stability_selection()`
- `PLNnetworkfamily$clone()`

**Method** `new()`**:** Initialize all models in the collection

*Usage:*

```
PLNnetworkfamily$new(penalties, data, control)
```

*Arguments:*

`penalties` a vector of positive real number controlling the level of sparsity of the underlying network.

`data` a named list used internally to carry the data matrices

`control` a list for controlling the optimization.

*Returns:* Update current `PLNnetworkfit` with smart starting values

**Method** `stability_selection()`**:** Compute the stability path by stability selection

*Usage:*

```
PLNnetworkfamily$stability_selection(
  subsamples = NULL,
  control = PLNnetwork_param()
)
```

*Arguments:*

`subsamples` a list of vectors describing the subsamples. The number of vectors (or list length) determines the number of subsamples used in the stability selection. Automatically set to 20 subsamples with size `10*sqrt(n)` if `n` >= 144 and `0.8*n` otherwise following Liu et al. (2010) recommendations.

`control` a list controlling the main optimization process in each call to `PLNnetwork()`. See `PLNnetwork()` and `PLN_param()` for details.

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNnetworkfamily$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

The function `PLNnetwork()`, the class `PLNnetworkfit`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
fits <- PLNnetwork(Abundance ~ 1, data = trichoptera)
class(fits)
```

---

## `PLNnetworkfit` *An R6 Class to represent a PLNfit in a sparse inverse covariance framework*

---

### Description

The function `PLNnetwork()` produces a collection of models which are instances of object with class `PLNnetworkfit`. This class comes with a set of methods, some of them being useful for the user: See the documentation for `plot()` and methods inherited from `PLNfit`.

### Super classes

`PLNmodels::PLNfit` -> `PLNmodels::PLNfit_fixedcov` -> `PLNnetworkfit`

### Active bindings

`vcov_model` character: the model used for the residual covariance

`penalty` the global level of sparsity in the current model

`penalty_weights` a matrix of weights controlling the amount of penalty element-wise.

`n_edges` number of edges if the network (non null coefficient of the sparse precision matrix)

`nb_param` number of parameters in the current PLN model

`pen_loglik` variational lower bound of the l1-penalized loglikelihood

`EBIC` variational lower bound of the EBIC

`density` proportion of non-null edges in the network

`criteria` a vector with loglik, penalized loglik, BIC, EBIC, ICL, R_squared, number of parameters, number of edges and graph density

### Methods

**Public methods:**

- `PLNnetworkfit$new()`
- `PLNnetworkfit$optimize()`
- `PLNnetworkfit$latent_network()`
- `PLNnetworkfit$plot_network()`
- `PLNnetworkfit$show()`
- `PLNnetworkfit$clone()`

**Method `new()`:** Initialize a `PLNnetworkfit` object

*Usage:*

```
PLNnetworkfit$new(data, control)
```

*Arguments:*

`data` a named list used internally to carry the data matrices

`control` a list for controlling the optimization.

**Method** `optimize()`**:** Call to the C++ optimizer and update of the relevant fields

*Usage:*

```
PLNnetworkfit$optimize(data, config)
```

*Arguments:*

`data` a named list used internally to carry the data matrices

`config` a list for controlling the optimization

**Method** `latent_network()`**:** Extract interaction network in the latent space

*Usage:*

```
PLNnetworkfit$latent_network(type = c("partial_cor", "support", "precision"))
```

*Arguments:*

`type` edge value in the network. Can be "support" (binary edges), "precision" (coefficient of the precision matrix) or "partial_cor" (partial correlation between species)

*Returns:* a square matrix of size `PLNnetworkfit$n`

**Method** `plot_network()`**:** plot the latent network.

*Usage:*

```
PLNnetworkfit$plot_network(
  type = c("partial_cor", "support"),
  output = c("igraph", "corrplot"),
  edge.color = c("#F8766D", "#00BFC4"),
  remove.isolated = FALSE,
  node.labels = NULL,
  layout = layout_in_circle,
  plot = TRUE
)
```

*Arguments:*

`type` edge value in the network. Either "precision" (coefficient of the precision matrix) or "partial_cor" (partial correlation between species).

`output` Output type. Either `igraph` (for the network) or `corrplot` (for the adjacency matrix)

`edge.color` Length 2 color vector. Color for positive/negative edges. Default is `c("#F8766D", "#00BFC4")`. Only relevant for igraph output.

`remove.isolated` if `TRUE`, isolated node are remove before plotting. Only relevant for igraph output.

`node.labels` vector of character. The labels of the nodes. The default will use the column names ot the response matrix.

`layout` an optional igraph layout. Only relevant for igraph output.

`plot` logical. Should the final network be displayed or only sent back to the user. Default is `TRUE`.

**Method** `show()`**:** User friendly print method

*Usage:*

```
PLNnetworkfit$show()
```

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNnetworkfit$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

The function `PLNnetwork()`, the class `PLNnetworkfamily`

## Examples

```
## Not run:
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
nets <- PLNnetwork(Abundance ~ 1, data = trichoptera)
myPLNnet <- getBestModel(nets)
class(myPLNnet)
print(myPLNnet)

## End(Not run)
```

---

# `PLNnetwork_param` *Control of PLNnetwork fit*

## Description

Helper to define list of parameters to control the PLN fit. All arguments have defaults.

## Usage

```
PLNnetwork_param(
  backend = c("nlopt", "torch"),
  inception_cov = c("full", "spherical", "diagonal"),
  trace = 1,
  n_penalties = 30,
  min_ratio = 0.1,
  penalize_diagonal = TRUE,
  penalty_weights = NULL,
  config_post = list(),
  config_optim = list(),
  inception = NULL
)
```

### Arguments

| | |
|---|---|
| `backend` | optimization back used, either "nlopt" or "torch". Default is "nlopt" |
| `inception_cov` | Covariance structure used for the inception model used to initialize the PLN-family. Defaults to "full" and can be constrained to "diagonal" and "spherical". |
| `trace` | a integer for verbosity. |
| `n_penalties` | an integer that specifies the number of values for the penalty grid when internally generated. Ignored when penalties is non `NULL` |
| `min_ratio` | the penalty grid ranges from the minimal value that produces a sparse to this value multiplied by `min_ratio`. Default is 0.1. |
| `penalize_diagonal` | boolean: should the diagonal terms be penalized in the graphical-Lasso? Default is `TRUE` |
| `penalty_weights` | either a single or a list of p x p matrix of weights (default: all weights equal to 1) to adapt the amount of shrinkage to each pairs of node. Must be symmetric with positive values. |
| `config_post` | a list for controlling the post-treatment (optional bootstrap, jackknife, R2, etc). |
| `config_optim` | a list for controlling the optimizer (either "nlopt" or "torch" backend). See details |
| `inception` | Set up the parameters initialization: by default, the model is initialized with a multivariate linear model applied on log-transformed data, and with the same formula as the one provided by the user. However, the user can provide a PLNfit (typically obtained from a previous fit), which sometimes speeds up the inference. |

### Details

See `PLN_param()` for a full description of the generic optimization parameters. PLNnetwork_param() also has two additional parameters controlling the optimization due the inner-outer loop structure of the optimizer:

- "ftol_out" outer solver stops when an optimization step changes the objective function by less than ftol multiplied by the absolute value of the parameter. Default is 1e-6
- "maxit_out" outer solver stops when the number of iteration exceeds maxit_out. Default is 50

### Value

list of parameters configuring the fit.

### See Also

`PLN_param()`

---

| PLNPCA | *Poisson lognormal model towards Principal Component Analysis* |
|---|---|

---

## Description

Fit the PCA variants of the Poisson lognormal with a variational algorithm. Use the (g)lm syntax for model specification (covariates, offsets).

## Usage

```
PLNPCA(formula, data, subset, weights, ranks = 1:5, control = PLNPCA_param())
```

## Arguments

| | |
|---|---|
| `formula` | an object of class "formula": a symbolic description of the model to be fitted. |
| `data` | an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which lm is called. |
| `subset` | an optional vector specifying a subset of observations to be used in the fitting process. |
| `weights` | an optional vector of observation weights to be used in the fitting process. |
| `ranks` | a vector of integer containing the successive ranks (or number of axes to be considered) |
| `control` | a list-like structure for controlling the optimization, with default generated by `PLNPCA_param()`. See the associated documentation. for details. |

## Value

an R6 object with class `PLNPCAfamily`, which contains a collection of models with class `PLNPCAfit`

## See Also

The classes `PLNPCAfamily` and `PLNPCAfit`, and the configuration function `PLNPCA_param()`.

## Examples

```
#' ## Use future to dispatch the computations on 2 workers
## Not run:
future::plan("multisession", workers = 2)

## End(Not run)

data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPCA <- PLNPCA(Abundance ~ 1 + offset(log(Offset)), data = trichoptera, ranks = 1:5)
```

```
# Shut down parallel workers
## Not run:
future::plan("sequential")

## End(Not run)
```

---

`PLNPCAfamily` *An R6 Class to represent a collection of PLNPCAfit*

---

## Description

The function `PLNPCA()` produces an instance of this class.

This class comes with a set of methods, some of them being useful for the user: See the documentation for `getBestModel()`, `getModel()` and `plot()`.

## Super class

`PLNmodels::PLNfamily` -> `PLNPCAfamily`

## Active bindings

`ranks` the dimensions of the successively fitted models

## Methods

### Public methods:

- `PLNPCAfamily$new()`
- `PLNPCAfamily$optimize()`
- `PLNPCAfamily$getModel()`
- `PLNPCAfamily$getBestModel()`
- `PLNPCAfamily$plot()`
- `PLNPCAfamily$show()`
- `PLNPCAfamily$clone()`

**Method** `new()`**:** Initialize all models in the collection.

*Usage:*

```
PLNPCAfamily$new(
  ranks,
  responses,
  covariates,
  offsets,
  weights,
  formula,
  control
)
```

*Arguments:*

`ranks` the dimensions of the successively fitted models

`responses` the matrix of responses common to every models

`covariates` the matrix of covariates common to every models

`offsets` the matrix of offsets common to every models

`weights` the vector of observation weights

`formula` model formula used for fitting, extracted from the formula in the upper-level call

`control` list controlling the optimization and the model

**Method** `optimize()`: Call to the C++ optimizer on all models of the collection

*Usage:*

```
PLNPCAfamily$optimize(config)
```

*Arguments:*

`config` list controlling the optimization.

**Method** `getModel()`: Extract model from collection and add "PCA" class for compatibility with `factoextra::fviz()`

*Usage:*

```
PLNPCAfamily$getModel(var, index = NULL)
```

*Arguments:*

`var` value of the parameter (rank for PLNPCA, sparsity for PLNnetwork) that identifies the model to be extracted from the collection. If no exact match is found, the model with closest parameter value is returned with a warning.

`index` Integer index of the model to be returned. Only the first value is taken into account.

*Returns:* a `PLNPCAfit` object

**Method** `getBestModel()`: Extract best model in the collection

*Usage:*

```
PLNPCAfamily$getBestModel(crit = c("ICL", "BIC"))
```

*Arguments:*

`crit` a character for the criterion used to performed the selection. Either "ICL", "BIC". Default is `ICL`

*Returns:* a `PLNPCAfit` object

**Method** `plot()`: Lineplot of selected criteria for all models in the collection

*Usage:*

```
PLNPCAfamily$plot(criteria = c("loglik", "BIC", "ICL"), reverse = FALSE)
```

*Arguments:*

`criteria` A valid model selection criteria for the collection of models. Any of "loglik", "BIC" or "ICL" (all).

`reverse` A logical indicating whether to plot the value of the criteria in the "natural" direction (loglik - penalty) or in the "reverse" direction (-2 loglik + penalty). Default to FALSE, i.e use the natural direction, on the same scale as the log-likelihood.

*Returns:* A `ggplot2::ggplot` object

**Method** `show()`**:** User friendly print method

*Usage:*

```
PLNPCAfamily$show()
```

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNPCAfamily$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

The function `PLNPCA()`, the class `PLNPCAfit()`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPCAs <- PLNPCA(Abundance ~ 1 + offset(log(Offset)), data = trichoptera, ranks = 1:5)
class(myPCAs)
```

---

# `PLNPCAfit` *An R6 Class to represent a PLNfit in a PCA framework*

## Description

The function `PLNPCA()` produces a collection of models which are instances of object with class `PLNPCAfit`. This class comes with a set of methods, some of them being useful for the user: See the documentation for the methods inherited by `PLNfit` and the `plot()` methods for PCA visualization

## Super class

`PLNmodels::PLNfit` -> `PLNPCAfit`

## Active bindings

`rank` the dimension of the current model

`vcov_model` character: the model used for the residual covariance

`nb_param` number of parameters in the current PLN model

`entropy` entropy of the variational distribution

`latent_pos` a matrix: values of the latent position vector (Z) without covariates effects or offset

`model_par` a list with the matrices associated with the estimated parameters of the pPCA model: B (covariates), Sigma (covariance), Omega (precision) and C (loadings) `percent_var` the percent of variance explained by each axis

`corr_circle` a matrix of correlations to plot the correlation circles

`scores` a matrix of scores to plot the individual factor maps (a.k.a. principal components)

`rotation` a matrix of rotation of the latent space

`eig` description of the eigenvalues, similar to percent_var but for use with external methods

`var` a list of data frames with PCA results for the variables: `coord` (coordinates of the variables), `cor` (correlation between variables and dimensions), `cos2` (Cosine of the variables) and `contrib` (contributions of the variable to the axes)

`ind` a list of data frames with PCA results for the individuals: `coord` (coordinates of the individuals), `cos2` (Cosine of the individuals), `contrib` (contributions of individuals to an axis inertia) and `dist` (distance of individuals to the origin).

`call` Hacky binding for compatibility with factoextra functions

## Methods

### Public methods:

- `PLNPCAfit$new()`
- `PLNPCAfit$update()`
- `PLNPCAfit$optimize()`
- `PLNPCAfit$optimize_vestep()`
- `PLNPCAfit$project()`
- `PLNPCAfit$setVisualization()`
- `PLNPCAfit$postTreatment()`
- `PLNPCAfit$plot_individual_map()`
- `PLNPCAfit$plot_correlation_circle()`
- `PLNPCAfit$plot_PCA()`
- `PLNPCAfit$show()`
- `PLNPCAfit$clone()`

**Method** `new()`**:** Initialize a `PLNPCAfit` object

*Usage:*

```
PLNPCAfit$new(rank, responses, covariates, offsets, weights, formula, control)
```

*Arguments:*

`rank` rank of the PCA (or equivalently, dimension of the latent space)

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in `PLNfamily`

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in `PLNfamily`

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in `PLNfamily`

`weights` an optional vector of observation weights to be used in the fitting process.

`formula` model formula used for fitting, extracted from the formula in the upper-level call

`control` a list for controlling the optimization. See details.

**Method** `update()`: Update a `PLNPCAfit` object

*Usage:*

```
PLNPCAfit$update(
  B = NA,
  Sigma = NA,
  Omega = NA,
  C = NA,
  M = NA,
  S = NA,
  Z = NA,
  A = NA,
  Ji = NA,
  R2 = NA,
  monitoring = NA
)
```

*Arguments:*

| | |
|---|---|
| `B` | matrix of regression matrix |
| `Sigma` | variance-covariance matrix of the latent variables |
| `Omega` | precision matrix of the latent variables. Inverse of Sigma. |
| `C` | matrix of PCA loadings (in the latent space) |
| `M` | matrix of mean vectors for the variational approximation |
| `S` | matrix of variance vectors for the variational approximation |
| `Z` | matrix of latent vectors (includes covariates and offset effects) |
| `A` | matrix of fitted values |
| `Ji` | vector of variational lower bounds of the log-likelihoods (one value per sample) |
| `R2` | approximate R^2 goodness-of-fit criterion |
| `monitoring` | a list with optimization monitoring quantities |

*Returns:* Update the current `PLNPCAfit` object

**Method** `optimize()`: Call to the C++ optimizer and update of the relevant fields

*Usage:*

```
PLNPCAfit$optimize(responses, covariates, offsets, weights, config)
```

*Arguments:*

| | |
|---|---|
| `responses` | the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in `PLNfamily` |
| `covariates` | design matrix (called X in the model). Will usually be extracted from the corresponding field in `PLNfamily` |
| `offsets` | offset matrix (called O in the model). Will usually be extracted from the corresponding field in `PLNfamily` |
| `weights` | an optional vector of observation weights to be used in the fitting process. |
| `config` | part of the `control` argument which configures the optimizer |

**Method** `optimize_vestep()`: Result of one call to the VE step of the optimization procedure: optimal variational parameters (M, S) and corresponding log likelihood values for fixed model parameters (C, B). Intended to position new data in the latent space for further use with PCA.

*Usage:*

```
PLNPCAfit$optimize_vestep(
  covariates,
  offsets,
  responses,
  weights = rep(1, self$n),
  control = PLNPCA_param(backend = "nlopt")
)
```

*Arguments:*

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in `PLNfamily`

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in `PLNfamily`

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in `PLNfamily`

`weights` an optional vector of observation weights to be used in the fitting process.

`control` a list for controlling the optimization. See details.

*Returns:* A list with three components:

- the matrix `M` of variational means,
- the matrix `S2` of variational variances
- the vector `log.lik` of (variational) log-likelihood of each new observation

**Method** `project()`**:** Project new samples into the PCA space using one VE step

*Usage:*

```
PLNPCAfit$project(newdata, control = PLNPCA_param(), envir = parent.frame())
```

*Arguments:*

`newdata` A data frame in which to look for variables, offsets and counts with which to predict.

`control` a list for controlling the optimization. See `PLN()` for details.

`envir` Environment in which the projection is evaluated

*Returns:*

- the named matrix of scores for the newdata, expressed in the same coordinate system as `self$scores`

**Method** `setVisualization()`**:** Compute PCA scores in the latent space and update corresponding fields.

*Usage:*

```
PLNPCAfit$setVisualization(scale.unit = FALSE)
```

*Arguments:*

`scale.unit` Logical. Should PCA scores be rescaled to have unit variance

**Method** `postTreatment()`**:** Update R2, fisher, std_err fields and set up visualization

*Usage:*

```
PLNPCAfit$postTreatment(
  responses,
  covariates,
  offsets,
  weights,
  config_post,
  config_optim,
  nullModel
)
```

*Arguments:*

`responses` the matrix of responses (called Y in the model). Will usually be extracted from the corresponding field in `PLNfamily`

`covariates` design matrix (called X in the model). Will usually be extracted from the corresponding field in `PLNfamily`

`offsets` offset matrix (called O in the model). Will usually be extracted from the corresponding field in `PLNfamily`

`weights` an optional vector of observation weights to be used in the fitting process.

`config_post` a list for controlling the post-treatments (optional bootstrap, jackknife, R2, etc.). See details

`config_optim` a list for controlling the optimizer (either "nlopt" or "torch" backend). See details

`nullModel` null model used for approximate R2 computations. Defaults to a GLM model with same design matrix but not latent variable.

*Details:* The list of parameters `config_post` controls the post-treatment processing, with the following entries:

- jackknife boolean indicating whether jackknife should be performed to evaluate bias and variance of the model parameters. Default is FALSE.
- bootstrap integer indicating the number of bootstrap resamples generated to evaluate the variance of the model parameters. Default is 0 (inactivated).
- variational_var boolean indicating whether variational Fisher information matrix should be computed to estimate the variance of the model parameters (highly underestimated). Default is FALSE.
- rsquared boolean indicating whether approximation of R2 based on deviance should be computed. Default is TRUE
- trace integer for verbosity. should be > 1 to see output in post-treatments

**Method** `plot_individual_map()`**:** Plot the factorial map of the PCA

*Usage:*

```
PLNPCAfit$plot_individual_map(
  axes = 1:min(2, self$rank),
  main = "Individual Factor Map",
  plot = TRUE,
  cols = "default"
)
```

*Arguments:*

`axes` numeric, the axes to use for the plot when map = "individual" or "variable". Default it c(1,min(rank))

`main` character. A title for the single plot (individual or variable factor map). If NULL (the default), an hopefully appropriate title will be used.

`plot` logical. Should the plot be displayed or sent back as ggplot object

`cols` a character, factor or numeric to define the color associated with the individuals. By default, all individuals receive the default color of the current palette.

*Returns:* a `ggplot2::ggplot` graphic

**Method** `plot_correlation_circle()`**:** Plot the correlation circle of a specified axis for a `PLNLDAfit` object

*Usage:*

```
PLNPCAfit$plot_correlation_circle(
  axes = 1:min(2, self$rank),
  main = "Variable Factor Map",
  cols = "default",
  plot = TRUE
)
```

*Arguments:*

`axes` numeric, the axes to use for the plot when map = "individual" or "variable". Default it c(1,min(rank))

`main` character. A title for the single plot (individual or variable factor map). If NULL (the default), an hopefully appropriate title will be used.

`cols` a character, factor or numeric to define the color associated with the variables. By default, all variables receive the default color of the current palette.

`plot` logical. Should the plot be displayed or sent back as ggplot object

*Returns:* a `ggplot2::ggplot` graphic

**Method** `plot_PCA()`**:** Plot a summary of the `PLNPCAfit` object

*Usage:*

```
PLNPCAfit$plot_PCA(
  nb_axes = min(3, self$rank),
  ind_cols = "ind_cols",
  var_cols = "var_cols",
  plot = TRUE
)
```

*Arguments:*

`nb_axes` scalar: the number of axes to be considered when map = "both". The default is min(3,rank).

`ind_cols` a character, factor or numeric to define the color associated with the individuals. By default, all variables receive the default color of the current palette.

`var_cols` a character, factor or numeric to define the color associated with the variables. By default, all variables receive the default color of the current palette.

`plot` logical. Should the plot be displayed or sent back as ggplot object

*Returns:* a `grob` object

**Method** `show()`**:** User friendly print method

*Usage:*

```
PLNPCAfit$show()
```

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PLNPCAfit$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

The function `PLNPCA`, the class `PLNPCAfamily`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPCAs <- PLNPCA(Abundance ~ 1 + offset(log(Offset)), data = trichoptera, ranks = 1:5)
myPCA <- getBestModel(myPCAs)
class(myPCA)
print(myPCA)
```

---

# `PLNPCA_param` *Control of PLNPCA fit*

## Description

Helper to define list of parameters to control the PLNPCA fit. All arguments have defaults.

## Usage

```
PLNPCA_param(
  backend = "nlopt",
  trace = 1,
  config_optim = list(),
  config_post = list(),
  inception = NULL
)
```

## Arguments

| | |
|---|---|
| `backend` | optimization back used, either "nlopt" or "torch". Default is "nlopt" |
| `trace` | a integer for verbosity. |
| `config_optim` | a list for controlling the optimizer (either "nlopt" or "torch" backend). See details |
| `config_post` | a list for controlling the post-treatments (optional bootstrap, jackknife, R2, etc.). See details |
| `inception` | Set up the parameters initialization: by default, the model is initialized with a multivariate linear model applied on log-transformed data, and with the same formula as the one provided by the user. However, the user can provide a PLNfit (typically obtained from a previous fit), which sometimes speeds up the inference. |

## Details

The list of parameters `config_optim` controls the optimizers. When "nlopt" is chosen the following entries are relevant

- "algorithm" the optimization method used by NLOPT among LD type, e.g. "CCSAQ", "MMA", "LBFGS". See NLOPT documentation for further details. Default is "CCSAQ".
- "maxeval" stop when the number of iteration exceeds maxeval. Default is 10000
- "ftol_rel" stop when an optimization step changes the objective function by less than ftol multiplied by the absolute value of the parameter. Default is 1e-8
- "xtol_rel" stop when an optimization step changes every parameters by less than xtol multiplied by the absolute value of the parameter. Default is 1e-6
- "ftol_abs" stop when an optimization step changes the objective function by less than ftol_abs. Default is 0.0 (disabled)
- "xtol_abs" stop when an optimization step changes every parameters by less than xtol_abs. Default is 0.0 (disabled)
- "maxtime" stop when the optimization time (in seconds) exceeds maxtime. Default is -1 (disabled)

When "torch" backend is used (only for PLN and PLNLDA for now), the following entries are relevant:

- "algorithm" the optimizer used by torch among RPROP (default), RMSPROP, ADAM and ADAGRAD
- "maxeval" stop when the number of iteration exceeds maxeval. Default is 10 000
- "numepoch" stop training once this number of epochs exceeds numepoch. Set to -1 to enable infinite training. Default is 1 000
- "num_batch" number of batches to use during training. Defaults to 1 (use full dataset at each epoch)
- "ftol_rel" stop when an optimization step changes the objective function by less than ftol multiplied by the absolute value of the parameter. Default is 1e-8

- "xtol_rel" stop when an optimization step changes every parameters by less than xtol multiplied by the absolute value of the parameter. Default is 1e-6
- "lr" learning rate. Default is 0.1.
- "momentum" momentum factor. Default is 0 (no momentum). Only used in RMSPROP
- "weight_decay" Weight decay penalty. Default is 0 (no decay). Not used in RPROP
- "step_sizes" pair of minimal (default: 1e-6) and maximal (default: 50) allowed step sizes. Only used in RPROP
- "etas" pair of multiplicative increase and decrease factors. Default is (0.5, 1.2). Only used in RPROP
- "centered" if TRUE, compute the centered RMSProp where the gradient is normalized by an estimation of its variance weight_decay (L2 penalty). Default to FALSE. Only used in RMSPROP

The list of parameters `config_post` controls the post-treatment processing (for most `PLN*()` functions), with the following entries (defaults may vary depending on the specific function, check `config_post_default_*` for defaults values):

- jackknife boolean indicating whether jackknife should be performed to evaluate bias and variance of the model parameters. Default is FALSE.
- bootstrap integer indicating the number of bootstrap resamples generated to evaluate the variance of the model parameters. Default is 0 (inactivated).
- variational_var boolean indicating whether variational Fisher information matrix should be computed to estimate the variance of the model parameters (highly underestimated). Default is FALSE.
- sandwich_var boolean indicating whether sandwich estimation should be used to estimate the variance of the model parameters (highly underestimated). Default is FALSE.
- rsquared boolean indicating whether approximation of R2 based on deviance should be computed. Default is TRUE

## Value

list of parameters configuring the fit.

---

## `PLN_param` *Control of a PLN fit*

## Description

Helper to define list of parameters to control the PLN fit. All arguments have defaults.

## Usage

```
PLN_param(
  backend = c("nlopt", "torch"),
  trace = 1,
  covariance = c("full", "diagonal", "spherical", "fixed"),
  Omega = NULL,
  config_post = list(),
  config_optim = list(),
  inception = NULL
)
```

## Arguments

| | |
|---|---|
| `backend` | optimization back used, either "nlopt" or "torch". Default is "nlopt" |
| `trace` | a integer for verbosity. |
| `covariance` | character setting the model for the covariance matrix. Either "full", "diagonal", "spherical" or "fixed". Default is "full". |
| `Omega` | precision matrix of the latent variables. Inverse of Sigma. Must be specified if `covariance` is "fixed" |
| `config_post` | a list for controlling the post-treatments (optional bootstrap, jackknife, R2, etc.). See details |
| `config_optim` | a list for controlling the optimizer (either "nlopt" or "torch" backend). See details |
| `inception` | Set up the parameters initialization: by default, the model is initialized with a multivariate linear model applied on log-transformed data, and with the same formula as the one provided by the user. However, the user can provide a PLNfit (typically obtained from a previous fit), which sometimes speeds up the inference. |

## Details

The list of parameters `config_optim` controls the optimizers. When "nlopt" is chosen the following entries are relevant

- "algorithm" the optimization method used by NLOPT among LD type, e.g. "CCSAQ", "MMA", "LBFGS". See NLOPT documentation for further details. Default is "CCSAQ".
- "maxeval" stop when the number of iteration exceeds maxeval. Default is 10000
- "ftol_rel" stop when an optimization step changes the objective function by less than ftol multiplied by the absolute value of the parameter. Default is 1e-8
- "xtol_rel" stop when an optimization step changes every parameters by less than xtol multiplied by the absolute value of the parameter. Default is 1e-6
- "ftol_abs" stop when an optimization step changes the objective function by less than ftol_abs. Default is 0.0 (disabled)
- "xtol_abs" stop when an optimization step changes every parameters by less than xtol_abs. Default is 0.0 (disabled)

- "maxtime" stop when the optimization time (in seconds) exceeds maxtime. Default is -1 (disabled)

When "torch" backend is used (only for PLN and PLNLDA for now), the following entries are relevant:

- "algorithm" the optimizer used by torch among RPROP (default), RMSPROP, ADAM and ADAGRAD
- "maxeval" stop when the number of iteration exceeds maxeval. Default is 10 000
- "numepoch" stop training once this number of epochs exceeds numepoch. Set to -1 to enable infinite training. Default is 1 000
- "num_batch" number of batches to use during training. Defaults to 1 (use full dataset at each epoch)
- "ftol_rel" stop when an optimization step changes the objective function by less than ftol multiplied by the absolute value of the parameter. Default is 1e-8
- "xtol_rel" stop when an optimization step changes every parameters by less than xtol multiplied by the absolute value of the parameter. Default is 1e-6
- "lr" learning rate. Default is 0.1.
- "momentum" momentum factor. Default is 0 (no momentum). Only used in RMSPROP
- "weight_decay" Weight decay penalty. Default is 0 (no decay). Not used in RPROP
- "step_sizes" pair of minimal (default: 1e-6) and maximal (default: 50) allowed step sizes. Only used in RPROP
- "etas" pair of multiplicative increase and decrease factors. Default is (0.5, 1.2). Only used in RPROP
- "centered" if TRUE, compute the centered RMSProp where the gradient is normalized by an estimation of its variance weight_decay (L2 penalty). Default to FALSE. Only used in RMSPROP

The list of parameters `config_post` controls the post-treatment processing (for most `PLN*()` functions), with the following entries (defaults may vary depending on the specific function, check `config_post_default_*` for defaults values):

- jackknife boolean indicating whether jackknife should be performed to evaluate bias and variance of the model parameters. Default is FALSE.
- bootstrap integer indicating the number of bootstrap resamples generated to evaluate the variance of the model parameters. Default is 0 (inactivated).
- variational_var boolean indicating whether variational Fisher information matrix should be computed to estimate the variance of the model parameters (highly underestimated). Default is FALSE.
- sandwich_var boolean indicating whether sandwich estimation should be used to estimate the variance of the model parameters (highly underestimated). Default is FALSE.
- rsquared boolean indicating whether approximation of R2 based on deviance should be computed. Default is TRUE

## Value

list of parameters configuring the fit.

---

## `plot.Networkfamily` *Display various outputs (goodness-of-fit criteria, robustness, diagnostic) associated with a collection of network fits (either `PLNnetworkfamily` or `ZIPLNnetworkfamily`)*

---

### Description

Display various outputs (goodness-of-fit criteria, robustness, diagnostic) associated with a collection of network fits (either `PLNnetworkfamily` or `ZIPLNnetworkfamily`)

### Usage

```
## S3 method for class 'Networkfamily'
plot(
  x,
  type = c("criteria", "stability", "diagnostic"),
  criteria = c("loglik", "pen_loglik", "BIC", "EBIC"),
  reverse = FALSE,
  log.x = TRUE,
  stability = 0.9,
  ...
)

## S3 method for class 'PLNnetworkfamily'
plot(
  x,
  type = c("criteria", "stability", "diagnostic"),
  criteria = c("loglik", "pen_loglik", "BIC", "EBIC"),
  reverse = FALSE,
  log.x = TRUE,
  stability = 0.9,
  ...
)

## S3 method for class 'ZIPLNnetworkfamily'
plot(
  x,
  type = c("criteria", "stability", "diagnostic"),
  criteria = c("loglik", "pen_loglik", "BIC", "EBIC"),
  reverse = FALSE,
  log.x = TRUE,
  stability = 0.9,
  ...
)
```

### Arguments

| | |
|---|---|
| `x` | an R6 object with class `PLNnetworkfamily` or `ZIPLNnetworkfamily` |

| | |
|---|---|
| `type` | a character, either "criteria", "stability" or "diagnostic" for the type of plot. |
| `criteria` | Vector of criteria to plot, to be selected among "loglik" (log-likelihood), "BIC", "ICL", "R_squared", "EBIC" and "pen_loglik" (penalized log-likelihood). Default is c("loglik", "pen_loglik", "BIC", "EBIC"). Only used when `type = "criteria"`. |
| `reverse` | A logical indicating whether to plot the value of the criteria in the "natural" direction (loglik - 0.5 penalty) or in the "reverse" direction (-2 loglik + penalty). Default to FALSE, i.e use the natural direction, on the same scale as the log-likelihood. |
| `log.x` | logical: should the x-axis be represented in log-scale? Default is `TRUE`. |
| `stability` | scalar: the targeted level of stability in stability plot. Default is .9. |
| `...` | additional parameters for S3 compatibility. Not used |

## Details

The BIC and ICL criteria have the form 'loglik - 1/2 * penalty' so that they are on the same scale as the model log-likelihood. You can change this direction and use the alternate form '-2*loglik + penalty', as some authors do, by setting `reverse = TRUE`.

## Value

Produces either a diagnostic plot (with `type = 'diagnostic'`), a stability plot (with `type = 'stability'`) or the evolution of the criteria of the different models considered (with `type = 'criteria'`, the default).

## Functions

- `plot(PLNnetworkfamily)`: Display various outputs associated with a collection of network fits
- `plot(ZIPLNnetworkfamily)`: Display various outputs associated with a collection of network fits

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
fits <- PLNnetwork(Abundance ~ 1, data = trichoptera)
## Not run:
plot(fits)

## End(Not run)
```

---

`plot.PLNfamily` *Display the criteria associated with a collection of PLN fits (a PLNfamily)*

---

## Description

Display the criteria associated with a collection of PLN fits (a PLNfamily)

## Usage

```
## S3 method for class 'PLNfamily'
plot(x, criteria = c("loglik", "BIC", "ICL"), reverse = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | an R6 object with class `PLNfamily` |
| `criteria` | vector of characters. The criteria to plot in c("loglik", "BIC", "ICL"). Default is c("loglik", "BIC", "ICL"). |
| `reverse` | A logical indicating whether to plot the value of the criteria in the "natural" direction (loglik - 0.5 penalty) or in the "reverse" direction (-2 loglik + penalty). Default to FALSE, i.e use the natural direction, on the same scale as the log-likelihood. |
| `...` | additional parameters for S3 compatibility. Not used |

## Details

The BIC and ICL criteria have the form 'loglik - 1/2 * penalty' so that they are on the same scale as the model log-likelihood. You can change this direction and use the alternate form '-2*loglik + penalty', as some authors do, by setting `reverse` = `TRUE`.

## Value

Produces a plot representing the evolution of the criteria of the different models considered, highlighting the best model in terms of BIC and ICL (see details).

## See Also

`plot.PLNPCAfamily()` and `plot.PLNnetworkfamily()`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPCAs <- PLNPCA(Abundance ~ 1 + offset(log(Offset)), data = trichoptera, ranks = 1:5)
## Not run:
plot(myPCAs)

## End(Not run)
```

---

`plot.PLNLDAfit` *LDA visualization (individual and/or variable factor map(s)) for a* `PLNPCAfit` *object*

---

## Description

LDA visualization (individual and/or variable factor map(s)) for a `PLNPCAfit` object

## Usage

```
## S3 method for class 'PLNLDAfit'
plot(
  x,
  map = c("both", "individual", "variable"),
  nb_axes = min(3, x$rank),
  axes = seq.int(min(2, x$rank)),
  var_cols = "var_colors",
  plot = TRUE,
  main = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | an R6 object with class PLNPCAfit |
| `map` | the type of output for the PCA visualization: either "individual", "variable" or "both". Default is "both". |
| `nb_axes` | scalar: the number of axes to be considered when map = "both". The default is min(3,rank). |
| `axes` | numeric, the axes to use for the plot when map = "individual" or "variable". Default it c(1,min(rank)) |
| `var_cols` | a character or factor to define the color associated with the variables. By default, all variables receive the default color of the current palette. |
| `plot` | logical. Should the plot be displayed or sent back as `ggplot2::ggplot` object |
| `main` | character. A title for the single plot (individual or variable factor map). If NULL (the default), an hopefully appropriate title will be used. |
| `...` | Not used (S3 compatibility). |

## Value

displays an individual and/or variable factor maps for the corresponding axes, and/or sends back a `ggplot2::ggplot` or gtable object

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLNLDA <- PLNLDA(Abundance ~ 1, grouping = Group, data = trichoptera)
## Not run:
plot(myPLNLDA, map = "individual", nb_axes = 2)

## End(Not run)
```

---

# `plot.PLNmixturefamily` *Display the criteria associated with a collection of PLNmixture fits (a PLNmixturefamily)*

## Description

Display the criteria associated with a collection of PLNmixture fits (a PLNmixturefamily)

## Usage

```
## S3 method for class 'PLNmixturefamily'
plot(
  x,
  type = c("criteria", "diagnostic"),
  criteria = c("loglik", "BIC", "ICL"),
  reverse = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | an R6 object with class `PLNmixturefamily` |
| `type` | a character, either `"criteria"` or `"diagnostic"` for the type of plot. |
| `criteria` | vector of characters. The criteria to plot in c("loglik", "BIC", "ICL"). Default is c("loglik", "BIC", "ICL"). |
| `reverse` | A logical indicating whether to plot the value of the criteria in the "natural" direction (loglik - 0.5 penalty) or in the "reverse" direction (-2 loglik + penalty). Default to FALSE, i.e use the natural direction, on the same scale as the log-likelihood. |
| `...` | additional parameters for S3 compatibility. Not used |

## Details

The BIC and ICL criteria have the form 'loglik - 1/2 * penalty' so that they are on the same scale as the model log-likelihood. You can change this direction and use the alternate form '-2*loglik + penalty', as some authors do, by setting `reverse = TRUE`.

## Value

Produces either a diagnostic plot (with `type = 'diagnostic'`) or the evolution of the criteria of the different models considered (with `type = 'criteria'`, the default).

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myMixtures <- PLNmixture(Abundance ~ 1 + offset(log(Offset)),
           data = trichoptera, control = PLNmixture_param(smoothing = "none"))
plot(myMixtures, reverse = TRUE)
```

---

# `plot.PLNmixturefit` *Mixture visualization of a* `PLNmixturefit` *object*

## Description

Represent the result of the clustering either by coloring the individual in a two-dimension PCA factor map, or by representing the expected matrix of count reorder according to the clustering.

## Usage

```
## S3 method for class 'PLNmixturefit'
plot(x, type = c("pca", "matrix"), main = NULL, plot = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `x` | an R6 object with class `PLNmixturefit` |
| `type` | character for the type of plot, either "pca", for or "matrix". Default is `"pca"`. |
| `main` | character. A title for the plot. If NULL (the default), an hopefully appropriate title will be used. |
| `plot` | logical. Should the plot be displayed or sent back as `ggplot2::ggplot` object |
| `...` | Not used (S3 compatibility). |

## Value

a `ggplot2::ggplot` graphic

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLNmixture(Abundance ~ 1 + offset(log(Offset)),
       data = trichoptera, control = PLNmixture_param(smoothing = "none")) %>% getBestModel()
## Not run:
plot(myPLN, "pca")
plot(myPLN, "matrix")

## End(Not run)
```

---

## `plot.PLNnetworkfit` *Extract and plot the network (partial correlation, support or inverse covariance) from a `PLNnetworkfit` object*

---

### Description

Extract and plot the network (partial correlation, support or inverse covariance) from a `PLNnetworkfit` object

### Usage

```
## S3 method for class 'PLNnetworkfit'
plot(
  x,
  type = c("partial_cor", "support"),
  output = c("igraph", "corrplot"),
  edge.color = c("#F8766D", "#00BFC4"),
  remove.isolated = FALSE,
  node.labels = NULL,
  layout = layout_in_circle,
  plot = TRUE,
  ...
)
```

### Arguments

| | |
|---|---|
| `x` | an R6 object with class `PLNnetworkfit` |
| `type` | character. Value of the weight of the edges in the network, either "partial_cor" (partial correlation) or "support" (binary). Default is `"partial_cor"`. |
| `output` | the type of output used: either 'igraph' or 'corrplot'. Default is `'igraph'`. |
| `edge.color` | Length 2 color vector. Color for positive/negative edges. Default is `c("#F8766D", "#00BFC4")`. Only relevant for igraph output. |
| `remove.isolated` | if `TRUE`, isolated node are remove before plotting. Only relevant for igraph output. |
| `node.labels` | vector of character. The labels of the nodes. The default will use the column names ot the response matrix. |
| `layout` | an optional igraph layout. Only relevant for igraph output. |
| `plot` | logical. Should the final network be displayed or only sent back to the user. Default is `TRUE`. |
| `...` | Not used (S3 compatibility). |

### Value

Send back an invisible object (igraph or Matrix, depending on the output chosen) and optionally displays a graph (via igraph or corrplot for large ones)

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
fits <- PLNnetwork(Abundance ~ 1, data = trichoptera)
myNet <- getBestModel(fits)
## Not run:
plot(myNet)

## End(Not run)
```

---

# `plot.PLNPCAfamily` *Display the criteria associated with a collection of PLNPCA fits (a PLNPCAfamily)*

## Description

Display the criteria associated with a collection of PLNPCA fits (a PLNPCAfamily)

## Usage

```
## S3 method for class 'PLNPCAfamily'
plot(x, criteria = c("loglik", "BIC", "ICL"), reverse = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | an R6 object with class `PLNPCAfamily` |
| `criteria` | vector of characters. The criteria to plot in c("loglik", "BIC", "ICL"). Default is c("loglik", "BIC", "ICL"). |
| `reverse` | A logical indicating whether to plot the value of the criteria in the "natural" direction (loglik - 0.5 penalty) or in the "reverse" direction (-2 loglik + penalty). Default to FALSE, i.e use the natural direction, on the same scale as the log-likelihood. |
| `...` | additional parameters for S3 compatibility. Not used |

## Details

The BIC and ICL criteria have the form 'loglik - 1/2 * penalty' so that they are on the same scale as the model log-likelihood. You can change this direction and use the alternate form '-2*loglik + penalty', as some authors do, by setting `reverse = TRUE`.

## Value

Produces a plot representing the evolution of the criteria of the different models considered, highlighting the best model in terms of BIC and ICL (see details).

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPCAs <- PLNPCA(Abundance ~ 1 + offset(log(Offset)), data = trichoptera, ranks = 1:5)
## Not run:
plot(myPCAs)

## End(Not run)
```

---

# `plot.PLNPCAfit` *PCA visualization (individual and/or variable factor map(s)) for a `PLNPCAfit` object*

---

## Description

PCA visualization (individual and/or variable factor map(s)) for a `PLNPCAfit` object

## Usage

```
## S3 method for class 'PLNPCAfit'
plot(
  x,
  map = c("both", "individual", "variable"),
  nb_axes = min(3, x$rank),
  axes = seq.int(min(2, x$rank)),
  ind_cols = "ind_colors",
  var_cols = "var_colors",
  plot = TRUE,
  main = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | an R6 object with class PLNPCAfit |
| `map` | the type of output for the PCA visualization: either "individual", "variable" or "both". Default is "both". |
| `nb_axes` | scalar: the number of axes to be considered when `map = "both"`. The default is `min(3,rank)`. |
| `axes` | numeric, the axes to use for the plot when `map = "individual"` or `map = "variable"`. Default it `c(1,min(rank))` |
| `ind_cols` | a character, factor or numeric to define the color associated with the individuals. By default, all variables receive the default color of the current palette. |
| `var_cols` | a character, factor or numeric to define the color associated with the variables. By default, all variables receive the default color of the current palette. |

| | |
|---|---|
| `plot` | logical. Should the plot be displayed or sent back as `ggplot2::ggplot` object |
| `main` | character. A title for the single plot (individual or variable factor map). If NULL (the default), an hopefully appropriate title will be used. |
| `...` | Not used (S3 compatibility). |

## Value

displays an individual and/or variable factor maps for the corresponding axes, and/or sends back a `ggplot2::ggplot` or gtable object

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPCAs <- PLNPCA(Abundance ~ 1 + offset(log(Offset)), data = trichoptera, ranks = 1:5)
myPCA <- getBestModel(myPCAs)
## Not run:
plot(myPCA, map = "individual", nb_axes=2, ind_cols = trichoptera$Group)
plot(myPCA, map = "variable", nb_axes=2)
plot(myPCA, map = "both", nb_axes=2, ind_cols = trichoptera$Group)

## End(Not run)
```

---

# `plot.ZIPLNfit_sparse` *Extract and plot the network (partial correlation, support or inverse covariance) from a `ZIPLNfit_sparse` object*

## Description

Extract and plot the network (partial correlation, support or inverse covariance) from a `ZIPLNfit_sparse` object

## Usage

```
## S3 method for class 'ZIPLNfit_sparse'
plot(
  x,
  type = c("partial_cor", "support"),
  output = c("igraph", "corrplot"),
  edge.color = c("#F8766D", "#00BFC4"),
  remove.isolated = FALSE,
  node.labels = NULL,
  layout = layout_in_circle,
  plot = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | an R6 object with class `ZIPLNfit_sparse` |
| `type` | character. Value of the weight of the edges in the network, either "partial_cor" (partial correlation) or "support" (binary). Default is `"partial_cor"`. |
| `output` | the type of output used: either 'igraph' or 'corrplot'. Default is `'igraph'`. |
| `edge.color` | Length 2 color vector. Color for positive/negative edges. Default is `c("#F8766D", "#00BFC4")`. Only relevant for igraph output. |
| `remove.isolated` | if `TRUE`, isolated node are remove before plotting. Only relevant for igraph output. |
| `node.labels` | vector of character. The labels of the nodes. The default will use the column names ot the response matrix. |
| `layout` | an optional igraph layout. Only relevant for igraph output. |
| `plot` | logical. Should the final network be displayed or only sent back to the user. Default is `TRUE`. |
| `...` | Not used (S3 compatibility). |

## Value

Send back an invisible object (igraph or Matrix, depending on the output chosen) and optionally displays a graph (via igraph or corrplot for large ones)

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
fit <- ZIPLN(Abundance ~ 1, data = trichoptera, control = ZIPLN_param(penalty = 0.1))
## Not run:
plot(fit)

## End(Not run)
```

---

# `predict.PLNfit` *Predict counts of a new sample*

## Description

Predict counts of a new sample

## Usage

```
## S3 method for class 'PLNfit'
predict(
  object,
  newdata,
  responses = NULL,
  level = 1,
  type = c("link", "response"),
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | an R6 object with class `PLNfit` |
| `newdata` | A data frame in which to look for variables and offsets with which to predict |
| `responses` | Optional data frame containing the count of the observed variables (matching the names of the provided as data in the PLN function), assuming the interest in in testing the model. |
| `level` | Optional integer value the level to be used in obtaining the predictions. Level zero corresponds to the population predictions (default if `responses` is not provided) while level one (default) corresponds to predictions after evaluating the variational parameters for the new data. |
| `type` | The type of prediction required. The default is on the scale of the linear predictors (i.e. log average count) |
| `...` | additional parameters for S3 compatibility. Not used |

## Value

A matrix of predicted log-counts (if `type = "link"`) or predicted counts (if `type = "response"`).

---

# `predict.PLNLDAfit` *Predict group of new samples*

## Description

Predict group of new samples

## Usage

```
## S3 method for class 'PLNLDAfit'
predict(
  object,
  newdata,
  type = c("posterior", "response", "scores"),
  scale = c("log", "prob"),
```

```
  prior = NULL,
  control = PLN_param(backend = "nlopt"),
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | an R6 object with class `PLNLDAfit` |
| `newdata` | A data frame in which to look for variables, offsets and counts with which to predict. |
| `type` | The type of prediction required. The default are posterior probabilities for each group (in either unnormalized log-scale or natural probabilities, see "scale" for details), "response" is the group with maximal posterior probability and "scores" is the average score along each separation axis in the latent space, with weights equal to the posterior probabilities. |
| `scale` | The scale used for the posterior probability. Either log-scale ("log", default) or natural probabilities summing up to 1 ("prob"). |
| `prior` | User-specified prior group probabilities in the new data. If NULL (default), prior probabilities are computed from the learning set. |
| `control` | a list for controlling the optimization. See `PLN()` for details. |
| `...` | additional parameters for S3 compatibility. Not used |

## Value

A matrix of posterior probabilities for each group (if type = "posterior"), a matrix of (average) scores in the latent space (if type = "scores") or a vector of predicted groups (if type = "response").

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myLDA <- PLNLDA(Abundance ~ 0 + offset(log(Offset)),
                grouping = Group,
                data = trichoptera)
## Not run:
post_probs <- predict(myLDA, newdata = trichoptera, type = "posterior", scale = "prob")
head(round(post_probs, digits = 3))
predicted_group <- predict(myLDA, newdata = trichoptera, type = "response")
table(predicted_group, trichoptera$Group, dnn = c("predicted", "true"))

## End(Not run)
```

---

## `predict.PLNmixturefit` *Prediction for a `PLNmixturefit` object*

---

### Description

Predict either posterior probabilities for each group or latent positions based on new samples

### Usage

```
## S3 method for class 'PLNmixturefit'
predict(
  object,
  newdata,
  type = c("posterior", "response", "position"),
 prior = matrix(rep(1/object$k, object$k), nrow(newdata), object$k, byrow = TRUE),
  control = PLNmixture_param(),
  ...
)
```

### Arguments

| | |
|---|---|
| `object` | an R6 object with class `PLNmixturefit` |
| `newdata` | A data frame in which to look for variables, offsets and counts with which to predict. |
| `type` | The type of prediction required. The default `posterior` are posterior probabilities for each group , `response` is the group with maximal posterior probability and `latent` is the averaged latent in the latent space, with weights equal to the posterior probabilities. |
| `prior` | User-specified prior group probabilities in the new data. The default uses a uniform prior. |
| `control` | a list-like structure for controlling the fit. See `PLNmixture_param()` for details. |
| `...` | additional parameters for S3 compatibility. Not used |

### Value

A matrix of posterior probabilities for each group (if type = "posterior"), a matrix of (average) position in the latent space (if type = "position") or a vector of predicted groups (if type = "response").

### Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLNmixture(Abundance ~ 1 + offset(log(Offset)),
        data = trichoptera, control = PLNmixture_param(smoothing = "none"))  %>% getBestModel()
predict(myPLN, trichoptera, "posterior")
predict(myPLN, trichoptera, "position")
predict(myPLN, trichoptera, "response")
```

---

## `predict.ZIPLNfit` *Predict counts of a new sample*

---

### Description

Predict counts of a new sample

### Usage

```
## S3 method for class 'ZIPLNfit'
predict(
  object,
  newdata,
  responses = NULL,
  level = 1,
  type = c("link", "response", "deflated"),
  ...
)
```

### Arguments

| | |
|---|---|
| `object` | an R6 object with class `ZIPLNfit` |
| `newdata` | A data frame in which to look for variables and offsets with which to predict |
| `responses` | Optional data frame containing the count of the observed variables (matching the names of the provided as data in the PLN function), assuming the interest in in testing the model. |
| `level` | Optional integer value the level to be used in obtaining the predictions. Level zero corresponds to the population predictions (default if `responses` is not provided) while level one (default) corresponds to predictions after evaluating the variational parameters for the new data. |
| `type` | Scale used for the prediction. Either `"link"` (default, predicted positions in the latent space), `"response"` (predicted average counts, accounting for zero-inflation) or `"deflated"` (predicted average counts, not accounting for zero-inflation and using only the PLN part of the model). |
| `...` | additional parameters for S3 compatibility. Not used |

### Details

Note that `level = 1` can only be used if responses are provided, as the variational parameters can't be estimated otherwise. In the absence of responses, `level` is ignored and the fitted values are returned

Note also that when `type = "response"` corresponds to predicting values with $(1-\pi)A$, where $A$ is the average count in the PLN part of the model and $\pi$ the probability of zero-inflation, whereas `type = "deflated"` corresponds to $A$.

---

## `predict_cond` *Predict counts conditionally*

---

### Description

Predict counts of a new sample conditionally on a (set of) observed variables

### Usage

```
predict_cond(
  object,
  newdata,
  cond_responses,
  type = c("link", "response"),
  var_par = FALSE
)

## S3 method for class 'PLNfit'
predict_cond(
  object,
  newdata,
  cond_responses,
  type = c("link", "response"),
  var_par = FALSE
)
```

### Arguments

| | |
|---|---|
| `object` | an R6 object with class `PLNfit` |
| `newdata` | A data frame in which to look for variables and offsets with which to predict |
| `cond_responses` | a data frame containing the counts of the observed variables (matching the names provided as data in the PLN function) |
| `type` | The type of prediction required. The default is on the scale of the linear predictors (i.e. log average count) |
| `var_par` | Boolean. Should new estimations of the variational parameters of mean and variance be sent back, as attributes of the matrix of predictions. Default to `FALSE`. |

### Value

A list containing:

| | |
|---|---|
| `pred` | A matrix of predicted log-counts (if `type = "link"`) or predicted counts (if `type = "response"`) |
| `M` | A matrix containing E(Z_uncond \| Y_c) for each given site. |
| `S` | A matrix containing Var(Z_uncond \| Y_c) for each given site (sites are the third dimension of the array) |

### Methods (by class)

- `predict_cond(PLNfit)`: Predict counts of a new sample conditionally on a (set of) observed variables for a `PLNfit`

### Examples

```
data(trichoptera)
trichoptera_prep <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLN(Abundance ~ Temperature + Wind, trichoptera_prep)
#Condition on the set of the first two species in the dataset (Hym, Hys) at the ten first sites
Yc <- trichoptera$Abundance[1:10, c(1, 2), drop=FALSE]
newX <- cbind(1, trichoptera$Covariate[1:10, c("Temperature", "Wind")])
pred <- predict_cond(myPLN, newX, Yc, type = "response")
```

---

## `prepare_data` *Prepare data for use in PLN models*

### Description

Prepare data in proper format for use in PLN model and its variants. The function (i) merges a count table and a covariate data frame in the most comprehensive way and (ii) computes offsets from the count table using one of several normalization schemes (TSS, CSS, RLE, GMPR, Wrench, etc). The function fails with informative messages when the heuristics used for sample matching fail.

### Usage

```
prepare_data(
  counts,
  covariates,
  offset = "TSS",
  call = rlang::caller_env(),
  ...
)
```

### Arguments

| | |
|---|---|
| `counts` | Required. An abundance count table, preferably with dimensions names and species as columns. |
| `covariates` | Required. A covariates data frame, preferably with row names. |
| `offset` | Optional. Normalization scheme used to compute scaling factors used as offset during PLN inference. Available schemes are "TSS" (Total Sum Scaling, default), "CSS" (Cumulative Sum Scaling, used in metagenomeSeq), "RLE" (Relative Log Expression, used in DESeq2), "GMPR" (Geometric Mean of Pairwise Ratio, introduced in Chen et al., 2018), Wrench (introduced in Kumar et al., 2018) or "none". Alternatively the user can supply its own vector or matrix of offsets (see note for specification of the user-supplied offsets). |
| `call` | Optional. The execution environment in which to set the local error call. |
| `...` | Additional parameters passed on to `compute_offset()` |

## Value

A data.frame suited for use in `PLN()` and its variants with two specials components: an abundance count matrix (in component "Abundance") and an offset vector/matrix (in component "Offset", only if offset is not set to "none")

## Note

User supplied offsets should be either vectors/column-matrices or have the same number of column as the original count matrix and either (i) dimension names or (ii) the same dimensions as the count matrix. Samples are trimmed in exactly the same way to remove empty samples.

## References

Chen, L., Reeve, J., Zhang, L., Huang, S., Wang, X. and Chen, J. (2018) GMPR: A robust normalization method for zero-inflated count data with application to microbiome sequencing data. PeerJ, 6, e4600 doi:10.7717/peerj.4600

Paulson, J. N., Colin Stine, O., Bravo, H. C. and Pop, M. (2013) Differential abundance analysis for microbial marker-gene surveys. Nature Methods, 10, 1200-1202 doi:10.1038/nmeth.2658

Anders, S. and Huber, W. (2010) Differential expression analysis for sequence count data. Genome Biology, 11, R106 doi:10.1186/gb20101110r106

Kumar, M., Slud, E., Okrah, K. et al. (2018) Analysis and correction of compositional bias in sparse sequencing count data. BMC Genomics 19, 799 doi:10.1186/s1286401851605

Robinson, M.D., Oshlack, A. (2010) A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biol 11, R25 doi:10.1186/gb2010113r25

## See Also

`compute_offset()` for details on the different normalization schemes

## Examples

```
data(trichoptera)
proper_data <- prepare_data(
 counts     = trichoptera$Abundance,
 covariates = trichoptera$Covariate,
 offset     = "GMPR",
 scale      = "count"
)
proper_data$Abundance
proper_data$Offset
```

## `rPLN` *PLN RNG*

### Description

Random generation for the PLN model with latent mean equal to mu, latent covariance matrix equal to Sigma and average depths (sum of counts in a sample) equal to depths

### Usage

```
rPLN(
  n = 10,
  mu = rep(0, ncol(Sigma)),
  Sigma = diag(1, 5, 5),
  depths = rep(10000, n)
)
```

### Arguments

| | |
|---|---|
| `n` | the sample size |
| `mu` | vectors of means of the latent variable |
| `Sigma` | covariance matrix of the latent variable |
| `depths` | Numeric vector of target depths. The first is recycled if there are not `n` values |

### Details

The default value for mu and Sigma assume equal abundances and no correlation between the different species.

### Value

a n * p count matrix, with row-sums close to depths, with an attribute "offsets" corresponding to the true generated offsets (in log-scale).

### Examples

```
## 10 samples of 5 species with equal abundances, no covariance and target depths of 10,000
rPLN()
## 2 samples of 10 highly correlated species with target depths 1,000 and 100,000
## very different abundances
mu <- rep(c(1, -1), each = 5)
Sigma <- matrix(0.8, 10, 10); diag(Sigma) <- 1
rPLN(n=2, mu = mu, Sigma = Sigma, depths = c(1e3, 1e5))
```

---

## `scRNA` *Single cell RNA-seq data*

### Description

A dataset containing the counts of the 500 most varying transcripts in the mixtures of 5 cell lines in human liver (obtained with standard 10x scRNAseq Chromium protocol).

### Usage

```
scRNA
```

### Format

A data frame named 'scRNA' with 3918 rows (the cells) and 3 variables:

**counts** a 500 trancript by 3918 count matrix

**cell_line** factor, the cell line of the current row (among 5)

**total_counts** Total number of reads for that cell ...

### Source

`https://github.com/LuyiTian/sc_mixology/`

---

## `sigma.PLNfit` *Extract variance-covariance of residuals 'Sigma'*

### Description

Extract the variance-covariance matrix of the residuals, usually noted

$$\Sigma$$

in PLN models. This captures the correlation between the species in the latent space.

### Usage

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

### Arguments

| | |
|---|---|
| `object` | an R6 object with class `PLNfit` |
| `...` | additional parameters for S3 compatibility. Not used |

## Value

A semi definite positive matrix of size p, assuming there are p species in the model.

## See Also

`coef.PLNfit()`, `standard_error.PLNfit()` and `vcov.PLNfit()` for other ways to access

$$\Sigma$$

.

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLN(Abundance ~ 1 + offset(log(Offset)), data = trichoptera)
sigma(myPLN) ## Sigma
```

---

# `sigma.PLNmixturefit` *Extract variance-covariance of residuals 'Sigma'*

## Description

Extract the variance-covariance matrix of the residuals, usually noted

$$\Sigma$$

in PLN models. This captures the correlation between the species in the latent space. or PLNmixture, it is a weighted mean of the variance-covariance matrices of each component.

## Usage

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

## Arguments

| | |
|---|---|
| `object` | an R6 object with class `PLNmixturefit` |
| `...` | additional parameters for S3 compatibility. Not used |

## Value

A semi definite positive matrix of size p, assuming there are p species in the model.

## See Also

`coef.PLNmixturefit()` for other ways to access

$$\Sigma$$

.

### Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLNmixture(Abundance ~ 1 + offset(log(Offset)),
       data = trichoptera, control = PLNmixture_param(smoothing = "none")) %>% getBestModel()
sigma(myPLN) ## Sigma
```

---

## `sigma.ZIPLNfit` *Extract variance-covariance of residuals 'Sigma'*

### Description

Extract the variance-covariance matrix of the residuals, usually noted $\Sigma$ in ZIPLN models.

### Usage

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

### Arguments

| | |
|---|---|
| `object` | an R6 object with class `ZIPLNfit` |
| `...` | additional parameters for S3 compatibility. Not used |

### Value

A semi definite positive matrix of size p, assuming there are p species in the model.

### See Also

`coef.ZIPLNfit()`

---

## `stability_selection` *Compute the stability path by stability selection*

### Description

This function computes the StARS stability criteria over a path of penalties. If a path has already been computed, the functions stops with a message unless `force = TRUE` has been specified.

### Usage

```
stability_selection(
  Robject,
  subsamples = NULL,
  control = PLNnetwork_param(),
  force = FALSE
)
```

## Arguments

| | |
|---|---|
| `Robject` | an object with class `PLNnetworkfamily` or `ZIPLNnetworkfamily`, i.e. an output from `PLNnetwork()` or `ZIPLNnetwork()` |
| `subsamples` | a list of vectors describing the subsamples. The number of vectors (or list length) determines th number of subsamples used in the stability selection. Automatically set to 20 subsamples with size `10*sqrt(n)` if `n >= 144` and `0.8*n` otherwise following Liu et al. (2010) recommendations. |
| `control` | a list controlling the main optimization process in each call to `PLNnetwork()` or `ZIPLNnetwork()`. See `PLN_param()` or `ZIPLN_param()` for details. |
| `force` | force computation of the stability path, even if a previous one has been detected. |

## Value

the list of subsamples. The estimated probabilities of selection of the edges are stored in the fields `stability_path` of the initial Robject with class `Networkfamily`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
fits <- PLNnetwork(Abundance ~ 1, data = trichoptera)
## Not run:
n <- nrow(trichoptera)
subs <- replicate(10, sample.int(n, size = n/2), simplify = FALSE)
stability_selection(nets, subsamples = subs)

## End(Not run)
```

---

`standard_error.PLNPCAfit`

# *Component-wise standard errors of B*

## Description

Extracts univariate standard errors for the estimated coefficient of B. Standard errors are computed from the (approximate) Fisher information matrix.

## Usage

```
## S3 method for class 'PLNPCAfit'
standard_error(
  object,
  type = c("variational", "jackknife", "sandwich"),
  parameter = c("B", "Omega")
)
```

```
standard_error(
  object,
  type = c("sandwich", "variational", "jackknife"),
  parameter = c("B", "Omega")
)

## S3 method for class 'PLNfit'
standard_error(
  object,
  type = c("sandwich", "variational", "jackknife", "bootstrap"),
  parameter = c("B", "Omega")
)

## S3 method for class 'PLNfit_fixedcov'
standard_error(
  object,
  type = c("sandwich", "variational", "jackknife", "bootstrap"),
  parameter = c("B", "Omega")
)

## S3 method for class 'PLNmixturefit'
standard_error(
  object,
  type = c("variational", "jackknife", "sandwich"),
  parameter = c("B", "Omega")
)

## S3 method for class 'PLNnetworkfit'
standard_error(
  object,
  type = c("variational", "jackknife", "sandwich"),
  parameter = c("B", "Omega")
)
```

## Arguments

| | |
|---|---|
| `object` | an R6 object with class PLNfit |
| `type` | string describing the type of variance approximation: "variational", "jackknife", "sandwich". Default is "sandwich". |
| `parameter` | string describing the target parameter: either B (regression coefficients) or Omega (inverse residual covariance) |

## Value

A p * d positive matrix (same size as $B$) with standard errors for the coefficients of $B$

## Methods (by class)

- `standard_error(PLNPCAfit)`: Component-wise standard errors of B in `PLNPCAfit` (not implemented yet)

- `standard_error(PLNfit)`: Component-wise standard errors of B in `PLNfit`
- `standard_error(PLNfit_fixedcov)`: Component-wise standard errors of B in `PLNfit_fixedcov`
- `standard_error(PLNmixturefit)`: Component-wise standard errors of B in `PLNmixturefit` (not implemented yet)
- `standard_error(PLNnetworkfit)`: Component-wise standard errors of B in `PLNnetworkfit` (not implemented yet)

### See Also

`vcov.PLNfit()` for the complete variance covariance estimation of the coefficient

### Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLN(Abundance ~ 1 + offset(log(Offset)), data = trichoptera,
              control = PLN_param(config_post = list(sandwich_var = TRUE)))
standard_error(myPLN)
```

---

## `trichoptera` *Trichoptera data set*

### Description

Data gathered between 1959 and 1960 during 49 insect trapping nights. For each trapping night, the abundance of 17 Trichoptera species is recorded as well as 6 meteorological variables which may influence the abundance of each species. Finally, the observations (that is to say, the trapping nights), have been classified into 12 groups corresponding to contiguous nights between summer 1959 and summer 1960.

### Usage

```
trichoptera
```

### Format

A list with 2 two data frames:

**Abundance** a 49 x 17 matrix of abundancies/counts (49 trapping nights and 17 trichoptera species)

**Covariate** a 49 x 7 data frame of covariates:

- **Temperature** Evening Temperature in Celsius
- **Wind** Wind in m/s
- **Pressure** Pressure in mm Hg
- **Humidity** relative to evening humidity in percent
- **Cloudiness** proportion of sky coverage at 9pm

**Precipitation** Nighttime precipitation in mm

**Group** a factor of 12 levels for the definition of the consecutive night groups

In order to prepare the data for using formula in multivariate analysis (multiple outputs and inputs), use `prepare_data()`. We only kept a subset of the original meteorological covariates for illustration purposes.

## Source

Data from P. Usseglio-Polatera.

## References

Usseglio-Polatera, P. and Auda, Y. (1987) Influence des facteurs météorologiques sur les résultats de piégeage lumineux. Annales de Limnologie, 23, 65–79. (code des espèces p. 76) See a data description at http://pbil.univ-lyon1.fr/R/pdf/pps034.pdf (in French)

## See Also

`prepare_data()`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
```

---

# `vcov.PLNfit` *Calculate Variance-Covariance Matrix for a fitted `PLN()` model object*

## Description

Returns the variance-covariance matrix of the main parameters of a fitted `PLN()` model object. The main parameters of the model correspond to

$$B$$

, as returned by `coef.PLNfit()`. The function can also be used to return the variance-covariance matrix of the residuals. The latter matrix can also be accessed via `sigma.PLNfit()`

## Usage

```
## S3 method for class 'PLNfit'
vcov(object, type = c("main", "covariance"), ...)
```

## Arguments

| | |
|---|---|
| `object` | an R6 object with class `PLNfit` |
| `type` | type of parameter that should be extracted. Either "main" (default) for<br>$$B$$<br>or "covariance" for<br>$$\Sigma$$ |
| `...` | additional parameters for S3 compatibility. Not used |

## Value

A matrix of variance/covariance extracted from the PLNfit model. If type="main" and $B$ is a matrix of size d * p, the result is a block-diagonal matrix with p (number of species) blocks of size d (number of covariates). if type="main", it is a symmetric matrix of size p. .

## See Also

`sigma.PLNfit()`, `coef.PLNfit()`, `standard_error.PLNfit()`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- PLN(Abundance ~ 1 + offset(log(Offset)), data = trichoptera)
vcov(myPLN, type = "covariance") ## Sigma
```

---

# `ZIPLN` *Zero Inflated Poisson lognormal model*

## Description

Fit the multivariate Zero Inflated Poisson lognormal model with a variational algorithm. Use the (g)lm syntax for model specification (covariates, offsets, subset).

## Usage

```
ZIPLN(
  formula,
  data,
  subset,
  zi = c("single", "row", "col"),
  control = ZIPLN_param()
)
```

## Arguments

| | |
|---|---|
| `formula` | an object of class "formula": a symbolic description of the model to be fitted. |
| `data` | an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which PLN is called. |
| `subset` | an optional vector specifying a subset of observations to be used in the fitting process. |
| `zi` | a character describing the model used for zero inflation, either of<br>• "single" (default, one parameter shared by all counts)<br>• "col" (one parameter per variable / feature)<br>• "row" (one parameter per sample / individual). If covariates are specified in the formula RHS (see details) this parameter is ignored. |
| `control` | a list-like structure for controlling the optimization, with default generated by `ZIPLN_param()`. See the associated documentation for details. |

## Details

Covariates for the Zero-Inflation parameter (using a logistic regression model) can be specified in the formula RHS using the pipe (`~ PLN effect | ZI effect`) to separate covariates for the PLN part of the model from those for the Zero-Inflation part. Note that different covariates can be used for each part.

## Value

an R6 object with class `ZIPLNfit`

## See Also

The class `ZIPLNfit`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
## Use different models for zero-inflation...
myZIPLN_single <- ZIPLN(Abundance ~ 1, data = trichoptera, zi = "single")
## Not run:
myZIPLN_row    <- ZIPLN(Abundance ~ 1, data = trichoptera, zi = "row")
myZIPLN_col    <- ZIPLN(Abundance ~ 1, data = trichoptera, zi = "col")
## ...including logistic regression on covariates
myZIPLN_covar  <- ZIPLN(Abundance ~ 1 | 1 + Wind, data = trichoptera)

## End(Not run)
```

---

## `ZIPLNfit` *An R6 Class to represent a ZIPLNfit*

---

### Description

The function `ZIPLN()` fits a model which is an instance of an object with class `ZIPLNfit`.

This class comes with a set of R6 methods, some of which are useful for the end-user and exported as S3 methods. See the documentation for `coef()`, `sigma()`, `predict()`.

Fields are accessed via active binding and cannot be changed by the user.

### Details

Covariates for the Zero-Inflation parameter (using a logistic regression model) can be specified in the formula RHS using the pipe (`~ PLN effect | ZI effect`) to separate covariates for the PLN part of the model from those for the Zero-Inflation part. Note that different covariates can be used for each part.

### Active bindings

`n` number of samples/sites

`q` number of dimensions of the latent space

`p` number of variables/species

`d` number of covariates in the PLN part

`d0` number of covariates in the ZI part

`nb_param_zi` number of parameters in the ZI part of the model

`nb_param_pln` number of parameters in the PLN part of the model

`nb_param` number of parameters in the ZIPLN model

`model_par` a list with the matrices of parameters found in the model (B, Sigma, plus some others depending on the variant)

`var_par` a list with two matrices, M and S2, which are the estimated parameters in the variational approximation

`optim_par` a list with parameters useful for monitoring the optimization

`latent` a matrix: values of the latent vector (Z in the model)

`latent_pos` a matrix: values of the latent position vector (Z) without covariates effects or offset

`fitted` a matrix: fitted values of the observations (A in the model)

`vcov_model` character: the model used for the covariance (either "spherical", "diagonal", "full" or "sparse")

`zi_model` character: the model used for the zero inflation (either "single", "row", "col" or "covar")

`loglik` (weighted) variational lower bound of the loglikelihood

`loglik_vec` element-wise variational lower bound of the loglikelihood

`BIC` variational lower bound of the BIC `entropy` Entropy of the variational distribution

`entropy_ZI` Entropy of the variational distribution

`entropy_PLN` Entropy of the Gaussian variational distribution in the PLN component

`ICL` variational lower bound of the ICL

`criteria` a vector with loglik, BIC, ICL and number of parameters

## Methods

### Public methods:

- `ZIPLNfit$update()`
- `ZIPLNfit$new()`
- `ZIPLNfit$optimize()`
- `ZIPLNfit$optimize_vestep()`
- `ZIPLNfit$predict()`
- `ZIPLNfit$show()`
- `ZIPLNfit$print()`
- `ZIPLNfit$clone()`

**Method** `update()`**:** Update a `ZIPLNfit` object

*Usage:*

```
ZIPLNfit$update(
  B = NA,
  B0 = NA,
  Pi = NA,
  Omega = NA,
  Sigma = NA,
  M = NA,
  S = NA,
  R = NA,
  Ji = NA,
  Z = NA,
  A = NA,
  monitoring = NA
)
```

*Arguments:*

`B` matrix of regression parameters in the Poisson lognormal component

`B0` matrix of regression parameters in the zero inflated component

`Pi` Zero inflated probability parameter (either scalar, row-vector, col-vector or matrix)

`Omega` precision matrix of the latent variables

`Sigma` covariance matrix of the latent variables

`M` matrix of mean vectors for the variational approximation

`S` matrix of standard deviation parameters for the variational approximation

`R` matrix of probabilities for the variational approximation

`Ji` vector of variational lower bounds of the log-likelihoods (one value per sample)

`Z` matrix of latent vectors (includes covariates and offset effects)

`A` matrix of fitted values

`monitoring` a list with optimization monitoring quantities

*Returns:* Update the current `ZIPLNfit` object

**Method** `new()`**:** Initialize a `ZIPLNfit` model

*Usage:*

```
ZIPLNfit$new(data, control)
```

*Arguments:*

`data` a named list used internally to carry the data matrices

`control` a list for controlling the optimization. See details.

**Method** `optimize()`**:** Call to the Cpp optimizer and update of the relevant fields

*Usage:*

```
ZIPLNfit$optimize(data, control)
```

*Arguments:*

`data` a named list used internally to carry the data matrices

`control` a list for controlling the optimization. See details.

**Method** `optimize_vestep()`**:** Result of one call to the VE step of the optimization procedure: optimal variational parameters (M, S, R) and corresponding log likelihood values for fixed model parameters (Sigma, B, B0). Intended to position new data in the latent space.

*Usage:*

```
ZIPLNfit$optimize_vestep(
  data,
  B = self$model_par$B,
  B0 = self$model_par$B0,
  Omega = self$model_par$Omega,
  control = ZIPLN_param(backend = "nlopt")$config_optim
)
```

*Arguments:*

`data` a named list used internally to carry the data matrices

`B` Optional fixed value of the regression parameters in the PLN component

`B0` Optional fixed value of the regression parameters in the ZI component

`Omega` inverse variance-covariance matrix of the latent variables

`control` a list for controlling the optimization. See details.

*Returns:* A list with three components:

- the matrix `M` of variational means,
- the matrix `S` of variational standard deviations
- the matrix `R` of variational ZI probabilities
- the vector `Ji` of (variational) log-likelihood of each new observation
- a list `monitoring` with information about convergence status

**Method** `predict()`**:** Predict position, scores or observations of new data. See `predict.ZIPLNfit()` for the S3 method and additional details

*Usage:*

```
ZIPLNfit$predict(
  newdata,
  responses = NULL,
  type = c("link", "response", "deflated"),
  level = 1,
  envir = parent.frame()
)
```

*Arguments:*

`newdata` A data frame in which to look for variables with which to predict. If omitted, the fitted values are used.

`responses` Optional data frame containing the count of the observed variables (matching the names of the provided as data in the PLN function), assuming the interest in in testing the model.

`type` Scale used for the prediction. Either `"link"` (default, predicted positions in the latent space), `"response"` (predicted average counts, accounting for zero-inflation) or `"deflated"` (predicted average counts, not accounting for zero-inflation and using only the PLN part of the model).

`level` Optional integer value the level to be used in obtaining the predictions. Level zero corresponds to the population predictions (default if `responses` is not provided) while level one (default) corresponds to predictions after evaluating the variational parameters for the new data.

`envir` Environment in which the prediction is evaluated

*Returns:* A matrix with predictions scores or counts.

**Method** `show()`**:** User friendly print method

*Usage:*

```
ZIPLNfit$show(
  model = paste("A multivariate Zero Inflated Poisson Lognormal fit with",
    self$vcov_model, "covariance model.\n")
)
```

*Arguments:*

`model` First line of the print output

**Method** `print()`**:** User friendly print method

*Usage:*

```
ZIPLNfit$print()
```

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
ZIPLNfit$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
## Not run:
# See other examples in function ZIPLN
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- ZIPLN(Abundance ~ 1, data = trichoptera)
class(myPLN)
print(myPLN)

## End(Not run)
```

---

# `ZIPLNfit_diagonal` *An R6 Class to represent a ZIPLNfit in a standard, general framework, with diagonal residual covariance*

---

## Description

An R6 Class to represent a ZIPLNfit in a standard, general framework, with diagonal residual covariance

An R6 Class to represent a ZIPLNfit in a standard, general framework, with diagonal residual covariance

## Super class

`PLNmodels::ZIPLNfit` -> `ZIPLNfit_diagonal`

## Active bindings

`nb_param_pln` number of parameters in the PLN part of the current model

`vcov_model` character: the model used for the residual covariance

## Methods

### Public methods:

- `ZIPLNfit_diagonal$new()`
- `ZIPLNfit_diagonal$clone()`

**Method** `new()`**:** Initialize a `ZIPLNfit_diagonal` model

*Usage:*

`ZIPLNfit_diagonal$new(data, control)`

*Arguments:*

`data` a named list used internally to carry the data matrices

`control` a list for controlling the optimization. See details.

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
ZIPLNfit_diagonal$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
## Not run:
# See other examples in function ZIPLN
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- ZIPLN(Abundance ~ 1, data = trichoptera, control = ZIPLN_param(covariance = "diagonal"))
class(myPLN)
print(myPLN)

## End(Not run)
```

---

# `ZIPLNfit_fixed` *An R6 Class to represent a ZIPLNfit in a standard, general framework, with fixed (inverse) residual covariance*

## Description

An R6 Class to represent a ZIPLNfit in a standard, general framework, with fixed (inverse) residual covariance

An R6 Class to represent a ZIPLNfit in a standard, general framework, with fixed (inverse) residual covariance

## Super class

`PLNmodels::ZIPLNfit` -> `ZIPLNfit_fixed`

## Active bindings

`nb_param_pln` number of parameters in the PLN part of the current model

`vcov_model` character: the model used for the residual covariance

## Methods

### Public methods:

- `ZIPLNfit_fixed$new()`
- `ZIPLNfit_fixed$clone()`

**Method** `new()`**:** Initialize a `ZIPLNfit_fixed` model

*Usage:*

```
ZIPLNfit_fixed$new(data, control)
```

*Arguments:*

`data` a named list used internally to carry the data matrices

`control` a list for controlling the optimization. See details.

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
ZIPLNfit_fixed$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
## Not run:
# See other examples in function ZIPLN
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- ZIPLN(Abundance ~ 1, data = trichoptera,
    control = ZIPLN_param(Omega = diag(ncol(trichoptera$Abundance))))
class(myPLN)
print(myPLN)

## End(Not run)
```

---

# `ZIPLNfit_sparse` *An R6 Class to represent a ZIPLNfit in a standard, general framework, with sparse inverse residual covariance*

## Description

An R6 Class to represent a ZIPLNfit in a standard, general framework, with sparse inverse residual covariance

An R6 Class to represent a ZIPLNfit in a standard, general framework, with sparse inverse residual covariance

## Super class

`PLNmodels::ZIPLNfit` -> `ZIPLNfit_sparse`

## Active bindings

`penalty` the global level of sparsity in the current model

`penalty_weights` a matrix of weights controlling the amount of penalty element-wise.

`n_edges` number of edges if the network (non null coefficient of the sparse precision matrix)

`nb_param_pln` number of parameters in the PLN part of the current model

`vcov_model` character: the model used for the residual covariance `pen_loglik` variational lower bound of the l1-penalized loglikelihood

`EBIC` variational lower bound of the EBIC

`density` proportion of non-null edges in the network

`criteria` a vector with loglik, penalized loglik, BIC, EBIC, ICL, R_squared, number of parameters, number of edges and graph density

## Methods

### Public methods:

- `ZIPLNfit_sparse$new()`
- `ZIPLNfit_sparse$latent_network()`
- `ZIPLNfit_sparse$plot_network()`
- `ZIPLNfit_sparse$clone()`

**Method** `new()`**:** Initialize a `ZIPLNfit_fixed` model

*Usage:*

```
ZIPLNfit_sparse$new(data, control)
```

*Arguments:*

`data` a named list used internally to carry the data matrices

`control` a list for controlling the optimization. See details.

**Method** `latent_network()`**:** Extract interaction network in the latent space

*Usage:*

```
ZIPLNfit_sparse$latent_network(type = c("partial_cor", "support", "precision"))
```

*Arguments:*

`type` edge value in the network. Can be "support" (binary edges), "precision" (coefficient of the precision matrix) or "partial_cor" (partial correlation between species)

*Returns:* a square matrix of size `ZIPLNfit_sparse$n`

**Method** `plot_network()`**:** plot the latent network.

*Usage:*

```
ZIPLNfit_sparse$plot_network(
  type = c("partial_cor", "support"),
  output = c("igraph", "corrplot"),
  edge.color = c("#F8766D", "#00BFC4"),
  remove.isolated = FALSE,
  node.labels = NULL,
  layout = layout_in_circle,
  plot = TRUE
)
```

*Arguments:*

`type` edge value in the network. Either "precision" (coefficient of the precision matrix) or "partial_cor" (partial correlation between species).

`output` Output type. Either `igraph` (for the network) or `corrplot` (for the adjacency matrix)

- `edge.color` Length 2 color vector. Color for positive/negative edges. Default is `c("#F8766D", "#00BFC4")`. Only relevant for igraph output.
- `remove.isolated` if `TRUE`, isolated node are remove before plotting. Only relevant for igraph output.
- `node.labels` vector of character. The labels of the nodes. The default will use the column names ot the response matrix.
- `layout` an optional igraph layout. Only relevant for igraph output.
- `plot` logical. Should the final network be displayed or only sent back to the user. Default is `TRUE`.

**Method `clone()`:** The objects of this class are cloneable with this method.

*Usage:*

```
ZIPLNfit_sparse$clone(deep = FALSE)
```

*Arguments:*

- `deep` Whether to make a deep clone.

## Examples

```
## Not run:
# See other examples in function ZIPLN
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- ZIPLN(Abundance ~ 1, data = trichoptera, control=  ZIPLN_param(penalty = 1))
class(myPLN)
print(myPLN)
plot(myPLN)

## End(Not run)
```

---

# `ZIPLNfit_spherical` *An R6 Class to represent a ZIPLNfit in a standard, general framework, with spherical residual covariance*

## Description

An R6 Class to represent a ZIPLNfit in a standard, general framework, with spherical residual covariance

An R6 Class to represent a ZIPLNfit in a standard, general framework, with spherical residual covariance

## Super class

`PLNmodels::ZIPLNfit` -> `ZIPLNfit_spherical`

## Active bindings

- `nb_param_pln` number of parameters in the PLN part of the current model
- `vcov_model` character: the model used for the residual covariance

## Methods

### Public methods:

- `ZIPLNfit_spherical$new()`
- `ZIPLNfit_spherical$clone()`

**Method** `new()`**:** Initialize a `ZIPLNfit_spherical` model

*Usage:*

```
ZIPLNfit_spherical$new(data, control)
```

*Arguments:*

`data` a named list used internally to carry the data matrices

`control` a list for controlling the optimization. See details.

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
ZIPLNfit_spherical$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
## Not run:
# See other examples in function ZIPLN
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myPLN <- ZIPLN(Abundance ~ 1, data = trichoptera, control = ZIPLN_param(covariance = "spherical"))
class(myPLN)
print(myPLN)

## End(Not run)
```

---

# `ZIPLNnetwork` *Zero Inflated Sparse Poisson lognormal model for network inference*

## Description

Perform sparse inverse covariance estimation for the Zero Inflated Poisson lognormal model using a variational algorithm. Iterate over a range of logarithmically spaced sparsity parameter values. Use the (g)lm syntax to specify the model (including covariates and offsets).

## Usage

```
ZIPLNnetwork(
  formula,
  data,
  subset,
  weights,
  zi = c("single", "row", "col"),
  penalties = NULL,
  control = ZIPLNnetwork_param()
)
```

## Arguments

| | |
|---|---|
| `formula` | an object of class "formula": a symbolic description of the model to be fitted. |
| `data` | an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which lm is called. |
| `subset` | an optional vector specifying a subset of observations to be used in the fitting process. |
| `weights` | an optional vector of observation weights to be used in the fitting process. |
| `zi` | a character describing the model used for zero inflation, either of<br>• "single" (default, one parameter shared by all counts)<br>• "col" (one parameter per variable / feature)<br>• "row" (one parameter per sample / individual). If covariates are specified in the formula RHS (see details) this parameter is ignored. |
| `penalties` | an optional vector of positive real number controlling the level of sparsity of the underlying network. if NULL (the default), will be set internally. See `PLNnetwork_param()` for additional tuning of the penalty. |
| `control` | a list-like structure for controlling the optimization, with default generated by `ZIPLNnetwork_param()`. See the associated documentation for details. |

## Details

Covariates for the Zero-Inflation parameter (using a logistic regression model) can be specified in the formula RHS using the pipe (`~ PLN effect | ZI effect`) to separate covariates for the PLN part of the model from those for the Zero-Inflation part. Note that different covariates can be used for each part.

## Value

an R6 object with class `ZIPLNnetworkfamily`

## See Also

The classes `ZIPLNfit` and `ZIPLNnetworkfamily`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
myZIPLNs <- ZIPLNnetwork(Abundance ~ 1, data = trichoptera, zi = "single")
```

---

# `ZIPLNnetworkfamily` *An R6 Class to represent a collection of ZIPLNnetwork*

---

## Description

The function `ZIPLNnetwork()` produces an instance of this class.

This class comes with a set of methods, some of them being useful for the user: See the documentation for `getBestModel()`, `getModel()` and plot()

## Super classes

`PLNmodels::PLNfamily` -> `PLNmodels::Networkfamily` -> `ZIPLNnetworkfamily`

## Public fields

`covariates0` the matrix of covariates included in the ZI component

## Methods

### Public methods:

- `ZIPLNnetworkfamily$new()`
- `ZIPLNnetworkfamily$stability_selection()`
- `ZIPLNnetworkfamily$clone()`

**Method** `new()`**:** Initialize all models in the collection

*Usage:*

```
ZIPLNnetworkfamily$new(penalties, data, control)
```

*Arguments:*

`penalties` a vector of positive real number controlling the level of sparsity of the underlying network.

`data` a named list used internally to carry the data matrices

`control` a list for controlling the optimization.

*Returns:* Update current `PLNnetworkfit` with smart starting values

**Method** `stability_selection()`**:** Compute the stability path by stability selection

*Usage:*

```
ZIPLNnetworkfamily$stability_selection(
  subsamples = NULL,
  control = ZIPLNnetwork_param()
)
```

*Arguments:*

`subsamples` a list of vectors describing the subsamples. The number of vectors (or list length) determines the number of subsamples used in the stability selection. Automatically set to 20 subsamples with size `10*sqrt(n)` if `n` >= 144 and `0.8*n` otherwise following Liu et al. (2010) recommendations.

`control` a list controlling the main optimization process in each call to `PLNnetwork()`. See `ZIPLNnetwork()` and `ZIPLN_param()` for details.

**Method `clone()`:** The objects of this class are cloneable with this method.

*Usage:*

```
ZIPLNnetworkfamily$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

The function `ZIPLNnetwork()`, the class `ZIPLNfit_sparse`

## Examples

```
data(trichoptera)
trichoptera <- prepare_data(trichoptera$Abundance, trichoptera$Covariate)
fits <- PLNnetwork(Abundance ~ 1, data = trichoptera)
class(fits)
```

---

# `ZIPLNnetwork_param` *Control of ZIPLNnetwork fit*

## Description

Helper to define list of parameters to control the ZIPLNnetwork fit. All arguments have defaults.

## Usage

```
ZIPLNnetwork_param(
  backend = c("nlopt"),
  inception_cov = c("full", "spherical", "diagonal"),
  trace = 1,
  n_penalties = 30,
  min_ratio = 0.1,
  penalize_diagonal = TRUE,
  penalty_weights = NULL,
  config_post = list(),
  config_optim = list(),
  inception = NULL
)
```

## Arguments

| | |
|---|---|
| `backend` | optimization back used, either "nlopt" or "torch". Default is "nlopt" |
| `inception_cov` | Covariance structure used for the inception model used to initialize the PLN-family. Defaults to "full" and can be constrained to "diagonal" and "spherical". |
| `trace` | a integer for verbosity. |
| `n_penalties` | an integer that specifies the number of values for the penalty grid when internally generated. Ignored when penalties is non `NULL` |
| `min_ratio` | the penalty grid ranges from the minimal value that produces a sparse to this value multiplied by `min_ratio`. Default is 0.1. |
| `penalize_diagonal` | boolean: should the diagonal terms be penalized in the graphical-Lasso? Default is `TRUE` |
| `penalty_weights` | either a single or a list of p x p matrix of weights (default: all weights equal to 1) to adapt the amount of shrinkage to each pairs of node. Must be symmetric with positive values. |
| `config_post` | a list for controlling the post-treatment (optional bootstrap, jackknife, R2, etc). |
| `config_optim` | a list for controlling the optimizer (either "nlopt" or "torch" backend). See details |
| `inception` | Set up the parameters initialization: by default, the model is initialized with a multivariate linear model applied on log-transformed data, and with the same formula as the one provided by the user. However, the user can provide a PLNfit (typically obtained from a previous fit), which sometimes speeds up the inference. |

## Details

See `PLNnetwork_param()` for a full description of the optimization parameters. Note that some defaults values are different than those used in `PLNnetwork_param()`:

- "ftol_out" (outer loop convergence tolerance the objective function) is set by default to 1e-6
- "maxit_out" (max number of iterations for the outer loop) is set by default to 100

## Value

list of parameters configuring the fit.

## See Also

`PLNnetwork_param()` and `PLN_param()`

---

`ZIPLN_param` *Control of a ZIPLN fit*

---

## Description

Helper to define list of parameters to control the PLN fit. All arguments have defaults.

## Usage

```
ZIPLN_param(
  backend = c("nlopt"),
  trace = 1,
  covariance = c("full", "diagonal", "spherical", "fixed", "sparse"),
  Omega = NULL,
  penalty = 0,
  penalize_diagonal = TRUE,
  penalty_weights = NULL,
  config_post = list(),
  config_optim = list(),
  inception = NULL
)
```

## Arguments

| | |
|---|---|
| `backend` | optimization back used, either "nlopt" or "torch". Default is "nlopt" |
| `trace` | a integer for verbosity. |
| `covariance` | character setting the model for the covariance matrix. Either "full", "diagonal", "spherical" or "fixed". Default is "full". |
| `Omega` | precision matrix of the latent variables. Inverse of Sigma. Must be specified if `covariance` is "fixed" |
| `penalty` | a user-defined penalty to sparsify the residual covariance. Defaults to 0 (no sparsity). |
| `penalize_diagonal` | boolean: should the diagonal terms be penalized in the graphical-Lasso? Default is `TRUE` |
| `penalty_weights` | either a single or a list of p x p matrix of weights (default: all weights equal to 1) to adapt the amount of shrinkage to each pairs of node. Must be symmetric with positive values. |
| `config_post` | a list for controlling the post-treatments (optional bootstrap, jackknife, R2, etc.). See details |
| `config_optim` | a list for controlling the optimizer (either "nlopt" or "torch" backend). See details |

| | |
|---|---|
| `inception` | Set up the parameters initialization: by default, the model is initialized with a multivariate linear model applied on log-transformed data, and with the same formula as the one provided by the user. However, the user can provide a PLNfit (typically obtained from a previous fit), which sometimes speeds up the inference. |

## Details

See `PLN_param()` and `PLNnetwork_param()` for a full description of the generic optimization parameters. Like `PLNnetwork_param()`, ZIPLN_param() has two parameters controlling the optimization due the inner-outer loop structure of the optimizer:

- "ftol_out" outer solver stops when an optimization step changes the objective function by less than `ftol_out` multiplied by the absolute value of the parameter. Default is 1e-6
- "maxit_out" outer solver stops when the number of iteration exceeds `maxit_out`. Default is 100 and one additional parameter controlling the form of the variational approximation of the zero inflation:
- "approx_ZI" either uses an exact or approximated conditional distribution for the zero inflation. Default is FALSE

## Value

list of parameters used during the fit and post-processing steps

# Index