Package 'fido'

March 24, 2023

Type Package

Title Bayesian Multinomial Logistic Normal Regression

Version 1.0.4

Date 2023-03-22

Maintainer Michelle Nixon
<pistner@psu.edu>

Description Provides methods for fitting and inspection of Bayesian Multinomial Logistic Normal Models using MAP estimation
and Laplace Approximation as developed in Silverman et. Al. (2022) <https://www.jmlr.org/papers/v23/19-882.html>. Key functionality is implemented in C++ for scalability. 'fido' replaces the previous package 'stray'.

License GPL-3

URL https://jsilve24.github.io/fido/

Depends R (>= 4.1.0)

Imports Rcpp (>= 0.12.17), dplyr, ggplot2, purrr, tidybayes, rlang, tidyr

LinkingTo Rcpp, RcppEigen, RcppNumerical, RcppZiggurat, BH

RoxygenNote 7.2.1

Additional_repositories https://michellepistner.github.io/fidoRepo

Suggests testthat (>= 2.1.0), knitr, rmarkdown, ape, numDeriv, MCMCpack, MicrobeDS, phyloseq

VignetteBuilder knitr

LazyData true

BugReports https://github.com/jsilve24/fido/issues

Encoding UTF-8

NeedsCompilation yes

Author Justin Silverman [aut],
Kim Roche [ctb],
Michelle Nixon [ctb, cre]

Repository CRAN

Date/Publication 2023-03-24 12:00:10 UTC

R topics documented:

alr *Compute the ALR of a matrix*

Description

Compute the ALR of a matrix

Usage

```
alr(x, d = NULL)
```

Arguments

x	A matrix where the rows are the samples
d	Index of column used as a reference. Defaults to last column

Value

matrix

alrInv *Compute the inverse ALR of a matrix*

Description

Compute the inverse ALR of a matrix

Usage

```
alrInv(y, d = NULL)
```

Arguments

y	An ALR transformed matrix
d	Index of column used as a reference. Defaults to last column

Value

matrix

alrInv_array *Compute the ALR of an array*

Description

Compute the ALR of an array

Usage

```
alrInv_array(y, d = dim(y)[coords] + 1, coords)
```

Arguments

y	multidimensional ALR transformed array
d	Index of column used as a reference. Defaults to last column
coords	index of dimension of ‘x‘ that represents coordinates

Value

array

alr_array *Compute the ALR of an array*

Description

Compute the ALR of an array

Usage

```
alr_array(x, d = dim(x)[parts], parts)
```

Arguments

x	multidimensional array in simplex
d	Index of column used as a reference. Defaults to last column
parts	index of dimension of ‘x‘ that represents parts

Value

array

as.list *Generic method to convert to list*

Description

Generic method to convert to list

Usage

```
as.list(x, ...)
```

Arguments

x	An object of class pibblefit or orthusfit
...	Other objects to pass

Value

A list object

as.list.orthusfit *Convert object of class orthusfit to a list*

Description

Convert object of class orthusfit to a list

Usage

```
## S3 method for class 'orthusfit'
as.list(x, ...)
```

Arguments

x	an object of class orthusfit
...	currently unused

Value

A list of the converted orthusfit object

as.list.pibblefit *Convert object of class pibblefit to a list*

Description

Convert object of class pibblefit to a list

Usage

```
## S3 method for class 'pibblefit'
as.list(x, ...)
```

Arguments

x	an object of class pibblefit
...	currently unused

Value

A list from the converted pibblefit object.

as.orthusfit *convert list to orthusfit*

Description

convert list to orthusfit

Usage

```
as.orthusfit(object)
```

Arguments

object	list object

Value

An orthusfit object

as.pibblefit *convert list to pibblefit*

Description

convert list to pibblefit

Usage

```
as.pibblefit(object)
```

Arguments

object	list object

Value

A pibblefit object

basset_fit *Interface to fit basset models*

Description

Basset (A Lazy Learner) - non-linear regression models in fido

Usage

```
basset(
  Y = NULL,
  X,
  upsilon = NULL,
  Theta = NULL,
  Gamma = NULL,
  Xi = NULL,
  init = NULL,
  pars = c("Eta", "Lambda", "Sigma"),
  ...
)

## S3 method for class 'bassetfit'
refit(m, pars = c("Eta", "Lambda", "Sigma"), ...)
```

Arguments

Y	D x N matrix of counts (if NULL uses priors only)
X	Q x N matrix of covariates (cannot be NULL)
upsilon	dof for inverse wishart prior (numeric must be > D) (default: D+3)
Theta	A function from dimensions dim(X) -> (D-1)xN (prior mean of gaussian process)
Gamma	A function from dimension dim(X) -> NxN (kernel matrix of gaussian process)
Xi	(D-1)x(D-1) prior covariance matrix (default: ALR transform of diag(1)*(upsilon-D)/2 - this is essentially iid on "base scale" using Aitchison terminology)
init	(D-1) x Q initialization for Eta for optimization
pars	character vector of posterior parameters to return
...	other arguments passed to pibble (which is used internally to fit the basset model)
m	object of class bassetfit

Details

the full model is given by:

$$Y_j \sim Multinomial(Pi_j)$$

$$Pi_j = Phi^{-1}(Eta_j)$$

$$Eta \sim MN_{D-1xN}(Lambda, Sigma, I_N)$$

$$Lambda \sim GP_{D-1xQ}(Theta(X), Sigma, Gamma(X))$$

$$Sigma \sim InvWish(upsilon, Xi)$$

Where Gamma(X) is short hand for the Gram matrix of the Kernel function.

Default behavior is to use MAP estimate for uncollaping the LTP model if laplace approximation is not preformed.

Value

an object of class bassetfit

check_dims *Check vector/matrix/data.frame for expected dimensions or throw error*

Description

Check vector/matrix/data.frame for expected dimensions or throw error

Usage

```
check_dims(x, d, par)
```

Arguments

x	object to check
d	expected dimensions
par	character name of x (for error message)

Value

nothing if no error, otherwise throws error

Examples

```
y <- c(1,3,4)
check_dims(y, 3, "y")
```

clr_array *Compute the CLR of an array*

Description

Compute the CLR of an array

Usage

```
clr_array(x, parts)
```

Arguments

x	multidimensional array in index
parts	index of dimension of ‘x‘ that represents parts

Value

array

coef *Return regression coefficients of pibblefit or orthusfit object*

Description

Return regression coefficients of pibblefit or orthusfit object

Usage

```
coef(object, ...)
```

Arguments

object	an object of class pibblefit or orthusfit
...	other options passed to coef.pibblefit or coef.orthusfit (see details)

Value

Array of dimension (D-1) x Q x iter

Examples

```
sim <- pibble_sim()
fit <- pibble(sim$Y, sim$X)
coef(fit)
```

coef.orthusfit *Return regression coefficients of orthus object*

Description

Orthus: Returned as array of dimension (D-1+P) x Q x iter (if in ALR or ILR) otherwise (D+P) x Q x iter.

Usage

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

Arguments

object	an object of class orthusfit
...	other options passed to coef.orthusfit (see details)

Details

Other arguments:

- use_names if column and row names were passed for Y and X in call to `pibble`, should these names be applied to output array.

Value

Array of dimension (D-1) x Q x iter

`coef.pibblefit` *Return regression coefficients of pibblefit object*

Description

Pibble: Returned as array of dimension (D-1) x Q x iter (if in ALR or ILR) otherwise DxQxiter (if in proportions or clr).

Usage

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

Arguments

`object`	an object of class pibblefit
`...`	other options passed to coef.pibblefit (see details)

Details

Other arguments:

- ‘use_names‘ if column and row names were passed for Y and X in call to `pibble`, should these names be applied to output array.

Value

Array of dimension (D-1) x Q x iter

`conjugateLinearModel` *Solve Bayesian Multivariate Conjugate Linear Model*

Description

See details for model. Notation: `N` is number of samples, `D` is the dimension of the response, `Q` is number of covariates.

Usage

```
conjugateLinearModel(Y, X, Theta, Gamma, Xi, upsilon, n_samples = 2000L)
```

Arguments

`Y`	matrix of dimension D x N
`X`	matrix of covariates of dimension Q x N
`Theta`	matrix of prior mean of dimension D x Q
`Gamma`	covariance matrix of dimension Q x Q
`Xi`	covariance matrix of dimension D x D
`upsilon`	scalar (must be > D-1) degrees of freedom for InvWishart prior
`n_samples`	number of samples to draw (default: 2000)

Details

$$Y \; MN_{D-1xN}(Lambda * X, Sigma, I_N)$$
$$Lambda \; MN_{D-1xQ}(Theta, Sigma, Gamma)$$
$$Sigma \; InvWish(upsilon, Xi)$$

This function provides a means of sampling from the posterior distribution of `Lambda` and `Sigma`.

Value

List with components

1. Lambda Array of dimension (D-1) x Q x n_samples (posterior samples)
2. Sigma Array of dimension (D-1) x (D-1) x n_samples (posterior samples)

Examples

```
sim <- pibble_sim()
eta.hat <- t(alr(t(sim$Y+0.65)))
fit <- conjugateLinearModel(eta.hat, sim$X, sim$Theta, sim$Gamma,
                            sim$Xi, sim$upsilon, n_samples=2000)
```

`convert_orthus_covariance` *Convert orthus covariance matricies between representations*

Description

Convert orthus covariance matricies between representations

Usage

```
oilrvar2ilrvar(Sigma, s, V1, V2)

oilrvar2clrvar(Sigma, s, V)

oclrvar2ilrvar(Sigma, s, V)

oalrvar2clrvar(Sigma, s, d1)

oclrvar2alrvar(Sigma, s, d2)

oalrvar2alrvar(Sigma, s, d1, d2)

oalrvar2ilrvar(Sigma, s, d1, V2)

oilrvar2alrvar(Sigma, s, V1, d2)
```

Arguments

`Sigma`	covariance matrix arrat in specified transformed space (dim(Sigma)[3]=iter)
`s`	first s rows and colums of Sigma are transformed
`V1`	ILR contrast matrix of basis Sigma is already in
`V2`	ILR contrast matrix of basis Sigma is desired in
`V`	ILR contrast matrix (i.e., transformation matrix of ILR)
`d1`	alr reference element Sigma is already expressed with respec to
`d2`	alr reference element Sigma is to be expressed with respect to

Value

matrix

`create_default_ilr_base` *Create a default ILR base*

Description

Create a default ILR base

Usage

```
create_default_ilr_base(D)
```

Arguments

`D`	the number of parts (e.g., number of columns in untransformed data)

Value

A matrix

`fido_package` *fido: Fitting and Analysis of Multinomial Logistic Normal Models*

Description

Provides methods for fitting and inspection of Bayesian Multinomial Logistic Normal Models using MAP estimation (with the ADAM optimizer) and Laplace Approximation. Key functionality is implemented in C++ for scalability.

`fido_transforms` *Transform Fit fido Parameters to other representations*

Description

These are a collection of convenience functions for transforming fido fit objects to a number of different representations including ILR bases, CLR coordinates, ALR coordinates, and proportions.

Usage

```
to_proportions(m)

to_alr(m, d)

to_ilr(m, V = NULL)

to_clr(m)

## S3 method for class 'pibblefit'
to_proportions(m)

## S3 method for class 'orthusfit'
to_proportions(m)

## S3 method for class 'pibblefit'
to_alr(m, d)

## S3 method for class 'orthusfit'
to_alr(m, d)

## S3 method for class 'pibblefit'
to_ilr(m, V = NULL)

## S3 method for class 'orthusfit'
to_ilr(m, V = NULL)

## S3 method for class 'pibblefit'
to_clr(m)

## S3 method for class 'orthusfit'
to_clr(m)
```

Arguments

m	object of class pibblefit or orthusfit (e.g., output of pibble or orthus)
d	(integer) multinomial category to take as new alr reference
V	(matrix) contrast matrix for ILR basis to transform into to (defaults to create_default_ilr_base(D))

Details

For orthus, transforms only appleid to log-ratio parameters

Note: that there is a degeneracy of representations for a covariance matrix represented in terms of proportions. As such the function to_proportions does not attempt to transform parameters Sigma or prior Xi and instead just removes them from the pibblefit object returned.

Value

object

---

gather_array *Gather Multidimensional Array to Tidy Tibble*

Description

Gather Multidimensional Array to Tidy Tibble

Usage

```
gather_array(a, value, ..., .id = NULL)
```

Arguments

a	multidimensional array
value	unquoted name of column with values (defaults to "var")
...	unquoted dimension names (defaults to "dim_1", "dim_2", etc...)
.id	if specified, name for column created with name of a captured

Value

data.frame

See Also

spread_array

Examples

```
a <- array(1:100, dim =c(10, 5, 2))
gather_array(a, sequence, A, B, C)
```

kernels *Multivariate RBF Kernel*

Description

Designed to be partially specified. (see examples)

Usage

```
SE(X, sigma = 1, rho = median(as.matrix(dist(t(X)))), jitter = 1e-10)

LINEAR(X, sigma = 1, c = rep(0, nrow(X)))
```

Arguments

X	covariate (dimension Q x N; i.e., covariates x samples)
sigma	scalar parameter
rho	scalar bandwidth parameter
jitter	small scalar to add to off-diagonal of gram matrix (for numerical underflow issues)
c	vector parameter defining intercept for linear kernel

Details

Gram matrix G is given by

SE (squared exponential):

$$G = \sigma^2 * exp(-[(X - c)'(X - c)]/(s * \rho^2))$$

LINEAR:

$$G = \sigma^2 * (X - c)'(X - c)$$

Value

Gram Matrix (N x N) (e.g., the Kernel evaluated at each pair of points)

`lambda_to_iqlr` *Transform Lambda into IQLR (Inter-Quantile Log-Ratio)*

Description

Takes idea from Wu et al. (citation below) and calculates IQLR for Lambda, potentially useful if you believe there is an invariant group of categories (e.g., taxa / genes) that are not changing (in absolute abundance) between samples. IQLR is defined as

$$IQLR_x = log(x_i / g(IQVF))$$

for i in 1,...,D. IQVF are the CLR coordinates whose variance is within the inter-quantile range (defined by `probs` argument to this function). A different IQVF is fit for each posteior sample as the IQVFs are calculted based on posterior estimates for Lambda. The variance of a CLR coordinate is defined as the norm of each row of Lambda[,focus.cov] (i.e., the covariation in Eta, explained by those covariates). This definition of variance allows uses to exclude variation from technical / trivial sources in calculation of IQVF/IQLR.

Usage

```
lambda_to_iqlr(m, focus.cov = NULL, probs = c(0.25, 0.75))
```

Arguments

`m`	object of class pibblefit (e.g., output of `pibble`)
`focus.cov`	vector of integers or characters specifying columns (covariates) of Lambda to include in calculating IQLR (if NULL, default, then uses all covariates)
`probs`	bounds for categories (i.e., features / genes / taxa) to include in calculation of iqlr (smaller bounds means more stringent inclusion criteria)

Details

Primarily intended for doing differential expression analysis under assumption that only small group of categories (e.g., taxa / genes) are changing

Value

array of dimension (D, Q, iter) where D is number of taxa, Q is number of covariates, and iter is number of posterior samples.

References

Jia R. Wu, Jean M. Macklaim, Briana L. Genge, Gregory B. Gloor (2017) Finding the center: corrections for asymmetry in high-throughput sequencing datasets. arxiv:1704.01841v1

Examples

```
sim <- pibble_sim()
fit <- pibble(sim$Y, sim$X)
# Use first two covariates to define iqlr, just show first 5 samples
lambda_to_iqlr(fit, 1:2)[,,1:5]
```

`lmvgamma` *Log of Multivarate Gamma Function - Gamma_p(a)*

Description

Log of Multivarate Gamma Function - Gamma_p(a)

Usage

```
lmvgamma(a, p)
```

Arguments

a	defined by Gamma_p(a)
p	defined by Gamma_p(a)

Value

Numeric

References

https://en.wikipedia.org/wiki/Multivariate_gamma_function

`lmvgamma_deriv` *Derivative of Log of Multivariate Gamma Function - Gamma_p(a)*

Description

Derivative of Log of Multivariate Gamma Function - Gamma_p(a)

Usage

```
lmvgamma_deriv(a, p)
```

Arguments

a	defined by Gamma_p(a)
p	defined by Gamma_p(a)

Value

Numeric

References

https://en.wikipedia.org/wiki/Multivariate_gamma_function

---

loglikMaltipooCollapsed

Calculations for the Collapsed Maltipoo Model

Description

Functions providing access to the Log Likelihood, Gradient, and Hessian of the collapsed maltipoo model. Note: These are convenience functions but are not as optimized as direct coding of the MaltipooCollapsed C++ class due to a lack of Memoization. By contrast function optimMaltipooCollapsed is much more optimized and massively cuts down on repeated calculations. A more efficient Rcpp module based implementation of these functions may following if the future. For model details see `optimMaltipooCollapsed` documentation

Usage

```
loglikMaltipooCollapsed(Y, upsilon, Theta, X, KInv, U, eta, ell, sylv = FALSE)

gradMaltipooCollapsed(Y, upsilon, Theta, X, KInv, U, eta, ell, sylv = FALSE)

hessMaltipooCollapsed(Y, upsilon, Theta, X, KInv, U, eta, ell, sylv = FALSE)
```

Arguments

Y	D x N matrix of counts
upsilon	(must be > D)
Theta	D-1 x Q matrix the prior mean for regression coefficients
X	Q x N matrix of covariates
KInv	D-1 x D-1 symmetric positive-definite matrix
U	a PQxQ matrix of stacked variance components
eta	matrix (D-1)xN of parameter values at which to calculate quantities
ell	P-vector of scale factors for each variance component (aka VCScale)
sylv	(default:false) if true and if N < D-1 will use sylvester determinant identity to speed computation

Value

see below

- loglikMaltipooCollapsed - double
- gradMaltipooCollapsed - vector
- hessMaltipooCollapsed- matrix

`loglikPibbleCollapsed` *Calculations for the Collapsed Pibble Model*

Description

Functions providing access to the Log Likelihood, Gradient, and Hessian of the collapsed pibble model. Note: These are convenience functions but are not as optimized as direct coding of the PibbleCollapsed C++ class due to a lack of Memoization. By contrast function optimPibbleCollapsed is much more optimized and massively cuts down on repeated calculations. A more efficient Rcpp module based implementation of these functions may following if the future. For model details see `optimPibbleCollapsed` documentation

Usage

```
loglikPibbleCollapsed(Y, upsilon, ThetaX, KInv, AInv, eta, sylv = FALSE)

gradPibbleCollapsed(Y, upsilon, ThetaX, KInv, AInv, eta, sylv = FALSE)

hessPibbleCollapsed(Y, upsilon, ThetaX, KInv, AInv, eta, sylv = FALSE)
```

Arguments

`Y`	D x N matrix of counts
`upsilon`	(must be > D)
`ThetaX`	D-1 x N matrix formed by Theta*X (Theta is Prior mean for regression coefficients)
`KInv`	Inverse of K for LTP (for Pibble defined as KInv = solve(Xi))
`AInv`	Inverse of A for LTP (for Pibble defined as AInv = solve(diag(N)+ X'GammaX))
`eta`	matrix (D-1)xN of parameter values at which to calculate quantities
`sylv`	(default:false) if true and if N < D-1 will use sylvester determinant identity to speed computation

Value

see below

- loglikPibbleCollapsed - double
- gradPibbleCollapsed - vector
- hessPibbleCollapsed- matrix

Examples

```
D <- 10
Q <- 2
N <- 30

# Simulate Data
Sigma <- diag(sample(1:8, D-1, replace=TRUE))
Sigma[2, 3] <- Sigma[3,2] <- -1
Gamma <- diag(sqrt(rnorm(Q)^2))
Theta <- matrix(0, D-1, Q)
Phi <-  Theta + t(chol(Sigma))%*%matrix(rnorm(Q*(D-1)), nrow=D-1)%*%chol(Gamma)
X <- matrix(rnorm(N*(Q-1)), Q-1, N)
X <- rbind(1, X)
Eta <- Phi%*%X + t(chol(Sigma))%*%matrix(rnorm(N*(D-1)), nrow=D-1)
Pi <- t(alrInv(t(Eta)))
Y <- matrix(0, D, N)
for (i in 1:N) Y[,i] <- rmultinom(1, sample(5000:10000), prob = Pi[,i])

# Priors
upsilon <- D+10
Xi <- Sigma*(upsilon-D)

# Precompute
KInv <- solve(Xi)
AInv <- solve(diag(N)+ t(X)%*%Gamma%*%X)
ThetaX <- Theta%*%X


loglikPibbleCollapsed(Y, upsilon, ThetaX, KInv, AInv, Eta)
gradPibbleCollapsed(Y, upsilon, ThetaX, KInv, AInv, Eta)[1:5]
hessPibbleCollapsed(Y, upsilon, ThetaX, KInv, AInv, Eta)[1:5,1:5]
```

---

mallard *Data from Silverman et al. (2018) Microbiome*

Description

High Resolution (hourly and daily) sampling of 4 in vitro artificial gut models with many technical replicates to identify technical variation.

Usage

```
data(mallard)
```

Format

A list containing "otu_table", "sample_data", "tax_table", and "refseq".

Details

This data is at the sequence variant level. Data at the family level processed as in Silverman et al. 2018 is given in `mallard_family`

References

Silverman et al. "Dynamic linear models guide design and analysis of microbiota studies within artificial human guts". Microbiome 2018 6:202

`mallard_family` *Data from Silverman et al. (2018) Microbiome*

Description

High Resolution (hourly and daily) sampling of 4 in vitro artificial gut models with many technical replicates to identify technical variation.

Usage

```
data(mallard_family)
```

Format

A list containing "otu_table", "sample_data", "tax_table", and "refseq".

Details

This data is at the family level and processed as in Silverman et al. 2018. Data at the sequence variant level without preprocessing is given in `mallard`

References

Silverman et al. "Dynamic linear models guide design and analysis of microbiota studies within artificial human guts". Microbiome 2018 6:202

`maltipoofit` *Create maltipoofit object*

Description

Create maltipoofit object

Usage

```
maltipoofit(
  D,
  N,
  Q,
  P,
  coord_system,
  iter = NULL,
  alr_base = NULL,
  ilr_base = NULL,
  Eta = NULL,
  Lambda = NULL,
  Sigma = NULL,
  Sigma_default = NULL,
  Y = NULL,
  X = NULL,
  upsilon = NULL,
  Theta = NULL,
  Xi = NULL,
  Xi_default = NULL,
  Gamma = NULL,
  init = NULL,
  ellinit = NULL,
  names_categories = NULL,
  names_samples = NULL,
  names_covariates = NULL,
  VCScale = NULL,
  U = NULL
)
```

Arguments

`D`	number of multinomial categories
`N`	number of samples
`Q`	number of covariates
`P`	number of variance components
`coord_system`	coordinate system objects are represented in (options include "alr", "clr", "ilr", and "proportions")

iter	number of posterior samples
alr_base	integer category used as reference (required if coord_system=="alr")
ilr_base	(D x D-1) contrast matrix (required if coord_system=="ilr")
Eta	Array of samples of Eta
Lambda	Array of samples of Lambda
Sigma	Array of samples of Sigma (null if coord_system=="proportions")
Sigma_default	Array of samples of Sigma in alr base D, used if coord_system=="proportions"
Y	DxN matrix of observed counts
X	QxN design matrix
upsilon	scalar prior dof of inverse wishart prior
Theta	prior mean of Lambda
Xi	Matrix of prior covariance for inverse wishart (null if coord_system=="proportions")
Xi_default	Matrix of prior covariance for inverse wishart in alr base D (used if coord_system=="proportions")
Gamma	QxQ covariance matrix prior for Lambda
init	matrix initial guess for Lambda used for optimization
ellinit	P vector initialization values for ell for optimization
names_categories	character vector
names_samples	character vector
names_covariates	character vector
VCScale	scale factors (delta) for variance components
U	a PQ x Q matrix of stacked variance components (each of dimension Q x Q)

Value

object of class maltipoofit

See Also

maltipoo

`maltipoo_fit` *Interface to fit maltipoo models*

Description

This function is largely a more user friendly wrapper around `optimMaltipooCollapsed` and `uncollapsePibble`. See details for model specification. Notation: `N` is number of samples, `D` is number of multinomial categories, `Q` is number of covariates, `P` is the number of variance components `iter` is the number of samples of `eta` (e.g., the parameter `n_samples` in the function `optimPibbleCollapsed`)

Usage

```
maltipoo(
  Y = NULL,
  X = NULL,
  upsilon = NULL,
  Theta = NULL,
  U = NULL,
  Xi = NULL,
  init = NULL,
  ellinit = NULL,
  pars = c("Eta", "Lambda", "Sigma"),
  ...
)
```

Arguments

`Y`	D x N matrix of counts (if NULL uses priors only)
`X`	Q x N matrix of covariates (design matrix) (if NULL uses priors only, must be present to sample Eta)
`upsilon`	dof for inverse wishart prior (numeric must be > D) (default: D+3)
`Theta`	(D-1) x Q matrix of prior mean for regression parameters (default: matrix(0, D-1, Q))
`U`	a PQ x Q matrix of stacked variance components (each of dimension Q x Q)
`Xi`	(D-1)x(D-1) prior covariance matrix (default: ALR transform of diag(1)*(upsilon-D)/2 - this is essentially iid on "base scale" using Aitchison terminology)
`init`	(D-1) x Q initialization for Eta for optimization
`ellinit`	P vector initialization values for ell for optimization
`pars`	character vector of posterior parameters to return
`...`	arguments passed to `optimPibbleCollapsed` and `uncollapsePibble`

Details

the full model is given by:

$$Y_j \sim Multinomial(Pi_j)$$

$$Pi_j = Phi^{-1}(Eta_j)$$

$$Eta \sim MN_{D-1xN}(Lambda * X, Sigma, I_N)$$

$$Lambda \sim MN_{D-1xQ}(Theta, Sigma, Gamma)$$

$$Gamma = e^{ell_1}U_1 + ... + e^{ell_P}U_P$$

$$Sigma \sim InvWish(upsilon, Xi)$$

Where A = (I_N + X * Gamma * X')^-1, K^-1 = Xi is a (D-1)x(D-1) covariance matrix, U_1 is a Q x Q covariance matrix (a variance component), e^ell_i is a scale for that variance component and Phi^-1 is ALRInv_D transform.

Default behavior is to use MAP estimate for uncollaping collapsed maltipoo model if laplace approximation is not preformed.

Parameters ell are treated as fixed and estimated by MAP estimation.

Value

an object of class maltipoofit

`metadata` *Data from Silverman et al. (2019) bioRxiv*

Description

Mock communities and calibration samples created for measuring and validating model of PCR bias. This data has been preprocessed as in the original manuscript.

Format

a data.frame metadata associated with the counts matrix 'Y'

References

Justin D. Silverman, Rachael J. Bloom, Sharon Jiang, Heather K. Durand, Sayan Mukherjee, Lawrence A. David. (2019) Measuring and Mitigating PCR Bias in Microbiome Data. bioRxiv 604025; doi: https://doi.org/10.1101/604025

miniclo *Closure operator*

Description

Closure operator

Usage

```
miniclo(x)
```

Arguments

x	vector or matrix (rows are samples, parts are columns) of data in simplex

Value

x with row entries divided by sum of row (converts vectors to row matricies)

Examples

```
x <- matrix(runif(30), 10, 3)
x <- miniclo(x)
```

miniclo_array *Closure Operation applied to array on margin*

Description

Array version of miniclo.

Usage

```
miniclo_array(x, parts)
```

Arguments

x	multidimensional array
parts	index of dimension of x that represents parts (e.g., compositional variables)

Value

array

Examples

```
x <- array(1:100, dim=c(10, 5, 2))
miniclo_array(x, parts=2)
```

mongrel-deprecated *mongrel*

Description

This function is deprecated, please use `pibble` instead.

Usage

```
mongrel(
  Y = NULL,
  X = NULL,
  upsilon = NULL,
  Theta = NULL,
  Gamma = NULL,
  Xi = NULL,
  init = NULL,
  pars = c("Eta", "Lambda", "Sigma"),
  ...
)
```

Arguments

`Y`	D x N matrix of counts (if NULL uses priors only)
`X`	Q x N matrix of covariates (design matrix) (if NULL uses priors only, must be present to sample Eta)
`upsilon`	dof for inverse wishart prior (numeric must be > D) (default: D+3)
`Theta`	(D-1) x Q matrix of prior mean for regression parameters (default: matrix(0, D-1, Q))
`Gamma`	QxQ prior covariance matrix (default: diag(Q))
`Xi`	(D-1)x(D-1) prior covariance matrix (default: ALR transform of diag(1)*(upsilon-D)/2 - this is essentially iid on "base scale" using Aitchison terminology)
`init`	(D-1) x Q initialization for Eta for optimization
`pars`	character vector of posterior parameters to return
`...`	arguments passed to `optimPibbleCollapsed` and `uncollapsePibble`

Value

An object of class pibblefit

name *Generic method for applying names to an object*

Description

Intended to be called internally by package

Usage

```
name(m, ...)
```

Arguments

m	object
...	other arguments to be passed

Value

object of same class but with names applied to dimensions

name.orthusfit *S3 for orthusfit apply names to orthusfit object*

Description

To avoid confusion, assigned default names to multinomial categories (c1 etc...) and zdimensions (z1 etc...)

Usage

```
## S3 method for class 'orthusfit'
name(m, ...)
```

Arguments

m	object of class orthusfit
...	currently ignored

Value

object of class orthusfit

name.pibblefit *S3 for pibblefit apply names to pibblefit object*

Description

S3 for pibblefit apply names to pibblefit object

Usage

```
## S3 method for class 'pibblefit'
name(m, ...)
```

Arguments

m	object of class pibblefit
...	currently ignored

Value

object of class pibblefit

names_covariates.pibblefit *Generic method for getting and setting dimension names of fit object*

Description

Generic method for getting and setting dimension names of fit object

Usage

```
## S3 method for class 'pibblefit'
names_covariates(m)

## S3 method for class 'pibblefit'
names_samples(m)

## S3 method for class 'pibblefit'
names_categories(m)

## S3 method for class 'pibblefit'
names_coords(m)

## S3 replacement method for class 'pibblefit'
names_covariates(m) <- value
```

```
## S3 replacement method for class 'pibblefit'
names_samples(m) <- value

## S3 replacement method for class 'pibblefit'
names_categories(m) <- value

names_covariates(m)

names_samples(m)

names_categories(m)

names_coords(m)

names_covariates(m) <- value

names_samples(m) <- value

names_categories(m) <- value
```

Arguments

m	object
value	character vector (or NULL)

Details

names_coords is different than names_categories. names_categories provides access to the basic names of each multinomial category. In contrast, names_coords provides access to the names of the coordinates in which an object is represented. These coordinate names are based on the category names. For example, category names may be, (OTU1, ..., OTUD) where as coordinate names could be (log(OTU1/OTUD), etc...) if object is in default coordinate system.

Value

A vector of names

---

ncategories.pibblefit *Generic method for accessing model fit dimensions*

Description

Generic method for accessing model fit dimensions

Usage

```
## S3 method for class 'pibblefit'
ncategories(m)

## S3 method for class 'pibblefit'
nsamples(m)

## S3 method for class 'pibblefit'
ncovariates(m)

## S3 method for class 'pibblefit'
niter(m)

## S3 method for class 'orthusfit'
ncategories(m)

## S3 method for class 'orthusfit'
nsamples(m)

## S3 method for class 'orthusfit'
ncovariates(m)

## S3 method for class 'orthusfit'
niter(m)

ncategories(m)

nsamples(m)

ncovariates(m)

niter(m)
```

Arguments

`m`	An object of class pibblefit

Details

An alternative approach to accessing these dimensions is to access them directly from the pibblefit object using list indexing. * `ncategories` is equivalent to `m$D` * `nsamples` is equivalent to `m$N` * `ncovariates` is equivalent to `m$Q`

Value

integer

`optimMaltipooCollapsed` *Function to Optimize the Collapsed Maltipoo Model*

Description

See details for model. Should likely be followed by function `uncollapsePibble`. Notation: `N` is number of samples, `D` is number of multinomial categories, and `Q` is number of covariates.

Usage

```
optimMaltipooCollapsed(
  Y,
  upsilon,
  Theta,
  X,
  KInv,
  U,
  init,
  ellinit,
  n_samples = 2000L,
  calcGradHess = TRUE,
  b1 = 0.9,
  b2 = 0.99,
  step_size = 0.003,
  epsilon = 1e-06,
  eps_f = 1e-10,
  eps_g = 1e-04,
  max_iter = 10000L,
  verbose = FALSE,
  verbose_rate = 10L,
  decomp_method = "cholesky",
  eigvalthresh = 0,
  jitter = 0
)
```

Arguments

`Y`	D x N matrix of counts
`upsilon`	(must be > D)
`Theta`	D-1 x Q matrix the prior mean for regression coefficients
`X`	Q x N matrix of covariates
`KInv`	D-1 x D-1 symmetric positive-definite matrix
`U`	a PQxQ matrix of stacked variance components
`init`	D-1 x N matrix of initial guess for eta used for optimization

ellinit	P vector of initial guess for ell used for optimization
n_samples	number of samples for Laplace Approximation (=0 very fast as no inversion or decomposition of Hessian is required)
calcGradHess	if n_samples=0 should Gradient and Hessian still be calculated using closed form solutions?
b1	(ADAM) 1st moment decay parameter (recommend 0.9) "aka momentum"
b2	(ADAM) 2nd moment decay parameter (recommend 0.99 or 0.999)
step_size	(ADAM) step size for descent (recommend 0.001-0.003)
epsilon	(ADAM) parameter to avoid divide by zero
eps_f	(ADAM) normalized function improvement stopping criteria
eps_g	(ADAM) normalized gradient magnitude stopping criteria
max_iter	(ADAM) maximum number of iterations before stopping
verbose	(ADAM) if true will print stats for stopping criteria and iteration number
verbose_rate	(ADAM) rate to print verbose stats to screen
decomp_method	decomposition of hessian for Laplace approximation 'eigen' (more stable-slightly, slower) or 'cholesky' (less stable, faster, default)
eigvalthresh	threshold for negative eigenvalues in decomposition of negative inverse hessian (should be <=0)
jitter	(default: 0) if >0 then adds that factor to diagonal of Hessian before decomposition (to improve matrix conditioning)

Details

Notation: Let Z_j denote the J-th row of a matrix Z. Model:

$$Y_j \sim Multinomial(Pi_j)$$
$$Pi_j = Phi^{-1}(Eta_j)$$
$$Eta \sim T_{D-1,N}(upsilon, ThetaX, K, A)$$

Where A = (I_N + e^ell_1*X*U_1*X' + ... + e^ell_P*X*U_P*X'), K is a D-1xD-1 covariance and Phi is ALRInv_D transform.

Gradient and Hessian calculations are fast as they are computed using closed form solutions. That said, the Hessian matrix can be quite large [N\(*D-1) x N*\(D-1)] and storage may be an issue.

Note: Warnings about large negative eigenvalues can either signal that the optimizer did not reach an optima or (more commonly in my experience) that the prior / degrees of freedom for the covariance (given by parameters `upsilon` and `KInv`) were too specific and at odds with the observed data. If you get this warning try the following.

1. Try restarting the optimization using a different initial guess for eta
2. Try decreasing (or even increasing)`step_size` (by increments of 0.001 or 0.002) and increasing `max_iter` parameters in optimizer. Also can try increasing `b1` to 0.99 and decreasing `eps_f` by a few orders of magnitude
3. Try relaxing prior assumptions regarding covariance matrix. (e.g., may want to consider decreasing parameter `upsilon` closer to a minimum value of D)
4. Try adding small amount of jitter (e.g., set `jitter=1e-5`) to address potential floating point errors.

Value

List containing (all with respect to found optima)

1. LogLik - Log Likelihood of collapsed model (up to proportionality constant)
2. Gradient - (if `calcGradHess`=true)
3. Hessian - (if `calcGradHess`=true) of the POSITIVE log posterior
4. Pars - Parameter value of eta
5. Samples - (D-1) x N x n_samples array containing posterior samples of eta based on Laplace approximation (if n_samples>0)
6. VCScale - value of e^ell_i at optima
7. logInvNegHessDet - the log determinant of the covariacne of the Laplace approximation, useful for calculating marginal likelihood

References

S. Ruder (2016) *An overview of gradient descent optimization algorithms*. arXiv 1609.04747

See Also

`uncollapsePibble`

`optimPibbleCollapsed` *Function to Optimize the Collapsed Pibble Model*

Description

See details for model. Should likely be followed by function `uncollapsePibble`. Notation: `N` is number of samples, `D` is number of multinomial categories, and `Q` is number of covariates.

Usage

```
optimPibbleCollapsed(
  Y,
  upsilon,
  ThetaX,
  KInv,
  AInv,
  init,
  n_samples = 2000L,
  calcGradHess = TRUE,
  b1 = 0.9,
  b2 = 0.99,
  step_size = 0.003,
  epsilon = 1e-06,
  eps_f = 1e-10,
```

```
  eps_g = 1e-04,
  max_iter = 10000L,
  verbose = FALSE,
  verbose_rate = 10L,
  decomp_method = "cholesky",
  optim_method = "adam",
  eigvalthresh = 0,
  jitter = 0,
  multDirichletBoot = -1,
  useSylv = TRUE,
  ncores = -1L,
  seed = -1L
)
```

Arguments

Y	D x N matrix of counts
upsilon	(must be > D)
ThetaX	D-1 x N matrix formed by Theta*X (Theta is Prior mean for regression coefficients)
KInv	D-1 x D-1 precision matrix (inverse of Xi)
AInv	N x N precision matrix given by (I_N + X'*Gamma*X)^-1
init	D-1 x N matrix of initial guess for eta used for optimization
n_samples	number of samples for Laplace Approximation (=0 very fast as no inversion or decomposition of Hessian is required)
calcGradHess	if n_samples=0 should Gradient and Hessian still be calculated using closed form solutions?
b1	(ADAM) 1st moment decay parameter (recommend 0.9) "aka momentum"
b2	(ADAM) 2nd moment decay parameter (recommend 0.99 or 0.999)
step_size	(ADAM) step size for descent (recommend 0.001-0.003)
epsilon	(ADAM) parameter to avoid divide by zero
eps_f	(ADAM) normalized function improvement stopping criteria
eps_g	(ADAM) normalized gradient magnitude stopping criteria
max_iter	(ADAM) maximum number of iterations before stopping
verbose	(ADAM) if true will print stats for stopping criteria and iteration number
verbose_rate	(ADAM) rate to print verbose stats to screen
decomp_method	decomposition of hessian for Laplace approximation 'eigen' (more stable-slightly, slower) or 'cholesky' (less stable, faster, default)
optim_method	(default:"adam") or "lbfgs"
eigvalthresh	threshold for negative eigenvalues in decomposition of negative inverse hessian (should be <=0)
jitter	(default: 0) if >=0 then adds that factor to diagonal of Hessian before decomposition (to improve matrix conditioning)

`multDirichletBoot`
: if >0 (overrides laplace approximation) and samples eta efficiently at MAP estimate from pseudo Multinomial-Dirichlet posterior.

`useSylv`
: (default: true) if N<D-1 uses Sylvester Determinant Identity to speed up calculation of log-likelihood and gradients.

`ncores`
: (default:-1) number of cores to use, if ncores==-1 then uses default from OpenMP typically to use all available cores.

`seed`
: (random seed for Laplace approximation – integer)

Details

Notation: Let Z_j denote the J-th row of a matrix Z. Model:

$$Y_j \sim Multinomial(Pi_j)$$
$$Pi_j = Phi^{-1}(Eta_j)$$
$$Eta \sim T_{D-1,N}(upsilon, Theta * X, K, A)$$

Where A = I_N + X * Gamma * X', K is a (D-1)x(D-1) covariance matrix, Gamma is a Q x Q covariance matrix, and Phi^-1 is ALRInv_D transform.

Gradient and Hessian calculations are fast as they are computed using closed form solutions. That said, the Hessian matrix can be quite large [N*(D-1) x N*(D-1)] and storage may be an issue.

Note: Warnings about large negative eigenvalues can either signal that the optimizer did not reach an optima or (more commonly in my experience) that the prior / degrees of freedom for the covariance (given by parameters `upsilon` and `KInv`) were too specific and at odds with the observed data. If you get this warning try the following.

1. Try restarting the optimization using a different initial guess for eta
2. Try decreasing (or even increasing)`step_size` (by increments of 0.001 or 0.002) and increasing `max_iter` parameters in optimizer. Also can try increasing `b1` to 0.99 and decreasing `eps_f` by a few orders of magnitude
3. Try relaxing prior assumptions regarding covariance matrix. (e.g., may want to consider decreasing parameter `upsilon` closer to a minimum value of D)
4. Try adding small amount of jitter (e.g., set `jitter=1e-5`) to address potential floating point errors.

Value

List containing (all with respect to found optima)

1. LogLik - Log Likelihood of collapsed model (up to proportionality constant)
2. Gradient - (if `calcGradHess=true`)
3. Hessian - (if `calcGradHess=true`) of the POSITIVE LOG POSTERIOR
4. Pars - Parameter value of eta at optima
5. Samples - (D-1) x N x n_samples array containing posterior samples of eta based on Laplace approximation (if n_samples>0)
6. Timer - Vector of Execution Times
7. logInvNegHessDet - the log determinant of the covariacne of the Laplace approximation, useful for calculating marginal likelihood

References

S. Ruder (2016) *An overview of gradient descent optimization algorithms*. arXiv 1609.04747

JD Silverman K Roche, ZC Holmes, LA David, S Mukherjee. *Bayesian Multinomial Logistic Normal Models through Marginally Latent Matrix-T Processes*. 2019, arXiv e-prints, arXiv:1903.11695

See Also

uncollapsePibble

Examples

```
sim <- pibble_sim()

# Fit model for eta
fit <- optimPibbleCollapsed(sim$Y, sim$upsilon, sim$Theta%*%sim$X, sim$KInv,
                            sim$AInv, random_pibble_init(sim$Y))
```

`orthusfit` *Create orthusfit object*

Description

Create orthusfit object

Usage

```
orthusfit(
  D,
  N,
  Q,
  P,
  coord_system,
  iter = NULL,
  alr_base = NULL,
  ilr_base = NULL,
  Eta = NULL,
  Lambda = NULL,
  Sigma = NULL,
  Sigma_default = NULL,
  Z = NULL,
  Y = NULL,
  X = NULL,
  upsilon = NULL,
  Theta = NULL,
  Xi = NULL,
  Xi_default = NULL,
  Gamma = NULL,
```

```
  init = NULL,
  names_categories = NULL,
  names_samples = NULL,
  names_Zdimensions = NULL,
  names_covariates = NULL
)
```

Arguments

D	number of multinomial categories
N	number of samples
Q	number of covariates
P	Dimension of second dataset (e.g., nrows(Z))
coord_system	coordinate system objects are represented in (options include "alr", "clr", "ilr", and "proportions")
iter	number of posterior samples
alr_base	integer category used as reference (required if coord_system=="alr")
ilr_base	(D x D-1) contrast matrix (required if coord_system=="ilr")
Eta	Array of samples of Eta
Lambda	Array of samples of Lambda
Sigma	Array of samples of Sigma (null if coord_system=="proportions")
Sigma_default	Array of samples of Sigma in alr base D, used if coord_system=="proportions"
Z	PxN matrix of real valued observations
Y	DxN matrix of observed counts
X	QxN design matrix
upsilon	scalar prior dof of inverse wishart prior
Theta	prior mean of Lambda
Xi	Matrix of prior covariance for inverse wishart (null if coord_system=="proportions")
Xi_default	Matrix of prior covariance for inverse wishart in alr base D (used if coord_system=="proportions")
Gamma	QxQ covariance matrix prior for Lambda
init	matrix initial guess for Lambda used for optimization
names_categories	character vector
names_samples	character vector
names_Zdimensions	character vector
names_covariates	character vector

Value

object of class orthusfit

See Also

pibble

---

orthus_fit *Interface to fit orthus models*

Description

This function is largely a more user friendly wrapper around optimPibbleCollapsed and uncollapsePibble for fitting orthus models. See details for model specification. Notation: N is number of samples, P is the number of dimensions of observations in the second dataset, D is number of multinomial categories, Q is number of covariates, iter is the number of samples of eta (e.g., the parameter n_samples in the function optimPibbleCollapsed)

Usage

```
orthus(
  Y = NULL,
  Z = NULL,
  X = NULL,
  upsilon = NULL,
  Theta = NULL,
  Gamma = NULL,
  Xi = NULL,
  init = NULL,
  pars = c("Eta", "Lambda", "Sigma"),
  ...
)
```

Arguments

Y	D x N matrix of counts (if NULL uses priors only)
Z	P x N matrix of counts (if NULL uses priors only - must be present/absent if Y is present/absent)
X	Q x N matrix of covariates (design matrix) (if NULL uses priors only, must be present to sample Eta)
upsilon	dof for inverse wishart prior (numeric must be > D) (default: D+3)
Theta	(D-1+P) x Q matrix of prior mean for regression parameters (default: matrix(0, D-1+P, Q))
Gamma	QxQ prior covariance matrix (default: diag(Q))
Xi	(D-1+P)x(D-1+P) prior covariance matrix (default: ALR transform of diag(1)*(upsilon-D)/2 - this is essentially iid on "base scale" using Aitchison terminology)
init	(D-1) x Q initialization for Eta for optimization
pars	character vector of posterior parameters to return
...	arguments passed to optimPibbleCollapsed and uncollapsePibble

Details

the full model is given by:

$$Y_j \sim Multinomial(Pi_j)$$

$$Pi_j = Phi^{-1}(Eta_j)$$

$$cbind(Eta, Z) \sim MN_{D-1+PxN}(Lambda * X, Sigma, I_N)$$

$$Lambda \sim MN_{D-1+PxQ}(Theta, Sigma, Gamma)$$

$$Sigma \sim InvWish(upsilon, Xi)$$

Where Gamma is a Q x Q covariance matrix, and Phi^-1 is ALRInv_D transform. That is, the orthus model models the latent multinomial log-ratios (Eta) and the observations of the second dataset jointly as a linear model. This allows Sigma to also describe the covariation between the two datasets.

Default behavior is to use MAP estimate for uncollaping the LTP model if laplace approximation is not preformed.

Value

an object of class pibblefit

References

JD Silverman K Roche, ZC Holmes, LA David, S Mukherjee. Bayesian Multinomial Logistic Normal Models through Marginally Latent Matrix-T Processes. 2019, arXiv e-prints, arXiv:1903.11695

See Also

`fido_transforms` provide convenience methods for transforming the representation of pibblefit objects (e.g., conversion to proportions, alr, clr, or ilr coordinates.)

`access_dims` provides convenience methods for accessing dimensions of pibblefit object

Examples

```
sim <- orthus_sim()
fit <- orthus(sim$Y, sim$Z, sim$X)
```

`orthus_lr_transforms` *Log-Ratio transforms for orthus objects*

Description

Log-Ratio transforms for orthus objects

Usage

```
oglr(x, s, V)

oglrInv(x, s, V)

oalr(x, s, d = NULL)

oalrInv(y, s, d = NULL)

oilr(x, s, V = NULL)

oilrInv(y, s, V = NULL)

oclr(x, s)

oclrInv(x, s)
```

Arguments

x	orthus data array (e.g., first s rows are multinomial parameters or log-ratios)
s	first s rows of x are transformed
V	transformation matrix (defines transform)
d	for ALR, which component (integer position) to take as reference (default is ncol(x)) for alrInv corresponds to column position in untransformed matrix.
y	orthus data array (e.g., first s rows are multinomial parameters or log-ratios)

Value

A matrix

orthus_sim *Simulate simple orthus dataset and priors (for testing)*

Description

Simulate simple orthus dataset and priors (for testing)

Usage

```
orthus_sim(
  D = 10,
  P = 10,
  N = 30,
  Q = 2,
  use_names = TRUE,
  true_priors = FALSE
)
```

Arguments

D	number of multinomial categories
P	number of dimensions of second dataset Z
N	number of samples
Q	number of covariates (first one is an intercept, must be > 1)
use_names	should samples, covariates, and categories be named
true_priors	should Xi and upsilon be chosen to have mean at true simulated value

Value

list

Examples

```
sim <- orthus_sim()
```

orthus_tidy_samples *Convert orthus samples of Eta Lambda and Sigma to tidy format*

Description

Combines them all into a single tibble, see example for formatting and column headers. Primarily designed to be used by summary.orthusfit.

Usage

```
orthus_tidy_samples(m, use_names = FALSE, as_factor = FALSE)
```

Arguments

m	an object of class orthusfit
use_names	should dimension indices be replaced by dimension names if provided in data used to fit pibble model.
as_factor	if use_names should names be returned as factor?

Value

tibble

Examples

```
sim <- orthus_sim()
fit <- orthus(sim$Y, sim$Z, sim$X)
fit_tidy <- orthus_tidy_samples(fit, use_names=TRUE)
head(fit_tidy)
```

---

pcrbias_mock *Data from Silverman et al. (2019) bioRxiv*

Description

Mock communities and calibration samples created for measuring and validating model of PCR bias. This data has been preprocessed as in the original manuscript.

Usage

```
data(pcrbias_mock)
```

Format

an matrix Y (counts for each community member) and a data.frame metadata

References

Justin D. Silverman, Rachael J. Bloom, Sharon Jiang, Heather K. Durand, Sayan Mukherjee, Lawrence A. David. (2019) Measuring and Mitigating PCR Bias in Microbiome Data. bioRxiv 604025; doi: https://doi.org/10.1101/604025

pibblefit *Create pibblefit object*

Description

Create pibblefit object

Usage

```
pibblefit(
  D,
  N,
  Q,
  coord_system,
  iter = NULL,
  alr_base = NULL,
  ilr_base = NULL,
  Eta = NULL,
  Lambda = NULL,
  Sigma = NULL,
  Sigma_default = NULL,
  Y = NULL,
  X = NULL,
  upsilon = NULL,
  Theta = NULL,
  Xi = NULL,
  Xi_default = NULL,
  Gamma = NULL,
  init = NULL,
  names_categories = NULL,
  names_samples = NULL,
  names_covariates = NULL
)
```

Arguments

D	number of multinomial categories
N	number of samples
Q	number of covariates
coord_system	coordinate system objects are represented in (options include "alr", "clr", "ilr", and "proportions")
iter	number of posterior samples
alr_base	integer category used as reference (required if coord_system=="alr")
ilr_base	(D x D-1) contrast matrix (required if coord_system=="ilr")
Eta	Array of samples of Eta

Lambda	Array of samples of Lambda
Sigma	Array of samples of Sigma (null if coord_system=="proportions")
Sigma_default	Array of samples of Sigma in alr base D, used if coord_system=="proportions"
Y	DxN matrix of observed counts
X	QxN design matrix
upsilon	scalar prior dof of inverse wishart prior
Theta	prior mean of Lambda
Xi	Matrix of prior covariance for inverse wishart (null if coord_system=="proportions")
Xi_default	Matrix of prior covariance for inverse wishart in alr base D (used if coord_system=="proportions")
Gamma	QxQ covariance matrix prior for Lambda
init	matrix initial guess for Lambda used for optimization
names_categories	character vector
names_samples	character vector
names_covariates	character vector

Value

object of class pibblefit

See Also

pibble

`pibble_fit` *Interface to fit pibble models*

Description

This function is largely a more user friendly wrapper around optimPibbleCollapsed and uncollapsePibble. See details for model specification. Notation: N is number of samples, D is number of multinomial categories, Q is number of covariates, iter is the number of samples of eta (e.g., the parameter n_samples in the function optimPibbleCollapsed)

Usage

```
pibble(
  Y = NULL,
  X = NULL,
  upsilon = NULL,
  Theta = NULL,
  Gamma = NULL,
  Xi = NULL,
```

```
  init = NULL,
  pars = c("Eta", "Lambda", "Sigma"),
  ...
)

## S3 method for class 'pibblefit'
refit(m, pars = c("Eta", "Lambda", "Sigma"), ...)
```

Arguments

Y	D x N matrix of counts (if NULL uses priors only)
X	Q x N matrix of covariates (design matrix) (if NULL uses priors only, must be present to sample Eta)
upsilon	dof for inverse wishart prior (numeric must be > D) (default: D+3)
Theta	(D-1) x Q matrix of prior mean for regression parameters (default: matrix(0, D-1, Q))
Gamma	QxQ prior covariance matrix (default: diag(Q))
Xi	(D-1)x(D-1) prior covariance matrix (default: ALR transform of diag(1)*(upsilon-D)/2 - this is essentially iid on "base scale" using Aitchison terminology)
init	(D-1) x Q initialization for Eta for optimization
pars	character vector of posterior parameters to return
...	arguments passed to optimPibbleCollapsed and uncollapsePibble
m	object of class pibblefit

Details

the full model is given by:

$$Y_j \sim Multinomial(Pi_j)$$
$$Pi_j = Phi^{-1}(Eta_j)$$
$$Eta \sim MN_{D-1xN}(Lambda * X, Sigma, I_N)$$
$$Lambda \sim MN_{D-1xQ}(Theta, Sigma, Gamma)$$
$$Sigma \sim InvWish(upsilon, Xi)$$

Where Gamma is a Q x Q covariance matrix, and Phi^-1 is ALRInv_D transform.

Default behavior is to use MAP estimate for uncollaping the LTP model if laplace approximation is not preformed.

Value

an object of class pibblefit

References

JD Silverman K Roche, ZC Holmes, LA David, S Mukherjee. Bayesian Multinomial Logistic Normal Models through Marginally Latent Matrix-T Processes. 2019, arXiv e-prints, arXiv:1903.11695

See Also

fido_transforms provide convenience methods for transforming the representation of pibblefit objects (e.g., conversion to proportions, alr, clr, or ilr coordinates.)

access_dims provides convenience methods for accessing dimensions of pibblefit object

Generic functions including summary, print, coef, as.list, predict, name, and sample_prior name_dims

Plotting functions provided by plot and ppc (posterior predictive checks)

Examples

```
sim <- pibble_sim()
fit <- pibble(sim$Y, sim$X)
```

---

pibble_sim *Simulate simple pibble dataset and priors (for testing)*

Description

Simulate simple pibble dataset and priors (for testing)

Usage

```
pibble_sim(D = 10, N = 30, Q = 2, use_names = TRUE, true_priors = FALSE)
```

Arguments

D	number of multinomial categories
N	number of samples
Q	number of covariates (first one is an intercept, must be > 1)
use_names	should samples, covariates, and categories be named
true_priors	should Xi and upsilon be chosen to have mean at true simulated value

Value

list

Examples

```
sim <- pibble_sim()
```

pibble_tidy_samples *Convert pibble samples of Eta Lambda and Sigma to tidy format*

Description

Combines them all into a single tibble, see example for formatting and column headers. Primarily designed to be used by summary.pibblefit.

Usage

```
pibble_tidy_samples(m, use_names = FALSE, as_factor = FALSE)
```

Arguments

m	an object of class pibblefit
use_names	should dimension indices be replaced by dimension names if provided in data used to fit pibble model.
as_factor	if use_names should names be returned as factor?

Value

tibble

Examples

```
sim <- pibble_sim()
fit <- pibble(sim$Y, sim$X)
fit_tidy <- pibble_tidy_samples(fit, use_names=TRUE)
head(fit_tidy)
```

plot.pibblefit *Plot Summaries of Posterior Distribution of pibblefit Parameters*

Description

Plot Summaries of Posterior Distribution of pibblefit Parameters

Usage

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

Arguments

x	an object of class pibblefit
...	other arguments passed to plot.pibblefit (see details)

Details

Other arguments:

- ‘par‘ parameter to plot (options: Lambda, Eta, and Sigma) (default="Lambda")
- ‘focus.cov‘ vector of covariates to include in plot (plots all if NULL)
- ‘focus.coord‘ vector of coordinates to include in plot (plots all if NULL)
- ‘focus.sample‘ vector of samples to include in plot (plots all if NULL)
- ‘use_names‘ if TRUE, uses dimension names found in data as plot labels rather than using dimension integer indices.

Value

ggplot object

Examples

```
sim <- pibble_sim(N=10, D=4, Q=3)
fit <- pibble(sim$Y, sim$X)
plot(fit, par="Lambda")
plot(fit, par="Sigma")
```

---

ppc *Generic method for visualizing posterior predictive checks*

Description

Generic method for visualizing posterior predictive checks

Usage

```
ppc(m, ...)
```

Arguments

m	object
...	other arguments passed that control visualization

Value

A plot

`ppc.pibblefit` *Visualization of Posterior Predictive Check of fit model*

Description

Visualization of Posterior Predictive Check of fit model

Usage

```
## S3 method for class 'pibblefit'
ppc(m, ...)
```

Arguments

`m`	an object of class pibblefit
`...`	other options passed to ppc (see details)

Details

ppc.pibblefit accepts the following additional arguments:

- "type" type of plot (options "lines", "points", "bounds")
- "iter" number of samples from posterior predictive distribution to plot (currently must be <= m$iter) if type=="lines" default is 50, if type=="ribbon" default is to use all available iterations.
- "from_scratch" should predictions of Y come from fitted Eta or from predictions of Eta from posterior of Lambda? (default: false)

Value

ggplot object

Examples

```
sim <- pibble_sim()
fit <- pibble(sim$Y, sim$X)
ppc(fit)
```

ppc_summary.pibblefit *Generic Method to Plot Posterior Predictive Summaries*

Description

Generic Method to Plot Posterior Predictive Summaries

Usage

```
## S3 method for class 'pibblefit'
ppc_summary(m, from_scratch = FALSE, ...)

ppc_summary(m, ...)
```

Arguments

m	model object
from_scratch	should predictions of Y come from fitted Eta or from predictions of Eta from posterior of Lambda? (default: false)
...	other arguments to pass

Value

vector

predict *Predict response from new data*

Description

Predict response from new data

Usage

```
predict(object, ...)
```

Arguments

object	An object of class pibblefit
...	Other objects to be passed to the 'predict' function

Value

(if summary==FALSE) array D x N x iter; (if summary==TRUE) tibble with calculated posterior summaries

Examples

```
sim <- pibble_sim()
fit <- pibble(sim$Y, sim$X)
predict(fit)[,,1:2] # just show 2 samples
```

`predict.bassetfit` *Predict using basset*

Description

Predict using basset

Usage

```
## S3 method for class 'bassetfit'
predict(
  object,
  newdata,
  response = "Lambda",
  size = NULL,
  use_names = TRUE,
  summary = FALSE,
  iter = NULL,
  from_scratch = FALSE,
  ...
)
```

Arguments

`object`	An object of class pibblefit
`newdata`	An optional matrix for which to evaluate prediction.
`response`	Options = "Lambda":Mean of regression, "Eta", "Y": counts
`size`	the number of counts per sample if response="Y" (as vector or matrix), default if newdata=NULL and response="Y" is to use colsums of m$Y. Otherwise uses median colsums of object$Y as default. If passed as a matrix should have dimensions ncol(newdata) x iter.
`use_names`	if TRUE apply names to output
`summary`	if TRUE, posterior summary of predictions are returned rather than samples
`iter`	number of iterations to return if NULL uses object$iter
`from_scratch`	should predictions of Y come from fitted Eta or from predictions of Eta from posterior of Lambda? (default: false)
`...`	other arguments passed to summarise_posterior

Details

currently only implemented for pibblefit objects in coord_system "default" "alr", or "ilr".

Value

(if summary==FALSE) array D x N x iter; (if summary==TRUE) tibble with calculated posterior summaries

---

`predict.pibblefit` *Predict response from new data*

Description

Predict response from new data

Usage

```
## S3 method for class 'pibblefit'
predict(
  object,
  newdata = NULL,
  response = "LambdaX",
  size = NULL,
  use_names = TRUE,
  summary = FALSE,
  iter = NULL,
  from_scratch = FALSE,
  ...
)
```

Arguments

`object`	An object of class pibblefit
`newdata`	An optional matrix for which to evaluate predictions. If NULL (default), the original data of the model is used.
`response`	Options = "LambdaX":Mean of regression, "Eta", "Y": counts
`size`	the number of counts per sample if response="Y" (as vector or matrix), default if newdata=NULL and response="Y" is to use colsums of m$Y. Otherwise uses median colsums of m$Y as default. If passed as a matrix should have dimensions ncol(newdata) x iter.
`use_names`	if TRUE apply names to output
`summary`	if TRUE, posterior summary of predictions are returned rather than samples
`iter`	number of iterations to return if NULL uses object$iter
`from_scratch`	should predictions of Y come from fitted Eta or from predictions of Eta from posterior of Lambda? (default: false)
`...`	other arguments passed to summarise_posterior

Details

currently only implemented for pibblefit objects in coord_system "default" "alr", or "ilr".

Value

(if summary==FALSE) array D x N x iter; (if summary==TRUE) tibble with calculated posterior summaries

Examples

```
sim <- pibble_sim()
fit <- pibble(sim$Y, sim$X)
predict(fit)[,,1:2] # just show 2 samples
```

---

print *Print dimensions and coordinate system information for an orthusfit or pibblefit object.*

Description

Print dimensions and coordinate system information for an orthusfit or pibblefit object.

Usage

```
print(x, ...)
```

Arguments

x	an object of class pibblefit or orthusfit
...	other arguments to pass to summary function

Value

No direct value, but a print out

Examples

```
sim <- pibble_sim()
fit <- pibble(sim$Y, sim$X)
print(fit)
```

`print.orthusfit` *Print dimensions and coordinate system information for orthusfit object.*

Description

Print dimensions and coordinate system information for orthusfit object.

Usage

```
## S3 method for class 'orthusfit'
print(x, summary = FALSE, ...)
```

Arguments

`x`	an object of class orthusfit
`summary`	if true also calculates and prints summary
`...`	other arguments to pass to summary function

Value

No direct return, prints out summary

See Also

`summary.orthusfit` summarizes posterior intervals

Examples

```
sim <- orthus_sim()
fit <- orthus(sim$Y, sim$Z, sim$X)
print(fit)
```

`print.pibblefit` *Print dimensions and coordinate system information for pibblefit object.*

Description

Print dimensions and coordinate system information for pibblefit object.

Usage

```
## S3 method for class 'pibblefit'
print(x, summary = FALSE, ...)
```

Arguments

x	an object of class pibblefit
summary	if true also calculates and prints summary
...	other arguments to pass to summary function

Value

No direct return, prints out summary

See Also

summary.pibblefit summarizes posterior intervals

Examples

```
sim <- pibble_sim()
fit <- pibble(sim$Y, sim$X)
print(fit)
```

---

random_pibble_init *Provide random initialization for pibble model*

Description

Randomly initializes based on ALR transform of counts plus random pseudocounts uniformily distributed between 0 and 1.

Usage

```
random_pibble_init(Y)
```

Arguments

Y	matrix (D x N) of counts

Details

Notation: N is number of samples and D is number of multinomial categories

Value

(D-1) x N matrix

Examples

```
Y <- matrix(sample(1:100, 100), 10, 10)
random_pibble_init(Y)
```

refit *Generic method for fitting model from passed model fit object*

Description

Generic method for fitting model from passed model fit object

Usage

```
refit(m, ...)
```

Arguments

m	object
...	other arguments passed that control fitting

Value

object of the same class as m

req *Generic method for ensuring object contains required elements*

Description

Intended to be called internally by package

Usage

```
req(m, r)
```

Arguments

m	object
r	vector of elements to test for

Value

throws error if required element is not present

req.maltipoofit *require elements to be non-null in pibblefit or throw error*

Description

require elements to be non-null in pibblefit or throw error

Usage

```
## S3 method for class 'maltipoofit'
req(m, r)
```

Arguments

m	object
r	vector of elements to test for

Value

Throws an error if null

req.orthusfit *require elements to be non-null in orthusfit or throw error*

Description

require elements to be non-null in orthusfit or throw error

Usage

```
## S3 method for class 'orthusfit'
req(m, r)
```

Arguments

m	object
r	vector of elements to test for

Value

None, throws an error if NULL

req.pibblefit *require elements to be non-null in pibblefit or throw error*

Description

require elements to be non-null in pibblefit or throw error

Usage

```
## S3 method for class 'pibblefit'
req(m, r)
```

Arguments

m	object
r	vector of elements to test for

Value

Nothing, throws an error if NULL

sample_prior *Generic method for sampling from prior distribution of object*

Description

Generic method for sampling from prior distribution of object

Usage

```
sample_prior(m, n_samples = 2000L, ...)
```

Arguments

m	object
n_samples	number of samples to produce
...	other arguments to be passed

Value

object of the same class

`sample_prior.pibblefit` *Sample from the prior distribution of pibblefit object*

Description

Note this can be used to sample from prior and then predict can be called to get counts or LambdaX (`predict.pibblefit`)

Usage

```
## S3 method for class 'pibblefit'
sample_prior(
  m,
  n_samples = 2000L,
  pars = c("Eta", "Lambda", "Sigma"),
  use_names = TRUE,
  ...
)
```

Arguments

`m`	object of class pibblefit
`n_samples`	number of samples to produce
`pars`	parameters to sample
`use_names`	should names be used if available
`...`	currently ignored

Details

Could be greatly speed up in the future if needed by sampling directly from cholesky form of inverse wishart (currently implemented as header in this library - see MatDist.h).

Value

A pibblefit object

Examples

```
# Sample prior of already fitted  pibblefit object
sim <- pibble_sim()
attach(sim)
fit <- pibble(Y, X)
head(sample_prior(fit))

# Sample prior as part of model fitting
m <- pibblefit(N=as.integer(sim$N), D=as.integer(sim$D), Q=as.integer(sim$Q),
```

```
                iter=2000L, upsilon=upsilon,
                Xi=Xi, Gamma=Gamma, Theta=Theta, X=X,
                coord_system="alr", alr_base=D)
m <- sample_prior(m)
plot(m) # plot prior distribution (defaults to parameter Lambda)
```

`store_coord` *Holds information on coordinates system to later be reapplied*

Description

`store_coord` stores coordinate information for pibblefit object and can be reapplied with function `reapply_coord`. Some coordinate systems are not useful for computation and this makes it simple keep returned object from computations in the same coordinate system as the input. (Likely most useful inside of a package)

Usage

```
store_coord(m)

reapply_coord(m, l)
```

Arguments

`m`	object of class pibblefit
`l`	object returned by function `store_coord`

Value

`store_coord` list with important information to identify c coordinate system of pibblefit object. `reapply_coord` pibblefit object in coordinate system previously stored.

`summarise_posterior` *Shortcut for summarize variable with quantiles and mean*

Description

Shortcut for summarize variable with quantiles and mean

Usage

```
summarise_posterior(data, var, ...)
```

Arguments

data	tidy data frame
var	variable name (unquoted) to be summarised
...	other expressions to pass to summarise

Details

Notation: pX refers to the X% quantile

Value

data.frame

Examples

```
d <- data.frame("a"=sample(1:10, 50, TRUE),
                "b"=rnorm(50))

# Summarize posterior for b over grouping of a and also calcuate
# minmum of b (in addition to normal statistics returned)
d <- dplyr::group_by(d, a)
summarise_posterior(d, b, mean.b = mean(b), min=min(b))
```

---

summary *Summarise pibblefit or orthusfit object and print posterior quantiles*

Description

Default calculates median, mean, 50% and 95% credible interval

Usage

```
summary(object, ...)
```

Arguments

object	an object of class pibblefit or orthusfit
...	other objects to be passed to ‘summary.pibblefit‘ or ‘summary.orthusfit‘

Value

A list if class is ‘pibblefit‘ or ‘orthusfit‘

summary.orthusfit *Summarise orthusfit object and print posterior quantiles*

Description

Default calculates median, mean, 50% and 95% credible interval

Usage

```
## S3 method for class 'orthusfit'
summary(
  object,
  pars = NULL,
  use_names = TRUE,
  as_factor = FALSE,
  gather_prob = FALSE,
  ...
)
```

Arguments

object	an object of class orthusfit
pars	character vector (default: c("Eta", "Lambda", "Sigma"))
use_names	should summary replace dimension indices with orthusfit names if names Y and X were named in call to orthus
as_factor	if use_names and as_factor then returns names as factors (useful for maintaining orderings when plotting)
gather_prob	if TRUE then prints quantiles in long format rather than wide (useful for some plotting functions)
...	other expressions to pass to summarise (using name 'val' unquoted is probably what you want)

Value

A list

summary.pibblefit *Summarise pibblefit object and print posterior quantiles*

Description

Default calculates median, mean, 50% and 95% credible interval

Usage

```
## S3 method for class 'pibblefit'
summary(
  object,
  pars = NULL,
  use_names = TRUE,
  as_factor = FALSE,
  gather_prob = FALSE,
  ...
)
```

Arguments

object	an object of class pibblefit
pars	character vector (default: c("Eta", "Lambda", "Sigma"))
use_names	should summary replace dimension indices with pibblefit names if names Y and X were named in call to pibble
as_factor	if use_names and as_factor then returns names as factors (useful for maintaining orderings when plotting)
gather_prob	if TRUE then prints quantiles in long format rather than wide (useful for some plotting functions)
...	other expressions to pass to summarise (using name 'val' unquoted is probably what you want)

Value

A list

---

uncollapsePibble *Uncollapse output from optimPibbleCollapsed to full pibble Model*

Description

See details for model. Should likely be called following optimPibbleCollapsed. Notation: N is number of samples, D is number of multinomial categories, Q is number of covariates, iter is the number of samples of eta (e.g., the parameter n_samples in the function optimPibbleCollapsed)

Usage

```
uncollapsePibble(
  eta,
  X,
  Theta,
  Gamma,
  Xi,
```

```
  upsilon,
  seed,
  ret_mean = FALSE,
  ncores = -1L
)
```

Arguments

eta	array of dimension (D-1) x N x iter (e.g., Pars output of function optimPibbleCollapsed)
X	matrix of covariates of dimension Q x N
Theta	matrix of prior mean of dimension (D-1) x Q
Gamma	covariance matrix of dimension Q x Q
Xi	covariance matrix of dimension (D-1) x (D-1)
upsilon	scalar (must be > D) degrees of freedom for InvWishart prior
seed	seed to use for random number generation
ret_mean	if true then uses posterior mean of Lambda and Sigma corresponding to each sample of eta rather than sampling from posterior of Lambda and Sigma (useful if Laplace approximation is not used (or fails) in optimPibbleCollapsed)
ncores	(default:-1) number of cores to use, if ncores==-1 then uses default from OpenMP typically to use all available cores.

Details

Notation: Let Z_j denote the J-th row of a matrix Z. While the collapsed model is given by:

$$Y_j \; Multinomial(Pi_j)$$

$$Pi_j = Phi^{-1}(Eta_j)$$

$$Eta \; T_{D-1,N}(upsilon, Theta * X, K, A)$$

Where A = I_N + X * Gamma * X', K = Xi is a (D-1)x(D-1) covariance matrix, Gamma is a Q x Q covariance matrix, and Phi^-1 is ALRInv_D transform.

The uncollapsed model (Full pibble model) is given by:

$$Y_j \; Multinomial(Pi_j)$$

$$Pi_j = Phi^{-1}(Eta_j)$$

$$Eta \; MN_{D-1xN}(Lambda * X, Sigma, I_N)$$

$$Lambda \; MN_{D-1xQ}(Theta, Sigma, Gamma)$$

$$Sigma \; InvWish(upsilon, Xi)$$

This function provides a means of sampling from the posterior distribution of Lambda and Sigma given posterior samples of Eta from the collapsed model.

Value

List with components

1. Lambda Array of dimension (D-1) x Q x iter (posterior samples)
2. Sigma Array of dimension (D-1) x (D-1) x iter (posterior samples)
3. The number of cores used
4. Timer

References

JD Silverman K Roche, ZC Holmes, LA David, S Mukherjee. Bayesian Multinomial Logistic Normal Models through Marginally Latent Matrix-T Processes. 2019, arXiv e-prints, arXiv:1903.11695

See Also

optimPibbleCollapsed

Examples

```
sim <- pibble_sim()

# Fit model for eta
fit <- optimPibbleCollapsed(sim$Y, sim$upsilon, sim$Theta%*%sim$X, sim$KInv,
                            sim$AInv, random_pibble_init(sim$Y))

# Finally obtain samples from Lambda and Sigma
fit2 <- uncollapsePibble(fit$Samples, sim$X, sim$Theta,
                                  sim$Gamma, sim$Xi, sim$upsilon,
                                  seed=2849)
```

---

verify *Generic method for verifying new objects*

Description

Intended to be called internally by package or object creator

Usage

```
verify(m, ...)
```

Arguments

m	object
...	other arguments to be passed to verify

Value

throws error if verify test fails

verify.bassetfit *Simple verification of passed bassetfit object*

Description

Simple verification of passed bassetfit object

Usage

```
## S3 method for class 'bassetfit'
verify(m, ...)
```

Arguments

m	an object of class bassetfit
...	not used

Value

throws error if any verification tests fail

verify.maltipoofit *Simple verification of passed multipoo object*

Description

Simple verification of passed multipoo object

Usage

```
## S3 method for class 'maltipoofit'
verify(m, ...)
```

Arguments

m	an object of class multipoo
...	not used

Value

throws error if any verification tests fail

verify.orthusfit *Simple verification of passed orthusfit object*

Description

Simple verification of passed orthusfit object

Usage

```
## S3 method for class 'orthusfit'
verify(m, ...)
```

Arguments

m	an object of class orthusfit
...	not used

Value

throws error if any verification tests fail

verify.pibblefit *Simple verification of passed pibblefit object*

Description

Simple verification of passed pibblefit object

Usage

```
## S3 method for class 'pibblefit'
verify(m, ...)
```

Arguments

m	an object of class pibblefit
...	not used

Value

throws error if any verification tests fail

Y *Data from Silverman et al. (2019) bioRxiv*

Description

Mock communities and calibration samples created for measuring and validating model of PCR bias. This data has been preprocessed as in the original manuscript.

Format

an matrix Y (counts for each community member)

References

Justin D. Silverman, Rachael J. Bloom, Sharon Jiang, Heather K. Durand, Sayan Mukherjee, Lawrence A. David. (2019) Measuring and Mitigating PCR Bias in Microbiome Data. bioRxiv 604025; doi: https://doi.org/10.1101/604025

Index