Package 'miWQS'

July 31, 2019

Title Multiple Imputation using Weighted Quantile Sum Regression

Version 0.1.0

Date 2019-07-31

Depends R (>= 3.5.0)

Imports stats, utils, grid, coda (>= 0.19-2), ggplot2 (>= 3.1.0), glm2 (>= 1.2.1), Hmisc (>= 4.1-1), invgamma (>= 1.1), MASS (>= 7.3-49), rlist (>= 0.4.6.1), Rsolnp (>= 1.16), survival (>= 2.43-1), tidyr (>= 0.8.2), truncnorm (>= 1.0-8)

Suggests GGally (>= 1.4.0), knitr (>= 1.23), mice (>= 3.3.0), matrixNormal (>= 0.0.0), norm (>= 1.0-9.5), pander (>= 0.6.3), rmarkdown (>= 1.13), scales (>= 1.0.0), sessioninfo (>= 1.1.1), spelling (>= 2.0), testthat (>= 2.0.1), wqs (>= 0.0.1)

Description The `miWQS` package handles the uncertainty due to below the detection limit in a correlated component mixture problem. Researchers want to determine if a set/mixture of continuous and correlated components/chemicals is associated with an outcome and if so, which components are important in that mixture. These components share a common outcome but are interval-censored between zero and low thresholds, or detection limits, that may be different across the components. The `miWQS` package applies the multiple imputation (MI) procedure to the weighted quantile sum regression (WQS) methodology for continuous, binary, or count outcomes. The two imputation models coded in `miWQS` package are: bootstrapping imputation (Lubin et.al (2004) <doi:10.1289/ehp.7199>) and Bayesian imputation.

License GPL-3

Encoding UTF-8

LazyData TRUE

Language en-US

BugReports `https://github.com/phargarten2/miWQS/issues`

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Paul M. Hargarten [aut, cre],
David C. Wheeler [aut, rev, ths]

Maintainer Paul M. Hargarten <hargartenp@vcu.edu>

Repository CRAN

Date/Publication 2019-07-31 05:00:07 UTC

R topics documented:

analyze.individually 2
coef.wqs 4
combine.AIC 5
do.many.wqs 6
estimate.wqs 8
impute.boot 11
impute.Lubin 13
impute.multivariate.bayesian 15
impute.sub 16
impute.univariate.bayesian.mi 17
make.quantile.matrix 19
plot.wqs 20
pool.mi 21
print.wqs 23
simdata87 24
wqs.pool.test 25

`analyze.individually`

Performing Individual Chemical Analysis

Description

An accessory function for `estimate.wqs()`. Performs individual chemical analyses to determine the constraint for the overall mixture effect on the outcome (β_1) in WQS regression. After adjusting for any covariates, the outcome regresses on each chemical *individually*. Returns a dataframe of statistics from these analyses.

Usage

```
analyze.individually(y, X, Z = NULL, family = c("gaussian", "binomial",
  "poisson"), offset = NULL)
```

Arguments

`y`	Outcome: numeric vector or factor. Assumed to follow an exponential family distribution.
`X`	Components/chemicals to be combined into an index; a numeric matrix or dataframe.

Z	Any covariates used. Ideally, a numeric matrix, but Z can be a factor, vector or data-frame. Assumed to be complete; observations with missing covariate values are ignored with a warning printed. If none, enter NULL.
family	The distribution of outcome y. A character value: if equal to "gaussian" a linear model is implemented; if equal to "binomial" a logistic model is implemented; if equal to "poisson", a log-link (rate or count) model is implemented. See `family` in stats package. Passed to **glm2**. Default: "gaussian".
offset	The at-risk population used as a numeric vector of length equal to the number of subjects when modeling rates in Poisson regression. Passed to **glm2**. Default: If there is no offset, enter NULL.

Details

Individual chemical analyses with the outcome can be used to determine whether the mixture of chemicals is positively or negatively related to the outcome. The constraint whether the overall mixture effect, β_1, is positive or negative is controlled by b1.pos argument in `estimate.wqs`. The b1.pos argument is TRUE if the overall chemical mixture effect is positively related to the outcome; otherwise, it is negatively related to the outcome. For each analysis, the outcome is regressed on the log of the observed values for each chemical and any other covariates Z, if they exist. This was accomplished using `glm2`. We summarized the results by recording the chemical name, estimating the log chemical effect and its standard error on the outcome, and using the Akaike Information Criterion (AIC) to indicate model fit.

By looking at the output, one can decide whether the chemical mixture is positive or negative. Generally, if the sign of estimates is mainly positive, we would decide to make b1.pos in `estimate.wqs` to be TRUE. This is just one approach to determine the direction of this constraint. Alternatively, one can conduct a WQS analysis for the positively related chemicals and another WQS analysis for the negatively related chemicals.

Value

A data-frame of statistics of individual chemical analysis is returned:

chemical.name name of the component

estimate the estimate of log chemical effect

Std.Error the standard error of log chemical effect

AIC Model Fit. See `AIC`.

See Also

Other wqs: `coef.wqs`, `do.many.wqs`, `estimate.wqs`, `make.quantile.matrix`, `plot.wqs`, `print.wqs`

Examples

```
# Binomial Example
data("simdata87")
analyze.individually(
  y = simdata87$y.scenario, X = simdata87$X.true, Z = simdata87$Z.sim,
```

```
  family = "binomial"
)
# Here, most of the "Estimate", which is the log_odds of each log of the
# chemical has on the outcome, are positive, possibly indicating a positive
# relationship between mixture of chemicals and the outcome.
```

`coef.wqs` *Finding WQS Coefficients*

Description

An accessor function that returns the coefficients from the validation WQS model, a **wqs** object.

Usage

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

Arguments

`object`	An object of class "wqs", usually as a result of `estimate.wqs`.
`...`	other arguments.

Details

In a **wqs** object, the *fit* element, a **glm2** object, is extracted. See `glm2{glm2}`.

See Also

`coef`

Other wqs: `analyze.individually`, `do.many.wqs`, `estimate.wqs`, `make.quantile.matrix`, `plot.wqs`, `print.wqs`

Examples

```
# Use simulated dataset and set seed for reproducibility.
data(simdata87)
set.seed(23456)
Wa <- estimate.wqs(
  y = simdata87$y.scenario, X = simdata87$X.true[ , 1:3],
  B = 10, family = "binomial"
)
coef(Wa)
```

`combine.AIC` *Combining AICs*

Description

Combines individual AIC estimates of separate models to get a sense of overall model fit.

Usage

```
combine.AIC(AIC)
```

Arguments

`AIC`	A vector of AICs to combine with length equal to the number of models completed (i.e. K).

Details

Used in stage 3 when combining WQS model fits used on different completely observed datasets. Similar to combining WQS parameter estimates, the mean of individual AIC estimates is taken as the central tendency estimate of WQS model fit. The standard deviation between individual AIC estimates indicates the difference in WQS model fit due to BDL values.

A vector of AICs may be generated from `do.many.wqs()`.

Value

The overall fit of a model across all imputation models: the mean AIC +/- the standard error. Saved as a 1x1 character vector.

Warning If AIC is a vector with one element, the AIC is returned as a character rounded to the nearest whole number with a warning printed that AIC cannot be combined.

NA

See Also

pool.mi

Examples

```
# AICs from do.many.wqs() example are as follows.
bayes.AIC <- c(1295.380, 1295.669)
combine.AIC(bayes.AIC)

# One AIC
combine.AIC(1295.380)
```

`do.many.wqs` *Performing Many WQS Regressions*

Description

Second Stage of Multiple Imputation: In order to analyze a complete imputed chemical array (X.imputed, n subjects by C chemicals by K imputations) via weighted quantile sum regression, `do.many.wqs()` repeatedly performs the same WQS analysis on each imputed dataset. It repeatedly executes the `estimate.wqs()` function.

Usage

```
do.many.wqs(y, X.imputed, Z = NULL, B = 100, ...)
```

Arguments

`y`	Outcome: numeric vector or factor. Assumed to follow an exponential family distribution.
`X.imputed`	Array of complete components with n subjects and C components and K imputations. Must be complete.
`Z`	Any covariates used. Ideally, a numeric matrix, but Z can be a factor, vector or data-frame. Assumed to be complete; observations with missing covariate values are ignored with a warning printed. If none, enter NULL.
`B`	Number of bootstrap samples to be used in estimating the weights in the training dataset. In order to use WQS without bootstrapping, set B = 1. However, Carrico et al 2014 suggests that bootstrap some large number (like 100 or 1000) can increase component selection. In that spirit, we set the default to 100.
`...`	Additional arguments passed to `estimate.wqs`, but the arguments y, X, Z, and place.bdls.Q1 have no effect.

Value

Returns a list with elements that consist of matrix and list versions of estimate.wqs() output:

- call: the function call, processed by **rlist**.
- C: the number of chemicals in mixture, number of columns in X.
- n: the sample size.
- wqs.imputed.estimates: Array with rows = # of parameters, 2 columns = mean and standard deviation, and 3rd dimension = K.
- AIC: The overall fit of WQS models, taken as the mean AIC and standard error across all imputation models. Saved as a character element. Can see AICs in all models by seeing wqs.fit.
- train.index: Numerical indices selected to form training dataset from the last WQS model.
- q.train: Vector of quantiles used in training data from the last WQS model

- train.comparison: A list of data-frames that compares the training and validation dataset for all WQS models.
- initial: Matrix with K columns that contains the initial values used for each WQS analysis.
- wqs.train.estimates: Data-frame with rows = B. Summarizes statistics from nonlinear regression in the training datasets of all analyses:

  beta1 estimate using solnp

  beta1_glm, SE_beta1, test_stat, pvalue estimates of WQS parameter in model using glm2.

  convergence whether or not the samples have converged

  weight estimates estimates of weight for each bootstrap.

  imputed A number indicating the completed dataset used in WQS analysis.
- wqs.fit:A list (length = K) of glm2 objects of the WQS model fit to validation data. These are all the WQS estimates for all analyses. See `glm2`.

Note

Note #1: We only impute the missing values of the components, X. Any missing data in the outcome and covariates are removed and ignored.

Note #2: No seed is set in this function. Because bootstraps and splitting is random, a seed should be set before every use.

Note #3: If there is one imputed dataset, `do.many.wqs` may be used, but it is overly complicated. Use the `estimate.wqs` function instead.

See Also

Other wqs: `analyze.individually`, `coef.wqs`, `estimate.wqs`, `make.quantile.matrix`, `plot.wqs`, `print.wqs`

Examples

```
data("simdata87")
# Create 2 multiple imputed datasets using bootstrapping, but only use first 2 chemicals.
l <- impute.boot(
  X = simdata87$X.bdl[, 1:2], DL = simdata87$DL[1:2],
  Z = simdata87$Z.sim[, 1], K = 2
)
# Perform WQS regression on each imputed dataset
set.seed(50679)
bayes.wqs <- do.many.wqs(
  y = simdata87$y.scenario, X.imputed = l$X.imputed,
  Z = simdata87$Z.sim,
  B = 10, family = "binomial"
)
bayes.wqs$wqs.imputed.estimates

# @importFrom scales ordinal
```

`estimate.wqs` *Weighted Quantile Sum (WQS) Regression*

Description

Performs weighted quantile sum (WQS) regression model for continuous, binary, and count outcomes that was extended from `wqs.est` (author: Czarnota) in the **wqs** package. By default, if there is any missing data, the missing data is assumed to be censored and placed in the first quantile. Accessory functions (print, coefficient, plot) also accompany each WQS object.

Usage

```
estimate.wqs(y, X, Z = NULL, proportion.train = 1L, n.quantiles = 4L,
  place.bdls.in.Q1 = if (anyNA(X)) TRUE else FALSE, B = 100L,
  b1.pos = TRUE, signal.fn = c("signal.none", "signal.converge.only",
  "signal.abs", "signal.test.stat"), family = c("gaussian", "binomial",
  "poisson"), offset = NULL, verbose = FALSE)
```

Arguments

`y`	Outcome: numeric vector or factor. Assumed to follow an exponential family distribution.
`X`	Components/chemicals to be combined into an index; a numeric matrix or data-frame.
`Z`	Any covariates used. Ideally, a numeric matrix, but Z can be a factor, vector or data-frame. Assumed to be complete; observations with missing covariate values are ignored with a warning printed. If none, enter NULL.
`proportion.train`	The proportion of data between 0 and 1 used to train the model. If proportion.train = 1L, all the data is used to both train and validate the model. Default: 1L.
`n.quantiles`	An integer to specify the number of quantiles to be used categorizing the columns of X, e.g. in quartiles (q = 4), deciles (q = 10), or percentiles (q = 100). Default: 4L.
`place.bdls.in.Q1`	Logical; if TRUE or X has any missing values, missing values in X are placed in the first quantile of the weighted sum. Otherwise, the data is complete (no missing data) and the data is split equally into quantiles.
`B`	Number of bootstrap samples to be used in estimating the weights in the training dataset. In order to use WQS without bootstrapping, set B = 1. However, Carrico et al 2014 suggests that bootstrap some large number (like 100 or 1000) can increase component selection. In that spirit, we set the default to 100.
`b1.pos`	Logical; TRUE if the mixture index is expected to be positively related to the outcome (the default). If mixture index is expected to be inversely related to the outcome, put FALSE.

`signal.fn`	A character value indicating which signal function is used in calculating the mean weight. See details.
`family`	The distribution of outcome y. A character value: if equal to "gaussian" a linear model is implemented; if equal to "binomial" a logistic model is implemented; if equal to "poisson", a log-link (rate or count) model is implemented. See `family` in stats package. Passed to **glm2**. Default: "gaussian".
`offset`	The at-risk population used as a numeric vector of length equal to the number of subjects when modeling rates in Poisson regression. Passed to **glm2**. Default: If there is no offset, enter NULL.
`verbose`	Logical; if TRUE, prints more information. Useful to check for errors in the code. Default: FALSE.

Details

The *solnp* algorithm, or a nonlinear optimization technique using augmented Lagrange method, is used to estimate the weights in the training set. If the log likelihood evaluated at the current parameters is too large (NaN), the log likelihood is reset to be 1e24. We have discovered no issue with this reset. A data-frame with object name *train.estimates* that summarizes statistics from the nonlinear regression is returned; it consists of these columns:

beta1 estimate using solnp

beta1_glm, SE_beta1, test_stat, pvalue estimates of WQS parameter in model using glm2.

convergence logical, if TRUE the solnp solver has converged. See *solnp*.

weight estimates estimates of weight for each bootstrap.

This package uses the *glm2* function in the **glm2** package to fit the validation model.

Signal functions allow the user to adjust what bootstraps are used in calculating the mean weight. Looking at a histogram of the overall mixture effect, which is an element after plotting a WQS object, may help you to choose a signal function. The *signal.fn* argument allows the user to choose between four signal functions:

signal.none Uses all bootstrap-estimated weights in calculating average weight.

signal.converge.only Uses the estimated weights for the bootstrap samples that converged.

signal.abs Applies more weight to the absolute value of test statistic for beta1, the overall mixture effect.

signal.test stat Applies more weight to the absolute value of test statistic for beta1, the overall mixture effect.

Rates can be modelled using the offset. The *offset* argument of `estimate.wqs()` function is on the normal scale, so please do not take a logarithm. The objective function used to model the mean rate of the *ith* individual λ_i with the offset is:

$$\lambda_i = offset * exp(\eta)$$

, where η is the linear term of a regression.

The object is a member of the" *"wqs"* class; accessory functions include `coef()`, `print()`, and `plot()`.

Value

`estimate.wqs` returns an object of class "wqs". A list with the following items: (** important)

call The function call, processed by **rlist**.

C The number of chemicals in mixture, number of columns in X.

n The sample size.

train.index Vector, The numerical indices selected to form the training dataset. Useful to do side-by-side comparisons.

q.train Matrix of quantiles used in training data.

q.valid Matrix of quantiles used in validation data.

train.comparison Data-frame that compares the training and validation datasets to validate equivalence

initial Vector: Initial values used in WQS

train.estimates Data-frame with rows = B. Summarizes statistics from nonlinear regression in training dataset. See details.

processed.weights ** A C x 2 matrix, mean bootstrapped weights (and their standard errors) after filtering using signal function. Used in calculating the WQS index.

WQS Vector of the weighted quantile sum estimate based on the processed weights.

fit ** glm2 object of the WQS model fit to validation data. See `glm2{glm2}`.

boot.index Matrix of bootstrap indices used in training dataset to estimate weights. Its dimension is the length of training dataset with number of columns = B.

Note

No seed is set in this function. Because bootstraps and splitting is random, a seed should be set before every use.

References

Carrico, C., Gennings, C., Wheeler, D. C., & Factor-Litvak, P. (2014). Characterization of Weighted Quantile Sum Regression for Highly Correlated Data in a Risk Analysis Setting. Journal of Agricultural, Biological, and Environmental Statistics, 20(1), 100–120. https://doi.org/10.1007/s13253-014-0180-3

Czarnota, J., Gennings, C., Colt, J. S., De Roos, A. J., Cerhan, J. R., Severson, R. K., ... Wheeler, D. C. (2015). Analysis of Environmental Chemical Mixtures and Non-Hodgkin Lymphoma Risk in the NCI-SEER NHL Study. Environmental Health Perspectives, 123(10), 965–970. https://doi.org/10.1289/ehp.1408630

Czarnota, J., Gennings, C., & Wheeler, D. C. (2015). Assessment of Weighted Quantile Sum Regression for Modeling Chemical Mixtures and Cancer Risk. Cancer Informatics, 14, 159–171. https://doi.org/10.4137/CIN.S17295

See Also

Other wqs: `analyze.individually`, `coef.wqs`, `do.many.wqs`, `make.quantile.matrix`, `plot.wqs`, `print.wqs`

Examples

```
# Example 1: Binary outcome using the example simulated dataset in this package.
 data(simdata87)
 set.seed(23456)
 W.bin4  <- estimate.wqs(
                 y = simdata87$y.scenario, X = simdata87$X.true[, 1:9],
                 B = 10, family = "binomial"
                 )
 W.bin4

# Example 2: Continuous outcome. Use WQSdata example from wqs package.
 if (requireNamespace("wqs", quietly = TRUE)) {
  library(wqs)
  data(WQSdata)
  set.seed(23456)
#    W <- wqs::wqs.est(y = WQSdata$y, X = WQSdata[,1:9], B = 10)
  Wa <- estimate.wqs (y = WQSdata$y, X = WQSdata[, 1:9], B = 10)
  Wa
 } else {
  message("you need to install the package wqs for this example")
 }
## More examples are found 02_WQS_Examples.
## Also checked vs. Jenna's code, as well as thesis data, to verify results.

# TEMP #' @importFrom makeJournalTables  make.descriptive.table  my.summary
```

`impute.boot` *Bootstrapping Imputation for Many Chemicals*

Description

If many chemicals have values below the detection limit, this function creates an imputed dataset using a bootstrap procedure as described in Lubin et al. 2004. It repeatedly invokes `impute.Lubin()`.

Usage

```
impute.boot(X, DL, Z = NULL, K = 5L)
```

Arguments

`X`	A numeric vector, matrix, or data-frame of chemical concentration levels with n subjects and C chemicals to be imputed. Missing values are indicated by NA's. Ideally, a numeric matrix.
`DL`	The detection limit for each chemical as a numeric vector with length equal to C chemicals. Vector must be complete (no NA's); any chemical that has a missing detection limit is not imputed. If DL is a data-frame or matrix with 1 row or 1 column, it is forced as a numeric vector.

Z	Any covariates used in imputing the chemical concentrations. Ideally, a numeric matrix; however, Z can be a factor, vector, or data-frame. Assumed to be complete; observations with missing covariate variables are ignored in the imputation, with a warning printed. If none, enter NULL.
K	A natural number of imputed datasets to generate. Defaults: 5L.

Value

A list of:

X.imputed A number of subjects (n) x number of chemicals (c) x K array of imputed X values.

bootstrap_index A n x K matrix of bootstrap indices selected for the imputation.

indicator.miss A check; the sum of imputed missing values above detection limit, which should be 0.

Note

Note #1: Code was adapted from Erin E. Donahue's original translation of the SAS macro developed from the paper.

Note #2: No seed is set. Please set seed so the same bootstraps are selected.

Note #3: If the length of the DL parameter is greater than the number of components, the smallest value is assumed to be a detection limit. A warning is printed to screen.

References

Lubin, J. H., Colt, J. S., Camann, D., Davis, S., Cerhan, J. R., Severson, R. K., ... Hartge, P. (2004). Epidemiologic Evaluation of Measurement Data in the Presence of Detection Limits. Environmental Health Perspectives, 112(17), 1691–1696. https://doi.org/10.1289/ehp.7199

See Also

Other imputation: `impute.Lubin`, `impute.multivariate.bayesian`, `impute.sub`

Examples

```
data("simdata87")
# Impute using one covariate.
l <- impute.boot(X = simdata87$X.bdl, DL = simdata87$DL, Z = simdata87$Z.sim[, 1], K = 2)
apply(l$X.imputed, 2:3, summary)
```

impute.Lubin *Lubin et al. 2004: Bootstrapping Imputation for One Chemical*

Description

For one chemical, this function creates an imputed dataset using a bootstrap procedure as described in Lubin et al. 2004.

Usage

```
impute.Lubin(chemcol, dlcol, Z = NULL, K = 5L, verbose = FALSE)
```

Arguments

chemcol	A numeric vector, the chemical concentration levels of length C. Censored values are indicated by NA. On original scale.
dlcol	The detection limit of the chemical. A value or a numeric vector of length C. Must be complete; a missing detection limit is ignored.
Z	Any covariates used in imputing the chemical concentrations. Ideally, a numeric matrix; however, Z can be a factor, vector, or data-frame. Assumed to be complete; observations with missing covariate variables are ignored in the imputation, with a warning printed. If none, enter NULL.
K	A natural number of imputed datasets to generate. Defaults: 5L.
verbose	Logical; if TRUE, prints more information. Useful to check for errors in the code. Default: FALSE.

Details

Lubin et al. (2004) evaluate several imputation approaches and show that a multiple imputation procedure using bootstrapping creates unbiased estimates and nominal confidence intervals unless the proportion of missing data is extreme. The authors coded the multiple imputation procedure in a SAS macro that is currently available. We converted the SAS macro into R code.

A single chemical with missing values is imputed. The distribution for the interval-censored data *chemcol* is assumed to be lognormal and censored between 0 and *DL*. After bootstrapping, the values BDL are imputed using the inverse transform method. In other words, generate $u_i \sim Unif(0.0001, dlcol)$ and assign value $F^{-1}(u)$ to x_i for $i = 1, ...n_0$ subjects with chemical values BDL.

impute.Lubin performs the following:

1. Input arguments
2. Obtain bootstrap samples.
3. Generate weights vector.
4. Use Surv function from Survival package to obtain survival object.
5. Use survreg function from Survival package to obtain survival model.

6. Sample from lognormal distribution with beta and variance from survival model as the parameters to obtain upper and lower bounds.
7. Randomly generate value from uniform distribution between the previously obtained upper and lower bounds.
8. Sample from the lognormal distribution to obtain the imputed data value associated with the above uniform value.
9. Repeat for all observations.

Value

A list of:

X.imputed A matrix with n subjects and K imputed datasets is returned.

bootstrap_index A n x K matrix of bootstrap indices selected for the imputation.

indicator.miss A check; the sum of imputed missing values above detection limit, which should be 0.

See Also

Other imputation: `impute.boot`, `impute.multivariate.bayesian`, `impute.sub`

Examples

```
#  Using abind::abind --possibly to impute multiple chemicals
#   ###Example 2: Simulation
  # Apply to an example simulated dataset.
  # A seed of 202 is executed before each run for reproducibility.
  data(simdata87)
  # Impute: 1 chemical
  # No Covariates
  set.seed(202)
  results_Lubin <- impute.Lubin(chemcol = simdata87$X.bdl[, 1], dlcol = simdata87$DL[1], K =
  str(results_Lubin)
  summary(results_Lubin$imputed_values)

  # 1 Covariate
  set.seed(202)
  sim.z1 <- impute.Lubin(simdata87$X.bdl[, 1], simdata87$DL[1],
                         K = 5, Z = simdata87$Z.sim[, 1])
  summary(sim.z1$imputed_values)

  # 2 Covariates
  set.seed(202)
  sim.z2 <- impute.Lubin(simdata87$X.bdl[, 1], simdata87$DL[1],
            K = 5, Z = simdata87$Z.sim[, -2])
  summary(sim.z2$imputed_values)
  summary(sim.z2$bootstrap_index)
```

`impute.multivariate.bayesian` *Multivariate Bayesian Imputation*

Description

Function is in works. Included to collect all imputation arguments in one place.

Usage

```
impute.multivariate.bayesian(X, DL, Z, prior.coeff.mean, prior.cov.mean,
  initial, T, n.burn, K, verbose)
```

Arguments

`X`	A numeric vector, matrix, or data-frame of chemical concentration levels with n subjects and C chemicals to be imputed. Missing values are indicated by NA's. Ideally, a numeric matrix.
`DL`	The detection limit for each chemical as a numeric vector with length equal to C chemicals. Vector must be complete (no NA's); any chemical that has a missing detection limit is not imputed. If DL is a data-frame or matrix with 1 row or 1 column, it is forced as a numeric vector.
`Z`	Any covariates used in imputing the chemical concentrations. Ideally, a numeric matrix; however, Z can be a factor, vector, or data-frame. Assumed to be complete; observations with missing covariate variables are ignored in the imputation, with a warning printed. If none, enter NULL.
`prior.coeff.mean`	The prior mean of number of covariates (p) x C coefficient matrix. The default, entered as NULL, will be a matrix of 1's, given by `J`.
`prior.cov.mean`	The prior mean of covariance matrix. The default, entered as NULL, is an identity matrix with size equal to the number of chemicals. Given by `I`.
`initial`	An optional list of initial values to be specified for the Gibbs Sampler. The default is NULL, which means initial values are generated automatically. See details. If supplied, the list consists of three elements: (1) the coefficient matrix p x C Gamma.initial, (2) the covariance matrix, C x C Sigma.initial, and (3) vec.log.X.initial, a vector of initial log imputed values, which is vectorized by subject, n0C x T. (n0 is total # of missing values.)
`T`	Number of total iterations for the Gibbs Sampler. Defaults: 1000L.
`n.burn`	The burn-in, which is the number of initial iterations to be discarded. Generally, the burn-in can be quite large as the imputed chemical matrices, X.imputed, are formed from the end of the chain – the lowest state used is $T - 10 * K$. Default is 1L (no burn-in).
`K`	A natural number of imputed datasets to generate. Defaults: 5L.
`verbose`	Logical; if TRUE, prints more information. Useful to check for errors in the code. Default: FALSE.

Value

nothing – currently there is no function here.

See Also

Other imputation: `impute.Lubin`, `impute.boot`, `impute.sub`

`impute.sub` *Imputing by Substitution*

Description

Imputes the values below the detection limit with 1/sqrt(2) of that's chemical's detection limit.

Usage

```
impute.sub(X, DL, verbose = FALSE)
```

Arguments

`X`	A numeric vector, matrix, or data-frame of chemical concentration levels with n subjects and C chemicals to be imputed. Missing values are indicated by NA's. Ideally, a numeric matrix.
`DL`	The detection limit for each chemical as a numeric vector with length equal to C chemicals. Vector must be complete (no NA's); any chemical that has a missing detection limit is not imputed. If DL is a data-frame or matrix with 1 row or 1 column, it is forced as a numeric vector.
`verbose`	Logical; if TRUE, prints more information. Useful to check for errors in the code. Default: FALSE.

Details

A matrix of components *X* are interval-censored between zero and different detection limits *DL*. Although *X* may refer to a variable with no obvious *DL*, we consider chemical concentrations *X* with each being partially observed.

Value

A n x C matrix where the BDL values of each chemical are substituted by its detection limit/sqrt(2).

See Also

Other imputation: `impute.Lubin`, `impute.boot`, `impute.multivariate.bayesian`

Examples

```
data("simdata87")
X.sub <- impute.sub(X = simdata87$X.bdl, DL = simdata87$DL, verbose = TRUE)
apply(X.sub, 2, quantile, c(0, 0.02, 0.04, 0.09, 0.25))
# Compare against X.true
round(apply(simdata87$X.true, 2, quantile, c(0.01, 0.05, 0.09, 0.25, 0.5, 0.8, 1)), 5)
```

---

impute.univariate.bayesian.mi

Univariate Bayesian Imputation

Description

Given interval-censored data between 0 and different detection limits (*DL*), `impute.univariate.bayesian.mi` generates K complete datasets using Univariate Bayesian Imputation.

Usage

```
impute.univariate.bayesian.mi(X, DL, T = 1000L, n.burn = 1L, K = 5L,
  verbose = FALSE)
```

Arguments

`X`	A numeric vector, matrix, or data-frame of chemical concentration levels with n subjects and C chemicals to be imputed. Missing values are indicated by NA's. Ideally, a numeric matrix.
`DL`	The detection limit for each chemical as a numeric vector with length equal to C chemicals. Vector must be complete (no NA's); any chemical that has a missing detection limit is not imputed. If DL is a data-frame or matrix with 1 row or 1 column, it is forced as a numeric vector.
`T`	Number of total iterations for the Gibbs Sampler. Defaults: 1000L.
`n.burn`	The burn-in, which is the number of initial iterations to be discarded. Generally, the burn-in can be quite large as the imputed chemical matrices, X.imputed, are formed from the end of the chain – the lowest state used is $T - 10 * K$. Default is 1L (no burn-in).
`K`	A natural number of imputed datasets to generate. Defaults: 5L.
`verbose`	Logical; if TRUE, prints more information. Useful to check for errors in the code. Default: FALSE.

Details

This is the Univariate Bayesian Imputation approach. Only one chemical is imputed at a time. Both the observed and missing data are assumed to follow

$$log(X_{ij}) \sim^{indep} Norm(\mu_j, \sigma_j^2), i = 1, ...n; j = 1, ...C$$

Subjects and chemicals are assumed to be independent. Jeffery's priors are placed on mean and variance for each chemical. Posterior simulation uses data augmentation approach. Initial values were selected as convergence is checked using Gelman-Rubin statistics. Given sample convergence, the K sets of posterior missing values come from the burned Markov chains thinned by K. The imputed values are then substituted for the missing data, forming K complete datasets.

Each of the posterior parameters from MCMC chain–mu.post, sigma.post, and log.x.miss–is saved as a list of mcmc objects (in **coda**) of length # of chemicals. (A list was chosen since the number of missing values n0 might be different from chemical to chemical).

Value

Returns a list that contains: ** Most important.

X.imputed ** An array of n subjects x C chemicals x K imputed sets on the normal scale.

mu.post A list with length equal to the number of chemicals, where each element of list (or for each chemical) is the posterior MCMC chain of the mean is saved as T x 1 `mcmc` object (in **coda**).

sigma.post A list with length equal to the number of chemicals, where each element of list (or for each chemical) is the posterior MCMC chain of the standard deviation, sigma, saved as T x 1 **coda::mcmc** object.

log.x.miss A list with length equal to the number of chemicals, where each element of list is a T x n_{0j} matrix of the log of the imputed missing values, saved as **coda::mcmc** object. n_{0j} is the # of missing values for the jth chemical.

convgd.table A data-frame summarizing convergence with C rows and columns of the Gelman-Rubin statistic and whether the point estimate is less than 1.1. Summary is also printed to the screen.

number.no.converged A check and summary of convgd.table. Total number of parameters that fail to indicate convergence of MCMC chains using Gelman-Rubin statistic. Should be 0.

indicator.miss A check; a vector of indicator variables where the number of missing values are above detection limit. Should be a vector of 0's.

Note

No seed is set in this function. Because bootstraps and MCMC are random, a seed should be set before every use.

Examples

```
# Example 1: 10% BDLs Example -------------------------
# Sample Dataset 87, using 10% BDL Scenario
data(simdata87)
set.seed(472195)
result.imputed <- impute.univariate.bayesian.mi(
   X = simdata87$X.bdl[, 1:6], DL = simdata87$DL[1:6],
   T = 1000, n.burn = 50,  K = 2)
# Did the MCMC converge? A summary of Gelman Statistics is provided.
summary(result.imputed$convg.table)
# Summary of Impouted Values
apply(result.imputed$X.imputed, 2:3, summary)
```

make.quantile.matrix

Making Quantiles of Correlated Index

Description

Scores quantiles from a numeric matrix. If the matrix has values missing between zero and some threshold, say the detection limit, all these missing values (indicated by NA) go into the first quantile.

Usage

```
make.quantile.matrix(X, n.quantiles, place.bdls.in.Q1 = if (anyNA(X))
  TRUE else FALSE, ..., verbose = FALSE)
```

Arguments

X	A numeric matrix. Any missing values are indicated by NA's.
n.quantiles	An integer to specify the number of quantiles to be used categorizing the columns of X, e.g. in quartiles (q = 4), deciles (q = 10), or percentiles (q = 100). Default: 4L.
place.bdls.in.Q1	Logical; if TRUE or X has any missing values, missing values in X are placed in the first quantile of the weighted sum. Otherwise, the data is complete (no missing data) and the data is split equally into quantiles.
...	Further arguments passed to or from other methods. Currently has no effect.
verbose	Logical; if TRUE, prints more information. Useful to check for errors in the code. Default: FALSE.

Details

Produces sample quantiles for a matrix *X* using quantile() function. Names are kept and the 7th quantile algorithm is used. As ties between quantiles may exist, .bincode() is used.

When there is missing data (as indicated by NA's), make.quantile.matrix places all of the censored data into the first quantile. The remaining quantiles are evenly spread over the observed data. A printed message is displaced what the function does.

Value

A matrix of quantiles with rows = nrow(X) and with columns = n.quantiles.

Note

Developed as an accessory function for estimate.wqs().

See Also

quantile

Other wqs: analyze.individually, coef.wqs, do.many.wqs, estimate.wqs, plot.wqs, print.wqs

Examples

```
# Example 1: Make quantiles for first nine chemicals using complete chemical data
data(simdata87)
q <- make.quantile.matrix(simdata87$X.true[, 1:9], 4)
q <- apply(q, 2, as.factor)
summary(q)

# Example 2: Place missing values of first nine chemicals in first quantiles
q2 <- make.quantile.matrix(simdata87$X.bdl[, 1:9], 4, verbose = TRUE)
summary(q2)
```

---

plot.wqs *Histograms of the Weights, Beta1, and WQS using* ggplot2

Description

Plots a WQS object as three histograms of the weights, the overall chemical effect, and WQS across bootstraps. These histograms are returned as **ggplot2** objects.

Usage

```
## S3 method for class 'wqs'
plot(x, filename = "myfile", ...)
```

Arguments

x	An object of class "wqs", usually as a result of estimate.wqs.
filename	DEFUNCT; argument not used; files are no longer saved. Suggested Name is WQS_Plot.
...	DEFUNCT. Arguments no longer passed to ggsave(). This argument currently has no effect.

Details

Three histograms are produced using geom_histogram with ten bins.

Once a Weighted Quantile Sum (WQS) regression is run, the **hist.weights** is a panel of histograms. These are distributions of the weight estimates to determine which chemicals are important in the mixture. Each weight is between 0 and 1 and sum to 1. The individual bootstrapped weight estimates were used to construct the overall chemical index, WQS.

The **hist.beta1** is the distribution of the overall effect of the mixture on the outcome across bootstraps in the training dataset. Due to the constraint in WQS regression, these estimates are either all positive or all negative as dictated by *b1.pos()* argument in `estimate.wqs`. The patterns detected here might be helpful in adjusting the signal function, which is controlled by *signal.fn()* argument in `estimate.wqs`.

The third histogram, **hist.wqs**, shows the range of overall chemical index, or WQS, across each bootstrap. Due to constraints, this always is between 0 and *n.quantiles - 1*.

Plots no longer saved automatically; please save manually using `ggsave`().

Value

A list of histograms

hist.weights A list of **ggplot2** histogram of weights across the bootstrap. Each component consists of a histogram with a weight estimate

hist.beta1 A histogram of the overall chemical mixture effect. This parameter is constrained to be all positive if the b1.pos argument in estimate.wqs() is TRUE.; otherwise, it is FALSE.

hist.WQS A histogram of the overall chemical sum, WQS. Due to constraints, it is always between 0 and *n.quantiles-1*.

See Also

Other wqs: `analyze.individually`, `coef.wqs`, `do.many.wqs`, `estimate.wqs`, `make.quantile.matrix`, `print.wqs`

Examples

```
# Use simulated dataset and set seed for reproducibility.
data(simdata87)
set.seed(23456)
Wa <- estimate.wqs(y = simdata87$y.scenario, X = simdata87$X.true[, 1:9],
                   B = 10, family = "binomial")
plot(Wa)
```

---

pool.mi *Pooling Multiple Imputation Results*

Description

Combine multiple parameter estimates (as used in MI) across the K imputed datasets using Rubin 1996 / 1987 formulas, including: calculating a pooled mean, standard error, missing data statistics, confidence intervals, and p-values.

Usage

```
pool.mi(to.pool, n = 999999, method = c("smallsample", "rubin"),
  alpha = 0.05, verbose = FALSE)
```

Arguments

to.pool	An array of p x 2 x K, where p is the number of parameters to be pooled, 2 refers to the parameters of mean and standard deviation, and K imputation draws. The rownames of to.pool are kept in the results.
n	A number providing the sample size. If nothing is specified, a large sample n = 99999 is assumed. Used in calculating the degrees of freedom; has no effect if K = 1.
method	A string to indicate the method to calculate the degrees of freedom, df.t. If method = "smallsample" (the default) then the Barnard-Rubin adjustment for small degrees of freedom is used. Otherwise, the method from Rubin (1987) is used.
alpha	Type I error used to form the confidence interval. Default: 0.05.
verbose	Logical; if TRUE, prints more information. Useful to check for errors in the code. Default: FALSE.

Details

Stage 3 of Multiple Imputation. The input is an array with p rows referring to the number of parameters to be combined. An estimate and within standard error forms the two columns of the array, which can be easily be taken as the first two columns of the coefficients element of the summary of a glm/lm object. The last dimension is the number of imputations, K. See dataset `wqs.pool.test` as an example.

Uses Rubin's rules to calculate the statistics of an imputed dataset including: the pooled mean, total standard error, a relative increase in variance, fraction of missing information, and 95% Confidence Interval and P-value based on the t-distribution approximation.

Assumes that each complete-data estimate is normally distributed.

Value

A data-frame is returned with the following columns:

pooled.mean The pooled univariate estimate, Qbar, formula (3.1.2) Rubin (1987).

pooled.total.se The total standard error of the pooled estimate, formula (3.1.5) Rubin (1987).

pooled.total.var The total variance of the pooled estimate, formula (3.1.5) Rubin (1987).

se.within The standard error of mean of the variances (i.e. the pooled within-imputation variance), formula (3.1.3) Rubin (1987).

se.between The between-imputation standard error, square root of formula (3.1.4) Rubin (1987).

relative.inc.var(r) The relative increase in variance due to nonresponse, formula (3.1.7) Rubin (1987).

proportion.var.missing(lambda) The proportion of variation due to nonresponse, formula (2.24) Van Buuren (2012).

frac.miss.info The fraction missing information due to nonresponse, formula (3.1.10) Rubin (1987).

df.t The degrees of freedom for t reference distribution, formula (3.1.6) Rubin (1987) or method of Barnard-Rubin (1999) (if method = "smallsample" (default)).

CI The (1-alpha)% confidence interval (CI) for each pooled estimate.

p.value The p-value used to test significance.

Note

Modified the `pool.scalar` (version R 3.4) in the **mice** package to handle multiple parameters at once in an array and combine them. Similar to `mi.inference` in the **norm** package, but the small-sample adjustment is missing.

References

Rubin, D. B. (1987). Multiple Imputation for nonresponse in surveys. New York: Wiley.

Rubin, D. B. (1996). Multiple Imputation After 18+ Years. Journal of the American Statistical Association, 91(434), 473–489. https://doi.org/10.2307/2291635.

Barnard, J., & Rubin, D. B. (1999). Small-Sample Degrees of Freedom with Multiple Imputation. Biometrika, 86(4), 948–955.

Examples

```
#### Example 1: Sample Dataset 87, using 10% BDL Scenario
data(wqs.pool.test)
# Example of the `to.pool` argument
head(wqs.pool.test)

# Pool WQS results and decrease in order of weights.
wqs.results.pooled <-   pool.mi(wqs.pool.test, n = 1000)
weight.dec <- c(order(wqs.results.pooled$pooled.mean[1:14], decreasing = TRUE), 15:16)
wqs.results.pooled <-  wqs.results.pooled[weight.dec, ]
wqs.results.pooled

# When there is 1 estimate (p = 1)
a <- pool.mi(wqs.pool.test[1, , , drop = FALSE], n = 1000)
a
# wqs.results.pooled["dieldrin", ]

# For single imputation (K = 1):
b <- pool.mi(wqs.pool.test[, , 1, drop = FALSE], n = 1000)
b

# Odds ratio and 95% CI using the CLT.
odds.ratio <- exp(wqs.results.pooled[15:16, c("pooled.mean", "CI.1", "CI.2")])
## makeJournalTables :: format.CI( odds.ratio, trim = TRUE, digits = 2, nsmall = 2)
odds.ratio

#  The mice package is suggested for the examples, but not needed for the function.
```

`print.wqs` *Prints the fitted WQS model along with the mean weights.*

Description

Prints the fitted WQS model along with the mean weights. Adjusted from print.lm.

Usage

```
## S3 method for class 'wqs'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	An object of class WQS, from estimate.wqs.
digits	minimal number of *significant* digits, see print.default.
...	further arguments passed to or from other methods.

See Also

Other wqs: analyze.individually, coef.wqs, do.many.wqs, estimate.wqs, make.quantile.matrix, plot.wqs

Examples

```
# See estimate.wqs().

# As base package is always available, there is no need to ever import base
```

---

simdata87 *Simulated Dataset 87*

Description

The 87th dataset from the simulation study with 10 percent of observations were below the detection limit (BDL) based of a real epidemiological dataset. Out of 1000 subjects, fourteen correlated chemicals are completely observed (in X.true). In this simulation design, each chemical was simulated from independent normal distributions.

BDLs were created using the bottom 10th percentile of the true data. Three covariates are considered: the child's age, the child's sex (Male/Female), and the child's ethnicity/race (White, Non-Hispanic White, and Other). After creating a model matrix, male white newborns (age = 0) serves as the reference. The age is simulated from a normal with mean of 3.78 and standard deviation of 1.85 truncated between 0 and 8. The categorical variables are simulated from independent binomial distributions. The outcome will be simulated using a logistic WQS model using the complete data:

$$logit(\mu_i) = -1.25 + log(1.75) * WQS_i + 0.032 * z_{age} + -0.0285 * z_{sex} + 0.540 * z_{His} + 0.120 * z_{other}$$

where

$$WQS_i = \Sigma_{j=1}^{c}(w_j * q_{ij})$$

with four of the 14 weights w_j's being 0.25 and the rest 0. The $q_i j$ refers to the quantile score of the *jth* chemical in the *ith* subject.

Usage

```
data(simdata87)
```

Format

A list that contains:

- y.scenario: a binary outcome (1 = case, 0 = control)
- X.true: 14 chemicals; complete data.
- X.bdl: 14 chemicals with NA's subbed for the bottom 10th percentile of the true values.
- DL: The detection limit. Here, found to be the 10th percentile of X.true
- n0: A vector of length 14 indicating the number of non-detects.
- delta: A vector of length 14 indicating whether the chemical is observed (1) or not (0)
- Z.sim: A data-frame of covariates consisting of:
  - Age: A continuous covariate of child's age , simulated using normal with mean of 3.78 and sd of 1.85, truncated between 0 and 8, the maximum age of leukemia.
  - Female: Binary variable child's sex, simulated using the proportion of females (0.42) by binomial distribution.
  - Hispanic, Non-Hispanic_Others: Two indicator variables of child's race/ethnicity, sampled from independent binomial distributions (proportion of Hispanic: 0.33; proportion of Other: 0.23).
- time: the time it took to simulate the data.

References

Ward, M. H., Colt, J. S., Metayer, C., Gunier, R. B., Lubin, J., Crouse, V., ... Buffler, P. A. (2009). Residential Exposure to Polychlorinated Biphenyls and Organochlorine Pesticides and Risk of Childhood Leukemia. Environmental Health Perspectives, 117(6), 1007–1013. https://doi.org/10.1289/ehp.0900583

Examples

```
simdata87 <- data(simdata87)
```

---

wqs.pool.test *Combining WQS Regression Estimates*

Description

wqs.pool.test was produced to demonstrate `pool.mi()`. The `simdata87` was first imputed multiple times to form an imputed X array. This was an example from `impute.univariate.bayesian.mi`. Multiple WQS regressions were run on the imputed X array to produce an array of WQS parameter estimates, *wqs.pool.test*.

Usage

```
data(wqs.pool.test)
```

Format

An array of 16 x 2 x 3, with

- 16 parameters as the rows (The 14 weights, Intercept, and WQS estimate of a WQS model),
- 2 refers to the parameters of mean and standard deviation
- K=3 complete imputed datasets.

See Also

`pool.mi`

Examples

```
wqs.pool.test <- data(wqs.pool.test)

# stage_3_pool_mi_example
```