Package: imp4p (via r-universe)

October 8, 2024

Type Package

Title Imputation for Proteomics

Version 1.2

Author Quentin Giai Gianetto

Maintainer Quentin Giai Gianetto <quentin2g@yahoo.fr>

Description Functions to analyse missing value mechanisms and to impute data sets in the context of bottom-up MS-based proteomics.

License GPL-3

Depends R (>= 3.3.0), Iso, stats, truncnorm, norm, missForest, missMDA

Encoding UTF-8

Imports Rcpp (>= 0.12.8)

LinkingTo Rcpp

NeedsCompilation yes

Date/Publication 2021-09-02 21:10:09 UTC

Repository https://jacky11.r-universe.dev

RemoteUrl https://github.com/cran/imp4p

RemoteRef HEAD

RemoteSha 5ba62f8299815b94cdad66fe85ac7e2ba702d6d5

Contents

`imp4p-package` *Introduction to the IMP4P package*

Description

This package provides functions to analyse missing value mechanisms in the context of bottom-up MS-based quantitative proteomics.

It allows estimating a mixture model of missing completely-at-random (MCAR) values and missing not-at-random (MNAR) values.

It also contains functions allowing the imputation of missing values under hypotheses of MCAR and/or MNAR values.

The main functions of this package are the `estim.mix` (estimation of a model of MCAR and MNAR (left-censored) values), `impute.mi` (multiple imputation) and `impute.mix` (imputation based on a decision rule). It provides also several imputation algorithms for MS-based data. They can be used to impute matrices containing peptide intensities (as Maxquant outputs for instance).

Missing values has to be indicated with NA and a log-2 transformation of the intensities has to be applied before using these functions. An example for using this package from MaxQuant outputs is provided in Giai Gianetto Q. (2021).

More explanations and details on the functions of this package are available in Giai Gianetto Q. et al. (2020) (doi: doi: 10.1101/2020.05.29.122770).

Author(s)

Maintainer: Quentin Giai Gianetto <quentin2g@yahoo.fr>

References

Giai Gianetto, Q., Wieczorek S., Couté Y., Burger, T. (2020). A peptide-level multiple imputation strategy accounting for the different natures of missing values in proteomics data. bioRxiv 2020.05.29.122770; doi: doi: 10.1101/2020.05.29.122770

Giai Gianetto, Q. (2021) Statistical analysis of post-translational modifications quantified by label-free proteomics across multiple biological conditions with R: illustration from SARS-CoV-2 infected cells. (pasteur-03243433)

`estim.bound` *Estimation of lower and upper bounds for missing values.*

Description

This function allows estimating lower and upper bounds for missing values of an input matrix. It can be used before to use the functions `prob.mcar` and `prob.mcar.tab`.

Usage

```
estim.bound(tab, conditions, q=0.95)
```

Arguments

`tab`	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to the intensities measured in an experimental sample, and each row to the ones of an identified peptide.
`conditions`	A vector of factors indicating the biological condition to which each sample belongs.
`q`	A numeric value allowing to define confidence intervals for missing values (see Details).

Details

In each condition, this function estimates lower and upper bounds for missing values of row `i` by:

`upper(i)=max(tab[i,]);`

`lower(i)=max(tab[i,])-quant_diff(q);`

where `quant_diff(q)` corresponds to a quantile value of the differences between the maximum and the minimum of the observed values for all the peptides in the condition. As a result, if `q` is close to 1, `quant_diff(q)` represents an extrem value between the maximum and the minimum of the intensity values in a condition for a peptide.

Value

A list composed of:

`tab.lower`	A matrix with the lower bounds for each missing value in `tab`.
`tab.upper`	A matrix with the upper bounds for each missing value in `tab`.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

See Also

prob.mcar,prob.mcar.tab

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600,pi.mcar=0.2,para=0.5,nb.cond=2,nb.repbio=3,
nb.sample=5,m.c=25,sd.c=2,sd.rb=0.5,sd.r=0.2);

data=res.sim$dat.obs;
cond=res.sim$conditions;

#Estimation of lower and upper bounds for each missing value
res=estim.bound(data,conditions=cond);
```

---

estim.mix *Estimation of a mixture model of MCAR and MNAR values in each column of a data matrix.*

Description

This function allows estimating a mixture model of MCAR and MNAR values in each column of data sets similar to the ones which can be studied in MS-based quantitative proteomics. Such data matrices contain intensity values of identified peptides.

Usage

```
estim.mix(tab, tab.imp, conditions, x.step.mod=150, x.step.pi=150,
nb.rei=200)
```

Arguments

tab	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
tab.imp	A matrix where the missing values of tab have been imputed under the assumption that they are all MCAR. For instance, such a matrix can be obtained by using the function impute.slsa of this package.
conditions	A vector of factors indicating the biological condition to which each column (experimental sample) belongs.
x.step.mod	The number of points in the intervals used for estimating the cumulative distribution functions of the mixing model in each column.

`x.step.pi` The number of points in the intervals used for estimating the proportion of MCAR values in each column.

`nb.rei` The number of initializations of the minimization algorithm used to estimate the proportion of MCAR values (see Details).

Details

This function aims to estimate the following mixture model in each column:

$F_{tot}(x) = \pi_{na} \times F_{na}(x) + (1 - \pi_{na}) \times F_{obs}(x)$

$F_{na}(x) = \pi_{mcar} \times F_{tot}(x) + (1 - \pi_{mcar}) \times F_{mnar}(x)$

where π_{na} is the proportion of missing values, π_{mcar} is the proportion of MCAR values, F_{tot} is the cumulative distribution function (cdf) of the complete values, F_{na} is the cdf of the missing values, F_{obs} is the cdf of the observed values, and F_{mnar} is the cdf of the MNAR values.

To estimate this model, a first step consists to compute a rough estimate of F_{na} by assuming that all missing values are MCAR (thanks to the argument `tab.imp`). This rough estimate is noted $\hat{F}_{na}$.

In a second step, the proportion of MCAR values is estimated. To do so, the ratio

$\hat{\pi}(x) = (1 - \hat{F}_{na}(x))/(1 - \hat{F}_{tot}(x))$

is computed for different x, where

$\hat{F}_{tot}(x) = \pi_{na} \times \hat{F}_{na}(x) + (1 - \pi_{na}) \times \hat{F}_{obs}(x)$

with $\hat{F}_{obs}$ the empirical cdf of the observed values.

Next, the following minimization is performed:

$\min_{1>k>0, a>0, d>0} f(k, a, d)$

where

$f(k, a, d) = \sum_x \frac{1}{s(x)^2} \times [\hat{\pi}(x) - k - (1 - k)\frac{\exp(-a \times [x - lower_x]^d)}{1 - \hat{F}_{tot}(x)}]^2$

where $s(x)^2$ is an estimate of the asymptotic variance of $\hat{\pi}(x)$, $lower_x$ is an estimate of the minimum of the complete values. To perform this minimization, the function `optim` with the method "L-BFGS-B" is used. Because it is function of its initialization, it is possible to reinitialize a number of times the minimisation algorithm with the argument `nb.rei`: the parameters leading to the lowest minimum are next kept.

Once `k`, `a` and `d` are estimated, one can use several methods to estimate π_{mcar}: it is estimated by k;

Value

A list composed of:

`abs.pi` A numeric matrix containing the intervals used for estimating the ratio `(1-F_na(x))/(1-F_tot(x))` in each column.

`pi.init` A numeric matrix containing the estimated ratios `(1-F_na(x))/(1-F_tot(x))` where `x` belongs to `abs.pi[,j]` for each sample `j`.

`var.pi.init` A numeric matrix containing the estimated asymptotic variances of `pi.init`.

trend.pi.init	A numeric matrix containing the estimated trend of the model used in the minimization algorithm.
abs.mod	A numeric vector containing the interval used for estimating the mixture models in each column.
pi.na	A numeric vector containing the proportions of missing values in each column.
F.na	A numeric matrix containing the estimated cumulative distribution functions of missing values in each column on the interval abs.mod.
F.tot	A numeric matrix containing the estimated cumulative distribution functions of complete values in each column on the interval abs.mod.
F.obs	A numeric matrix containing the estimated cumulative distribution functions of observed values in each column on the interval abs.mod.
pi.mcar	A numeric vector containing the estimations of the proportion of MCAR values in each column.
MinRes	A numeric matrix containing the three parameters of the model used in the minimization algorithm (three first rows), and the value of minimized function.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

See Also

impute.slsa

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600);

#Imputation of missing values with a MCAR-devoted algorithm: here the slsa algorithm
dat.slsa=impute.slsa(tab=res.sim$dat.obs,conditions=res.sim$condition,repbio=res.sim$repbio);

#Estimation of the mixture model
res=estim.mix(tab=res.sim$dat.obs, tab.imp=dat.slsa, conditions=res.sim$condition);
```

---

fast_apply_nb_na *Function similar to the function* apply(X,dim,function(x)sum(is.na(x))).

Description

This function is similar to the function apply(X,dim,function(x)sum(is.na(x))) but written thanks to the Rcpp package and therefore faster than apply(X,dim,function(x)sum(is.na(x))).

Usage

```
fast_apply_nb_na(X, dim)
```

Arguments

X	A data matrix containing numeric and missing values.
dim	A numeric value: 1 if the number of missing values has to be computed for each row of X, or 2 if it has to be computed for each column of X.

Value

A numeric vector containing the number of missing values in either each row or each column of X.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

Examples

```
## The function is currently defined as
##function (X, dim)
##{
##    .Call("imp4p_fast_apply_nb_na", PACKAGE = "imp4p", X, dim)
##  }
##
## You can compare the execution time with a traditional apply function by
## library(rbenchmark)
## res.sim=sim.data(nb.pept=2000,nb.miss=600);
## benchmark(fast_apply_nb_na(res.sim$dat.obs, 1),
##           apply(res.sim$dat.obs,1,function(x)sum(is.na(x))))
##
##
##
```

---

fast_apply_nb_not_na *Function similar to the function* apply(X,dim,function(x)sum(!is.na(x))).

Description

This function is similar to the function apply(X,dim,function(x)sum(!is.na(x))) but written thanks to the Rcpp package and therefore faster than apply(X,dim,function(x)sum(!is.na(x))).

Usage

```
fast_apply_nb_not_na(X, dim)
```

Arguments

X	A data matrix containing numeric and missing values.
dim	A numeric value: 1 if the number of observed values has to be computed for each row of X, or 2 if it has to be computed for each column of X.

Value

A numeric vector containing the number of observed values in either each row or each column of X.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

Examples

```
## The function is currently defined as
##function (X, dim)
##{
##    .Call("imp4p_fast_apply_nb_not_na", PACKAGE = "imp4p", X,
##        dim)
##  }
##
## You can compare the execution time with a traditional apply function by
## library(rbenchmark)
## res.sim=sim.data(nb.pept=2000,nb.miss=600);
## benchmark(fast_apply_nb_not_na(res.sim$dat.obs, 1),
##           apply(res.sim$dat.obs,1,function(x)sum(!is.na(x))))
```

---

fast_apply_sd_na_rm_T *Function similar to the function* apply(X,dim,sd,na.rm=TRUE).

Description

This function is similar to the function apply(X,dim,sd,na.rm=TRUE) but written thanks to the Rcpp package and therefore faster than apply(X,dim,sd,na.rm=TRUE).

Usage

```
fast_apply_sd_na_rm_T(X, dim)
```

Arguments

X	A data matrix containing numeric and missing values.
dim	A numeric value: 1 if the standard deviation has to be computed for each row of X, or 2 if the standard deviation has to be computed for each column of X.

Value

A numeric vector containing the standard deviation of observed values in either each row or each column of X.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

Examples

```
## The function is currently defined as
##function (X, dim)
##{
##    .Call("imp4p_fast_apply_sd_na_rm_T", PACKAGE = "imp4p", X,
##        dim)
##  }
##
## You can compare the execution time with a traditional apply function by
## library(rbenchmark)
## res.sim=sim.data(nb.pept=2000,nb.miss=600);
## benchmark(fast_apply_sd_na_rm_T(res.sim$dat.obs, 1),
##           apply(res.sim$dat.obs,1,sd,na.rm=TRUE))
```

---

fast_apply_sum_na_rm_T

Function similar to the function `apply(X,dim,sum,na.rm=TRUE)`.

Description

This function is similar to the function `apply(X,dim,sum,na.rm=TRUE)` but written thanks to the Rcpp package and therefore faster than `apply(X,dim,sum,na.rm=TRUE)`.

Usage

```
fast_apply_sum_na_rm_T(X, dim)
```

Arguments

X	A data matrix containing numeric and missing values.
dim	A numeric value: 1 if the sum has to be computed for each row of X, or 2 if the sum has to be computed for each column of X.

Value

A numeric vector containing the sum of observed values in either each row or each column of X.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

Examples

```
## The function is currently defined as
##function (X, dim)
##{
##    .Call("imp4p_fast_apply_sum_na_rm_T", PACKAGE = "imp4p",
##        X, dim)
##  }
##
## You can compare the execution time with a traditional apply function by
## library(rbenchmark)
## res.sim=sim.data(nb.pept=2000,nb.miss=600);
## benchmark(fast_apply_sum_na_rm_T(res.sim$dat.obs, 1),
##           apply(res.sim$dat.obs,1,sum,na.rm=TRUE))
```

`fast_sim` *Function to compute similarity measures between a vector and each row of a matrix.*

Description

This function allows computing a similarity measure between a vector and each row of a matrix. The similarity measure is defined by d^2 where d is the Euclidean distance between the vector and each row. It is implemented thanks to the RCpp package.

Usage

```
fast_sim(prot, mat)
```

Arguments

`prot`	A numeric vector containing numeric and missing values.
`mat`	A data matrix containing numeric and missing values.

Value

A numeric vector containing the values of the similarity measures between the `prot` vector and each row of the `mat` matrix.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=20000,nb.miss=1000);

#Fast computation of similarities
fast_sim(res.sim$dat.obs[1,],res.sim$dat.obs);
```

`gen.cond` *Function allowing to create a vector indicating the membership of each sample to a condition.*

Description

This function creates a vector of factors where each element refers to a condition to which a sample belongs.

Usage

```
gen.cond(nb_cond=2,nb_sample=5)
```

Arguments

`nb_cond`	Number of biological conditions.
`nb_sample`	Number of samples in each condition.

Value

A vector of factors of length nb_cond*nb_sample.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

Examples

```
cond=gen.cond(nb_cond=2,nb_sample=6)
#[1] 1 1 1 1 1 1 2 2 2 2 2 2
#Levels: 1 2
```

`impute.igcda` *Imputing missing values by assuming that the distribution of complete values is Gaussian in each column of an input matrix. This algorithm is named "Imputation under a Gaussian Complete Data Assumption" (IGCDA).*

Description

This function allows imputing missing values under the assumption that the distribution of complete values has to be Gaussian in each column.

Note that the imputed values are not necessary small values (compared to observed values).

Usage

```
impute.igcda(tab, tab.imp, conditions, q=0.95)
```

Arguments

tab	A numeric vector or matrix with observed and missing values.
tab.imp	A matrix where the missing values of tab have been imputed under the assumption that they are all MCAR. For instance, such a matrix can be obtained by using the function impute.slsa of this package.
conditions	A vector of factors indicating the biological condition to which each column (experimental sample) belongs.
q	A quantile value (see Details).

Details

The mean and variance of the Gaussian distribution are determined using a linear regression between the quantiles of the observed values q_{obs} and the ones of the standard normal distribution q_{N(0,1)}.

The quantile value is used for determining the minimum of imputed values. This minimum is determined by the minimum observed value in the dataset minus quant_diff(q) where quant_diff(q) corresponds to a quantile value of the differences between the maximum and the minimum of the observed values for all the peptides in the condition. As a result, if q is close to 1, quant_diff(q) represents an extrem value between the maximum and the minimum of the intensity values in a condition for a peptide.

Value

The numeric input matrix with imputed values. The distribution of the intensity values in each of its columns is supposed to be Gaussian.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600);

#Imputation of missing values with a MCAR-devoted algorithm: here the slsa algorithm
dat.slsa=impute.slsa(tab=res.sim$dat.obs,conditions=res.sim$condition,repbio=res.sim$repbio);

#Imputation of missing values under a Gaussian assumption
dat.gauss=impute.igcda(tab=res.sim$dat.obs, tab.imp=dat.slsa, conditions=res.sim$conditions);
```

impute.mi *Imputation of data sets containing peptide intensities with a multiple imputation strategy.*

Description

This function allows imputing data sets containing peptide intensities with a multiple imputation strategy distinguishing MCAR and MNAR values. For details, see Giai Gianetto Q. et al. (2020) (doi: doi: 10.1101/2020.05.29.122770).

Usage

```
impute.mi(tab, conditions, repbio=NULL, reptech=NULL, nb.iter=3, nknn=15, selec=1000,
siz=900, weight=1, ind.comp=1, progress.bar=TRUE, x.step.mod=300, x.step.pi=300,
nb.rei=100, q=0.95, methodMCAR="mle",ncp.max=5,
maxiter = 10, ntree = 100, variablewise = FALSE,
decreasing = FALSE, verbose = FALSE, mtry = floor(sqrt(ncol(tab))),
replace = TRUE, classwt = NULL, cutoff = NULL, strata = NULL, sampsize = NULL,
nodesize = NULL, maxnodes = NULL, xtrue = NA,
parallelize = c('no', 'variables', 'forests'),
methodMNAR="igcda",q.min = 0.025, q.norm = 3, eps = 0,
distribution = "unif", param1 = 3, param2 = 1, R.q.min=1);
```

Arguments

tab	A data matrix containing only numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
conditions	A vector of factors indicating the biological condition to which each column (experimental sample) belongs.
repbio	A vector of factors indicating the biological replicate to which each column belongs. Default is NULL (no experimental design is considered).
reptech	A vector of factors indicating the technical replicate to which each column belongs. Default is NULL (no experimental design is considered).
nb.iter	The number of iterations used for the multiple imputation method (see mi.mix).
methodMCAR	The method used for imputing MCAR data. If methodMCAR="mle" (default), then the impute.mle function is used (imputation using an EM algorithm). If methodMCAR="pca", then the impute.PCA function is used (imputation using Principal Component Analysis). If methodMCAR="rf", then the impute.RF function is used (imputation using Random Forest). Else, the impute.slsa function is used (imputation using Least Squares on nearest neighbours).
methodMNAR	The method used for imputing MNAR data. If methodMNAR="igcda" (default), then the impute.igcda function is used. Else, the impute.pa function is used.
nknn	The number of nearest neighbours used in the SLSA algorithm (see impute.slsa).

selec	A parameter to select a part of the dataset to find nearest neighbours between rows. This can be useful for big data sets (see impute.slsa).
siz	A parameter to select a part of the dataset to perform imputations with the MCAR-devoted algorithm. This can be useful for big data sets (see mi.mix).
weight	The way of weighting in the algorithm (see impute.slsa).
ind.comp	If ind.comp=1, only nearest neighbours without missing values are selected to fit linear models (see impute.slsa). Else, they can contain missing values.
progress.bar	If TRUE, a progress bar is displayed.
x.step.mod	The number of points in the intervals used for estimating the cumulative distribution functions of the mixing model in each column (see estim.mix).
x.step.pi	The number of points in the intervals used for estimating the proportion of MCAR values in each column (see estim.mix).
nb.rei	The number of initializations of the minimization algorithm used to estimate the proportion of MCAR values (see Details) (see estim.mix).
q	A quantile value (see impute.igcda).
ncp.max	parameter of the impute.PCA function.
maxiter	parameter of the impute.RF function.
ntree	parameter of the impute.RF function.
variablewise	parameter of the impute.RF function.
decreasing	parameter of the impute.RF function.
verbose	parameter of the impute.RF function.
mtry	parameter of the impute.RF function.
replace	parameter of the impute.RF function.
classwt	parameter of the impute.RF function.
cutoff	parameter of the impute.RF function.
strata	parameter of the impute.RF function.
sampsize	parameter of the impute.RF function.
nodesize	parameter of the impute.RF function.
maxnodes	parameter of the impute.RF function.
xtrue	parameter of the impute.RF function.
parallelize	parameter of the impute.RF function.
q.min	parameter of the impute.pa function.
q.norm	parameter of the impute.pa function.
eps	parameter of the impute.pa function.
distribution	parameter of the impute.pa function.
param1	parameter of the impute.pa function.
param2	parameter of the impute.pa function.
R.q.min	parameter of the impute.pa function.

Details

First, a mixture model of MCAR and MNAR values is estimated in each column of `tab`. This model is used to estimate probabilities that each missing value is MCAR. Then, these probabilities are used to perform a multiple imputation strategy (see `mi.mix`). Rows with no value in a condition are imputed using the `impute.pa` function. More details and explanations can be bound in Giai Gianetto (2020).

Value

The input matrix `tab` with imputed values instead of missing values.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

References

Giai Gianetto, Q., Wieczorek S., Couté Y., Burger, T. (2020). A peptide-level multiple imputation strategy accounting for the different natures of missing values in proteomics data. bioRxiv 2020.05.29.122770; doi: doi: 10.1101/2020.05.29.122770

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600,nb.cond=2);

#Imputation of the dataset noting the conditions to which the samples belong.
result=impute.mi(tab=res.sim$dat.obs, conditions=res.sim$conditions);

#Imputation of the dataset noting the conditions to which the samples belong
#and also their biological replicate, and using the SLSA method for the MCAR values
result=impute.mi(tab=res.sim$dat.obs, conditions=res.sim$conditions,
repbio=res.sim$repbio, methodMCAR = "slsa");

#For large data sets, the SLSA imputation can be accelerated thanks to the selec parameter
#and the siz parameter (see impute.slsa and mi.mix)
#but it may result in a less accurate data imputation. Note that selec has to be greater than siz.
#Here, nb.iter is fixed to 3
result1=impute.mi(tab=res.sim$dat.obs, conditions=res.sim$conditions, progress.bar=TRUE,
selec=400, siz=300, nb.iter=3, methodMCAR = "slsa", methodMNAR = "igcda");
```

`impute.mix` *Imputation using a decision rule under an assumption of a mixture of MCAR and MNAR values.*

Description

This function allows imputing data sets with a MCAR-devoted algorithm and a MNAR-devoted algorithm using probabilities that missing values are MCAR. If such a probability is superior to a chosen threshold, then the MCAR-devoted algorithm is used, otherwise it is the MNAR-devoted algorithm. For details, see Giai Gianetto, Q. et al. (2020) (doi: doi: 10.1101/2020.05.29.122770).

Usage

```
impute.mix(tab, prob.MCAR, threshold, conditions, repbio=NULL, reptech=NULL,
methodMCAR="mle",nknn=15,weight=1, selec="all", ind.comp=1, progress.bar=TRUE, q=0.95,
ncp.max=5, maxiter = 10, ntree = 100, variablewise = FALSE, decreasing = FALSE,
verbose = FALSE, mtry = floor(sqrt(ncol(tab))), replace = TRUE,classwt = NULL,
cutoff = NULL, strata = NULL, sampsize = NULL, nodesize = NULL, maxnodes = NULL,
xtrue = NA, parallelize = c('no', 'variables', 'forests'),
methodMNAR="igcda", q.min = 0.025, q.norm = 3, eps = 0, distribution = "unif",
param1 = 3, param2 = 1, R.q.min=1)
```

Arguments

tab	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
prob.MCAR	A matrix of probabilities that each missing value is MCAR. For instance such a matrix can be obtained from the function prob.mcar.tab of this package.
threshold	A value such that if the probability that a missing value is MCAR is superior to it, then a MCAR-devoted algorithm is used, otherwise it is a MNAR-devoted algorithm that is used.
conditions	A vector of factors indicating the biological condition to which each column (experimental sample) belongs.
repbio	A vector of factors indicating the biological replicate to which each column belongs. Default is NULL (no experimental design is considered).
reptech	A vector of factors indicating the technical replicate to which each column belongs. Default is NULL (no experimental design is considered).
methodMCAR	The method used for imputing MCAR data. If methodi="mle" (default), then the impute.mle function is used (imputation using an EM algorithm). If methodi="pca", then the impute.PCA function is used (imputation using Principal Component Analysis). If methodi="rf", then the impute.RF function is used (imputation using Random Forest). Else, the impute.slsa function is used (imputation using Least Squares on nearest neighbours).
methodMNAR	The method used for imputing MNAR data. If methodMNAR="igcda" (default), then the impute.igcda function is used. Else, the impute.pa function is used.
nknn	The number of nearest neighbours used in the SLSA algorithm (see impute.slsa).
weight	The way of weighting in the algorithm (see impute.slsa).
selec	A parameter to select a part of the dataset to find nearest neighbours between rows. This can be useful for big data sets (see impute.slsa).

ind.comp	If ind.comp=1, only nearest neighbours without missing values are selected to fit linear models (see impute.slsa). Else, they can contain missing values.
progress.bar	If TRUE, a progress bar is displayed.
q	A quantile value (see impute.igcda).
ncp.max	parameter of the impute.PCA function.
maxiter	parameter of the impute.RF function.
ntree	parameter of the impute.RF function.
variablewise	parameter of the impute.RF function.
decreasing	parameter of the impute.RF function.
verbose	parameter of the impute.RF function.
mtry	parameter of the impute.RF function.
replace	parameter of the impute.RF function.
classwt	parameter of the impute.RF function.
cutoff	parameter of the impute.RF function.
strata	parameter of the impute.RF function.
sampsize	parameter of the impute.RF function.
nodesize	parameter of the impute.RF function.
maxnodes	parameter of the impute.RF function.
xtrue	parameter of the impute.RF function.
parallelize	parameter of the impute.RF function.
q.min	parameter of the impute.pa function.
q.norm	parameter of the impute.pa function.
eps	parameter of the impute.pa function.
distribution	parameter of the impute.pa function.
param1	parameter of the impute.pa function.
param2	parameter of the impute.pa function.
R.q.min	parameter of the impute.pa function.

Details

The missing values for which prob.MCAR is superior to a chosen threshold are imputed with one of the MCAR-devoted imputation methods (impute.mle, impute.RF, impute.PCA or impute.slsa). The other missing values are considered MNAR and imputed with impute.igcda. More details and explanations can be bound in Giai Gianetto (2020).

Value

The input matrix tab with imputed values instead of missing values.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

References

Giai Gianetto, Q., Wieczorek S., Couté Y., Burger, T. (2020). A peptide-level multiple imputation strategy accounting for the different natures of missing values in proteomics data. bioRxiv 2020.05.29.122770; doi: doi: 10.1101/2020.05.29.122770

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600);

#Fast imputation of missing values with the impute.rand algorithm
dat.rand=impute.rand(tab=res.sim$dat.obs,conditions=res.sim$condition);

#Estimation of the mixture model
res=estim.mix(tab=res.sim$dat.obs, tab.imp=dat.rand, conditions=res.sim$condition);

#Computing probabilities to be MCAR
born=estim.bound(tab=res.sim$dat.obs,conditions=res.sim$condition);
proba=prob.mcar.tab(born$tab.upper,res);

#Imputation under the assumption of MCAR and MNAR values
tabi=impute.mix(tab=res.sim$dat.obs, prob.MCAR=proba, threshold=0.5, conditions=res.sim$conditions,
repbio=res.sim$repbio, methodMCAR="slsa", methodMNAR="igcda", nknn=15, weight=1, selec="all",
ind.comp=1, progress.bar=TRUE);
```

`impute.mle` *Imputing missing values using a maximum likelihood estimation (MLE).*

Description

Imputing missing values using the EM algorithm proposed in section 5.4.1 of Schafer (1997). The function is based on the imp.norm function of the R package norm.

Usage

```
impute.mle(tab, conditions)
```

Arguments

`tab`	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
`conditions`	A vector of factors indicating the biological condition to which each sample belongs.

Details

See section 5.4.1 of Schafer (1997) for the theory. It is built from functions proposed in the R package norm.

Value

The input matrix `tab` with imputed values instead of missing values.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

References

Schafer, J. L. (1997). Analysis of incomplete multivariate data. Chapman and Hall/CRC.

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600,nb.cond=2);

#Imputation of missing values with the mle algorithm
dat.mle=impute.mle(tab=res.sim$dat.obs,conditions=res.sim$condition);
```

---

impute.pa — *Imputation of peptides having no value in a biological condition (present in a condition / absent in another).*

Description

This function imputes missing values by small values.

Usage

```
impute.pa(tab, conditions, q.min = 0.025, q.norm = 3, eps = 0,
distribution = "unif", param1 = 3, param2 = 1, R.q.min=1)
```

Arguments

`tab`	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
`conditions`	A vector of factors indicating the biological condition to which each column (experimental sample) belongs.

q.min	A quantile value of the observed values allowing defining the maximal value which can be generated. This maximal value is defined by the quantile q.min of the observed values distribution minus eps. Default is 0.025 (the maximal value is the 2.5 percentile of observed values minus eps).
q.norm	A quantile value of a normal distribution allowing defining the minimal value which can be generated. Default is 3 (the minimal value is the maximal value minus qn*median(sd(observed values)) where sd is the standard deviation of a row in a condition).
eps	A value allowing defining the maximal value which can be generated. This maximal value is defined by the quantile q.min of the observed values distribution minus eps. Default is 0.
distribution	Distribution used to generated missing values. You have the choice between "unif" for the uniform distribution, "beta" for the Beta distribution or "dirac" for the Dirac distribution. Default is "unif".
param1	Parameter shape1 of the Beta distribution.
param2	Parameter shape2 of the Beta distribution.
R.q.min	Parameter used for the Dirac distribution. In this case, all the missing values are imputed by a single value which is equal to R.q.min*quantile(tab[,j], probs=q.min, na.rm=T). Default is 1 : the imputed value is the qmin quantile of observed values.

Details

This function replaces the missing values in a column by random draws from a specified distribution. The value of eps can be interpreted as a minimal fold-change value above which the present/absent peptides appear.

Value

A list composed of :

- tab.imp : the input matrix tab with imputed values instead of missing values.

- para : the parameters of the distribution which has been used to impute.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600);

#Imputation of the simulated data set with small values
data.small.val=impute.pa(res.sim$dat.obs,res.sim$conditions);
```

impute.PCA *Imputing missing values using Principal Components Analysis.*

Description

Imputing missing values using the algorithm proposed by Josse and Husson (2013). The function is based on the imputePCA function of the R package missMDA.

Usage

```
impute.PCA(tab, conditions, ncp.max=5)
```

Arguments

tab	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
conditions	A vector of factors indicating the biological condition to which each sample belongs.
ncp.max	integer corresponding to the maximum number of components to test (used in the estim_ncpPCA function of R package missMDA).

Details

See Josse and Husson (2013) for the theory. It is built from functions proposed in the R package missMDA.

Value

The input matrix tab with imputed values instead of missing values.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

References

Josse, J & Husson, F. (2013). Handling missing values in exploratory multivariate data analysis methods. Journal de la SFdS. 153 (2), pp. 79-99.

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600,nb.cond=2);

#Imputation of missing values with PCA
dat.pca=impute.PCA(tab=res.sim$dat.obs,conditions=res.sim$condition);
```

`impute.rand` *Imputation of peptides with a random value.*

Description

For each row (peptide), this function imputes missing values by random values following a Gaussian distribution.

Usage

```
impute.rand(tab, conditions)
```

Arguments

`tab`	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
`conditions`	A vector of factors indicating the biological condition to which each column (experimental sample) belongs.

Details

For each row (peptide), this function imputes missing values by random values following a Gaussian distribution centered on the mean of the observed values in the condition for the specific peptide and with a standard deviation equal to the first quartile of the distribution of the standard deviation the values observed for all the peptides. Rows with only missing values in a condition are not imputed (the `impute.pa` function can be used for this purpose).

Value

The input matrix `tab` with imputed values instead of missing values.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600);

#Imputation of the simulated data set with random values
data.rand=impute.rand(res.sim$dat.obs,res.sim$conditions);
```

impute.RF *Imputing missing values using Random Forest.*

Description

Imputing missing values using the algorithm proposed by Stekhoven and Buehlmann (2012). The function is based on the missForest function of the R package missForest.

Usage

```
impute.RF(tab, conditions,
                    maxiter = 10, ntree = 100, variablewise = FALSE,
                    decreasing = FALSE, verbose = FALSE,
                    mtry = floor(sqrt(ncol(tab))), replace = TRUE,
                    classwt = NULL, cutoff = NULL, strata = NULL,
                    sampsize = NULL, nodesize = NULL, maxnodes = NULL,
                    xtrue = NA, parallelize = c('no', 'variables', 'forests'))
```

Arguments

tab	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
conditions	A vector of factors indicating the biological condition to which each sample belongs.
maxiter	parameter of the missForest function (missForest R package).
ntree	parameter of the missForest function (missForest R package).
variablewise	parameter of the missForest function (missForest R package).
decreasing	parameter of the missForest function (missForest R package).
verbose	parameter of the missForest function (missForest R package).
mtry	parameter of the missForest function (missForest R package).
replace	parameter of the missForest function (missForest R package).
classwt	parameter of the missForest function (missForest R package).
cutoff	parameter of the missForest function (missForest R package).
strata	parameter of the missForest function (missForest R package).
sampsize	parameter of the missForest function (missForest R package).
nodesize	parameter of the missForest function (missForest R package).
maxnodes	parameter of the missForest function (missForest R package).
xtrue	parameter of the missForest function (missForest R package).
parallelize	parameter of the missForest function (missForest R package).

Details

See Stekhoven and Buehlmann (2012) for the theory. It is built from functions proposed in the R package missForest.

Value

The input matrix `tab` with imputed values instead of missing values.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

References

Stekhoven, D.J. and Buehlmann, P. (2012), 'MissForest - nonparametric missing value imputation for mixed-type data', Bioinformatics, 28(1) 2012, 112-118, doi: 10.1093/bioinformatics/btr597

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600,nb.cond=2);

#Imputation of missing values with Random Forest
dat.rf=impute.RF(tab=res.sim$dat.obs,conditions=res.sim$condition);
```

`impute.slsa` *Imputing missing values using an adaptation of the LSimpute algorithm (Bo et al. (2004)) to experimental designs. This algorithm is named "Structured Least Squares Algorithm" (SLSA).*

Description

This function is an adaptation of the LSimpute algorithm (Bo et al. (2004)) to experimental designs usually met in MS-based quantitative proteomics.

Usage

```
impute.slsa(tab, conditions, repbio=NULL, reptech=NULL, nknn=30, selec="all", weight="o",
ind.comp=1, progress.bar=TRUE)
```

Arguments

tab	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
conditions	A vector of factors indicating the biological condition to which each sample belongs.
repbio	A vector of factors indicating the biological replicate to which each sample belongs. Default is NULL (no experimental design is considered).
reptech	A vector of factors indicating the technical replicate to which each sample belongs. Default is NULL (no experimental design is considered).
nknn	The number of nearest neighbours used in the algorithm (see Details).
selec	A parameter to select a part of the dataset to find nearest neighbours between rows. This can be useful for big data sets (see Details).
weight	The way of weighting in the algorithm (see Details).
ind.comp	If ind.comp=1, only nearest neighbours without missing values are selected to fit linear models (see Details). Else, they can contain missing values.
progress.bar	If TRUE, a progress bar is displayed.

Details

This function imputes the missing values condition by condition. The rows of the input matrix are imputed when they have at least one observed value in the considered condition. For the rows having only missing values in a condition, you can use the impute.pa function.

For each row, a similarity measure between the observed values of this row and the ones of the other rows is computed. The similarity measure which is used is the absolute pairwise correlation coefficient if at least three side-by-side values are observed, and the inverse of the euclidean distance between side-by-side observed values in the other cases.

For big data sets, this step can be time consuming and that is why the input parameter selec allows to select random rows in the data set. If selec="all", then all the rows of the data set are considered; while if selec is a numeric value, for instance selec=100, then only 100 random rows are selected in the data set for computing similarity measures with each row containing missing values.

Once similarity measures are computed for a specific row, then the nknn rows with the highest similarity measures are considered to fit linear models and to predict several estimates for each missing value (see Bo et al. (2004)). If ind.comp=1, then only nearest neighbours without missing values in the condition are considered. However, unlike the original algorithm, our algorithm allows to consider the design of experiments that are specified in input through the vectors conditions, repbio and reptech. Note that conditions has to get a lower number of levels than repbio; and repbio has to get a lower number of levels than reptech.

In the original algorithm, several predictions of each missing value are done from the estimated linear models and, then, they are weighted in function of their similarity measure and summed (see Bo et al. (2004)). In our algorithm, one can use the original weighting function of Bo et al. (2004) if weight="o", i.e. (sim^2/(1-sim^2+1e-06))^2 where sim is the similarity measure; or the weighting function sim^weight if weight is a numeric value.

Value

The input matrix `tab` with imputed values instead of missing values.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

References

Bo, T. H., Dysvik, B., & Jonassen, I. (2004). LSimpute: accurate estimation of missing values in microarray data with least squares methods. Nucleic acids research, 32(3), e34.

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600);

#Imputation of missing values with the slsa algorithm
dat.slsa=impute.slsa(tab=res.sim$dat.obs,conditions=res.sim$condition,repbio=res.sim$repbio);
```

---

mi.mix *Multiple imputation from a matrix of probabilities of being MCAR for each missing value.*

Description

This function allows imputing data sets with a multiple imputation strategy. For details, see Giai Gianetto Q. et al. (2020) (doi: doi: 10.1101/2020.05.29.122770).

Usage

```
mi.mix(tab, tab.imp, prob.MCAR, conditions, repbio=NULL, reptech=NULL, nb.iter=3, nknn=15,
weight=1, selec="all", siz=500, ind.comp=1, methodMCAR="mle", q=0.95,
progress.bar=TRUE, details=FALSE, ncp.max=5, maxiter = 10, ntree = 100,
variablewise = FALSE, decreasing = FALSE, verbose = FALSE, mtry = floor(sqrt(ncol(tab))),
replace = TRUE,classwt = NULL, cutoff = NULL, strata = NULL, sampsize = NULL,
nodesize = NULL, maxnodes = NULL,xtrue = NA, parallelize = c('no', 'variables',
'forests'), methodMNAR="igcda",q.min = 0.025, q.norm = 3, eps = 0, distribution = "unif",
param1 = 3, param2 = 1, R.q.min=1)
```

Arguments

`tab`	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.

tab.imp	A matrix where the missing values of tab have been imputed under the assumption that they are all MCAR. For instance, such a matrix can be obtained from the function impute.slsa of this package.
prob.MCAR	A matrix of probabilities that each missing value is MCAR. For instance such a matrix can be obtained from the function prob.mcar.tab of this package.
conditions	A vector of factors indicating the biological condition to which each column (experimental sample) belongs.
repbio	A vector of factors indicating the biological replicate to which each column belongs. Default is NULL (no experimental design is considered).
reptech	A vector of factors indicating the technical replicate to which each column belongs. Default is NULL (no experimental design is considered).
nb.iter	The number of iterations used for the multiple imputation method.
nknn	The number of nearest neighbours used in the SLSA algorithm (see impute.slsa).
selec	A parameter to select a part of the dataset to find nearest neighbours between rows. This can be useful for big data sets (see impute.slsa).
siz	A parameter to select a part of the dataset to perform imputations with a MCAR-devoted algorithm. This can be useful for big data sets. Note that siz needs to be inferior to selec.
weight	The way of weighting in the algorithm (see impute.slsa).
ind.comp	If ind.comp=1, only nearest neighbours without missing values are selected to fit linear models (see impute.slsa). Else, they can contain missing values.
methodMCAR	The method used for imputing MCAR data. If methodi="mle" (default), then the impute.mle function is used (imputation using an EM algorithm). If methodi="pca", then the impute.PCA function is used (imputation using Principal Component Analysis). If methodi="rf", then the impute.RF function is used (imputation using Random Forest). Else, the impute.slsa function is used (imputation using Least Squares on nearest neighbours).
methodMNAR	The method used for imputing MNAR data. If methodMNAR="igcda" (default), then the impute.igcda function is used. Else, the impute.pa function is used.
q	A quantile value (see impute.igcda).
progress.bar	If TRUE, a progress bar is displayed.
details	If TRUE, the function gives a list of three values: imputed.matrix a matrix with the average of imputed values for each missing value, sd.imputed.matrix a matrix with the standard deviations of imputed values for each missing value, all.imputed.matrices an array with all the nb.iter matrices of imputed values that have been generated.
ncp.max	parameter of the impute.PCA function.
maxiter	parameter of the impute.RF function.
ntree	parameter of the impute.RF function.
variablewise	parameter of the impute.RF function.
decreasing	parameter of the impute.RF function.
verbose	parameter of the impute.RF function.

mtry	parameter of the impute.RF function.
replace	parameter of the impute.RF function.
classwt	parameter of the impute.RF function.
cutoff	parameter of the impute.RF function.
strata	parameter of the impute.RF function.
sampsize	parameter of the impute.RF function.
nodesize	parameter of the impute.RF function.
maxnodes	parameter of the impute.RF function.
xtrue	parameter of the impute.RF function.
parallelize	parameter of the impute.RF function.
q.min	parameter of the impute.pa function.
q.norm	parameter of the impute.pa function.
eps	parameter of the impute.pa function.
distribution	parameter of the impute.pa function.
param1	parameter of the impute.pa function.
param2	parameter of the impute.pa function.
R.q.min	parameter of the impute.pa function.

Details

At each iteration, a matrix indicating the MCAR values is generated by Bernouilli distributions having parameters given by the matrix prob.MCAR. The generated MCAR values are next imputed thanks to the matrix tab.imp. For each row containing MNAR values, the other rows are imputed thanks to the function impute.igcda and, next, the considered row is imputed thanks to one of the MCAR-devoted imputation methods (impute.mle, impute.RF, impute.PCA or impute.slsa). So, the function impute.igcda allows to deform the correlation structure of the dataset in view to be closer to that of the true values, while the MCAR-devoted imputation method will impute by taking into account this modified correlation structure.

Value

The input matrix tab with average imputed values instead of missing values if details=FALSE (default). If details=TRUE, a list of three values: imputed.matrix a matrix with the average of imputed values for each missing value, sd.imputed.matrix a matrix with the standard deviations of imputed values for each missing value, all.imputed.matrices an array with all the nb.iter matrices of imputed values that have been generated.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

References

Giai Gianetto, Q., Wieczorek S., Couté Y., Burger, T. (2020). A peptide-level multiple imputation strategy accounting for the different natures of missing values in proteomics data. bioRxiv 2020.05.29.122770; doi: doi: 10.1101/2020.05.29.122770

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=5000,nb.miss=1000);

#Fast imputation of missing values with the impute.rand algorithm
dat.rand=impute.rand(tab=res.sim$dat.obs,conditions=res.sim$condition);

#Estimation of the mixture model
res=estim.mix(tab=res.sim$dat.obs, tab.imp=dat.rand, conditions=res.sim$condition);

#Computing probabilities to be MCAR
born=estim.bound(tab=res.sim$dat.obs,conditions=res.sim$condition);
proba=prob.mcar.tab(tab.u=born$tab.upper,res=res);

#Multiple imputation strategy with 3 iterations (can be time consuming in function of the data set!)
data.mi=mi.mix(tab=res.sim$dat.obs, tab.imp=dat.rand, prob.MCAR=proba, conditions=
res.sim$conditions, repbio=res.sim$repbio, nb.iter=3);
```

---

miss.mcar.process *Estimating the MCAR mechanism in a sample.*

Description

This function allows estimating the MCAR data mechanism, i.e. the probability to be MCAR given that the value is missing in function of the intensity level, from an estimation of a mixture model of MNAR and MCAR values (see `estim.mix` function).

Usage

```
miss.mcar.process(abs,pi_mcar,F_tot,F_na)
```

Arguments

`abs`	The interval on which is estimated the MCAR data mechanism.
`pi_mcar`	An estimation of the proportion of MCAR values.
`F_tot`	An estimation of the cumulative distribution function of the complete values on the interval `abs`.
`F_na`	An estimation of the cumulative distribution function of the missing values on the interval `abs`.

Value

A list composed of:

`abs`	The interval on which is estimated the MCAR data mechanism.
`p`	The estimated probability to be MCAR given that the value is missing on the interval `abs`.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

See Also

estim.mix

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600,pi.mcar=0.2,para=0.5,nb.cond=2,nb.repbio=3,
nb.sample=5,m.c=25,sd.c=2,sd.rb=0.5,sd.r=0.2);

#Imputation of missing values with the slsa algorithm
dat.slsa=impute.slsa(tab=res.sim$dat.obs,conditions=res.sim$condition,repbio=res.sim$repbio);

#Estimation of the mixture model
res=estim.mix(tab=res.sim$dat.obs, tab.imp=dat.slsa, conditions=res.sim$condition);

#Estimating the MCAR mechanism in the first replicate
mcp=miss.mcar.process(res$abs.mod,res$pi.mcar[1],res$F.tot[,1],res$F.na[,1])
plot(mcp$abs,mcp$p,ty="l",xlab="Intensity values",ylab="Estimated probability to be MCAR")
```

---

miss.total.process *Estimating the missing data mechanism in a sample.*

Description

This function allows estimating the missing data mechanism, i.e. the probability to be missing in function of the intensity level, from an estimation of a mixture model of MNAR and MCAR values (see estim.mix function).

Usage

```
miss.total.process(abs,pi_na,F_na,F_tot)
```

Arguments

abs	The interval on which is estimated the missing data mechanism.
pi_na	The proportion of missing values.
F_na	An estimation of the cumulative distribution function of the missing values on the interval abs.
F_tot	An estimation of the cumulative distribution function of the complete values on the interval abs.

Value

A list composed of:

abs	The interval on which is estimated the missing data mechanism.
p	The estimated probability to be missing in function of the intensity level.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

See Also

estim.mix

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600);

#Imputation of missing values with the slsa algorithm
dat.slsa=impute.slsa(tab=res.sim$dat.obs,conditions=res.sim$condition,repbio=res.sim$repbio);

#Estimation of the mixture model
res=estim.mix(tab=res.sim$dat.obs, tab.imp=dat.slsa, conditions=res.sim$condition);

#Estimating the missing mechanism in the first replicate
mtp=miss.total.process(res$abs.mod,res$pi.na[1],res$F.na[,1],res$F.tot[,1])
plot(mtp$abs,mtp$p,ty="l",xlab="Intensity values",ylab="Estimated probability to be missing")
```

---

pi.mcar.karpievitch *Estimating the proportion of MCAR values in biological conditions using the method of Karpievitch (2009).*

Description

This function allows estimating the proportion of MCAR values in biological conditions using the method of Karpievitch (2009).

Usage

```
pi.mcar.karpievitch(tab,conditions)
```

Arguments

tab	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
conditions	A vector of factors indicating the biological condition to which each column (experimental sample) belongs.

Value

A list composed of:

pi.mcar	The proportion of MCAR values in each biological condition.
prop.na	The proportion of missing values for each peptide in each condition.
moy	The average of observed values for each peptide in each condition.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

References

Karpievitch, Y., Stanley, J., Taverner, T., Huang, J., Adkins, J. N., Ansong, C., ... & Smith, R. D. (2009). A statistical framework for protein quantitation in bottom-up MS-based proteomics. Bioinformatics, 25(16), 2028-2034.

See Also

estim.mix

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600);

#Proportion of MCAR values in each condition
pi.mcar.karpievitch(tab=res.sim$dat.obs,conditions=res.sim$conditions)
```

---

pi.mcar.logit — *Estimating the proportion of MCAR values in a sample using a logit model.*

Description

This function allows estimating the proportion of MCAR values in a sample using a logit model.

Usage

```
pi.mcar.logit(tab,conditions)
```

Arguments

tab	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
conditions	A vector of factors indicating the biological condition to which each column (experimental sample) belongs.

Value

A list composed of:

pi.mcar	The estimated proportion of MCAR values.
coef1	The estimated intercept of each logit model estimated in a sample.
coef2	The estimated coefficient of each logit model estimated in a sample.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

See Also

estim.mix

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600);

#Proportion of MCAR values in each sample
pi.mcar.logit(tab=res.sim$dat.obs,conditions=res.sim$conditions)
```

---

pi.mcar.probit *Estimating the proportion of MCAR values in a sample using a probit model.*

Description

This function allows estimating the proportion of MCAR values in a sample using a probit model.

Usage

```
pi.mcar.probit(tab,conditions)
```

Arguments

tab	A data matrix containing numeric and missing values. Each column of this matrix is assumed to correspond to an experimental sample, and each row to an identified peptide.
conditions	A vector of factors indicating the biological condition to which each column (experimental sample) belongs.

Value

A list composed of:

pi.mcar	The estimated proportion of MCAR values.
coef1	The estimated intercept of each probit model estimated in a sample.
coef2	The estimated coefficient of each probit model estimated in a sample.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

See Also

estim.mix

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600);

#Proportion of MCAR values in each sample
pi.mcar.probit(tab=res.sim$dat.obs, conditions=res.sim$condition);
```

---

prob.mcar *Estimation of a vector of probabilities that missing values are MCAR.*

Description

This function returns a vector of probabilities that each missing value is MCAR from specified confidence intervals.

Usage

```
prob.mcar(b.u,absc,pi.na,pi.mcar,F.tot,F.obs)
```

Arguments

b.u	A numeric vector of upper bounds for missing values.
absc	The interval on which is estimated the MCAR data mechanism.
pi.na	The estimated proportion of missing values.
pi.mcar	The estimated proportion of MCAR values among missing values.
F.tot	An estimation of the cumulative distribution function of the complete values on the interval absc.
F.obs	An estimation of the cumulative distribution function of the missing values on the interval absc.

Value

A numeric vector of estimated probabilities to be MCAR for missing values assuming upper bounds for them (b.u). The input arguments absc, pi.mcar, pi.na, F.tot and F.obs can be estimated thanks to the function estim.mix.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

See Also

estim.mix

Examples

```
#Simulating data
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600);

#Imputation of missing values with the slsa algorithm
dat.slsa=impute.slsa(tab=res.sim$dat.obs,conditions=res.sim$condition,repbio=res.sim$repbio);

#Estimation of the mixture model
res=estim.mix(tab=res.sim$dat.obs, tab.imp=dat.slsa, conditions=res.sim$condition);

#Computing probabilities to be MCAR
born=estim.bound(tab=res.sim$dat.obs,conditions=res.sim$condition);

#Computing probabilities to be MCAR in the first column of result$tab.mod
proba=prob.mcar(b.u=born$tab.upper[,1],absc=res$abs.mod,pi.na=res$pi.na[1],
pi.mcar=res$pi.mcar[1], F.tot=res$F.tot[,1], F.obs=res$F.obs[,1]);
```

---

prob.mcar.tab *Estimation of a matrix of probabilities that missing values are MCAR.*

Description

This function returns a matrix of probabilities that each missing value is MCAR from specified confidence intervals.

Usage

```
prob.mcar.tab(tab.u,res)
```

Arguments

tab.u	A numeric matrix of upper bounds for missing values.
res	An output list resulting from the function estim.mix.

Value

A numeric matrix of estimated probabilities to be MCAR for missing values assuming upper bounds for them (`tab.u`).

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

See Also

`estim.mix`

Examples

```
#Simulating data
res.sim=sim.data(nb.pept=2000,nb.miss=600,para=5);

#Imputation of missing values with a MCAR-devoted algorithm: here the slsa algorithm
dat.slsa=impute.slsa(tab=res.sim$dat.obs,conditions=res.sim$condition,repbio=res.sim$repbio);

#Estimation of the mixture model
res=estim.mix(tab=res.sim$dat.obs, tab.imp=dat.slsa, conditions=res.sim$condition);

#Computing probabilities to be MCAR
born=estim.bound(tab=res.sim$dat.obs,conditions=res.sim$condition);
proba=prob.mcar.tab(born$tab.upper,res);

#Histogram of probabilities to be MCAR associated to generated MCAR values
hist(proba[res.sim$list.MCAR[[1]],1],

freq=FALSE,main="Estimated probabilities to be MCAR for known MCAR values",xlab="",col=2);
```

`sim.data` *Simulation of data sets by controlling the proportion of MCAR values and the distribution of MNAR values.*

Description

This function simulates data sets similar to MS-based bottom-up proteomic data sets.

Usage

```
sim.data(nb.pept=15000,nb.miss=5000,pi.mcar=0.2,para=3,nb.cond=1,nb.repbio=3,
nb.sample=3,m.c=25,sd.c=2,sd.rb=0.5,sd.r=0.2)
```

Arguments

nb.pept	The number of rows (identified peptides) of the generated data set.
nb.miss	The number of missing values to generate in each column.
pi.mcar	The proportion of MCAR values in each column.
para	Parameter used for simulating MNAR values in columns (see Details).
nb.cond	The number of studied biological conditions.
nb.repbio	The number of biological samples in each condition.
nb.sample	The number of samples coming from each biological sample.
m.c	The mean of the average values in each condition.
sd.c	The standard deviation of the average values in each condition.
sd.rb	The standard deviation of the average values in each biological sample.
sd.r	The standard deviation of values in each row among the samples coming from a same biological sample.

Details

First, the average of intensities of a peptide i in a condition is generated by a Gaussian distribution $m_{cond} \sim N(m.c, sd.c)$. Second, the effect of a biological sample is generated by $m_{bio} \sim N(0, sd.rb)$. The value of a peptide i in the sample j belonging to a specific biological sample and a specific condition is finally generated by $x_{ij} \sim N(m_{cond} + m_{bio}, sd.r)$.

Next, the MCAR values are generated in each column by random draws without replacement among the indexes of rows. The MNAR values are generated in the remaining indexes of rows by random draws without replacement and by respecting the following probabilities:

$P(x_{ij} isMNAR) = 1 - (x_{ij} - min_i(x_{ij}))/((max_i(x_{ij}) - min_i(x_{ij})) * (para))$

where $para$ allows adjusting the distribution of MNAR values. If $para = 0$, then the MNAR values are uniformly distributed among intensity level. More $para$ is high and more the MNAR values arise for small intensity levels and not for high intensity levels.

Value

dat.obs	The simulated data set.
dat.comp	The simulated data set without missing values.
list.MCAR	The index of MCAR values among the rows in each column of the data set.
nMCAR	The number of MCAR values in each sample (after deleting rows with only generated missing values).
nNA	The number of missing values in each sample (after deleting rows with only generated missing values).
conditions	A vector of factors indicating the biological condition to which each sample belongs.
repbio	A vector of factors indicating the biological sample to which each sample belongs.

Author(s)

Quentin Giai Gianetto <quentin2g@yahoo.fr>

Examples

```
## The function can be used as
res.sim=sim.data(nb.pept=2000,nb.miss=600);
## Simulated data matrix
data=res.sim$dat.obs;
## Vector of conditions of membership for each sample
cond=res.sim$conditions;
## Vector of biological sample of membership for each sample
repbio=res.sim$repbio;
## Percentage of generated MCAR values for each sample
pi_mcar=res.sim$nMCAR/res.sim$nNA
```

---

translatedRandomBeta *Function to generated values following a translated Beta distribution*

Description

Function to generate values following a translated Beta distribution

Usage

```
translatedRandomBeta(n, min, max, param1 = 3, param2 = 1)
```

Arguments

n	Number of values to generate.
min	Minimum of the values to be generated.
max	Maximum of the values to be generated.
param1	Parameter of the Beta distribution.
param2	Parameter of the Beta distribution.

Value

A vector of values following a translated Beta distribution.

Index