Package 'FunChisq'

May 7, 2026

Type Package

Version 2.5.4

Date 2024-05-10

Title Model-Free Functional Chi-Squared and Exact Tests

Author Yang Zhang [aut],
Hua Zhong [aut] (ORCID: <https://orcid.org/0000-0003-1962-2603>),
Hien Nguyen [aut] (ORCID: <https://orcid.org/0000-0002-7237-4752>),
Ruby Sharma [aut] (ORCID: <https://orcid.org/0000-0001-7774-4065>),
Sajal Kumar [aut] (ORCID: <https://orcid.org/0000-0003-0930-1582>),
Yiyi Li [aut] (ORCID: <https://orcid.org/0000-0001-8859-3987>),
Joe Song [aut, cre] (ORCID: <https://orcid.org/0000-0002-6883-6547>)

Maintainer Joe Song <joemsong@cs.nmsu.edu>

Description Statistical hypothesis testing methods for
inferring model-free functional dependency using asymptotic
chi-squared or exact distributions. Functional test
statistics are asymmetric and functionally optimal, unique
from other related statistics. Tests in this package reveal
evidence for causality based on the causality-by-
functionality principle. They include asymptotic functional
chi-squared tests (Zhang & Song 2013) <doi:10.48550/arXiv.1311.2707>,
an adapted functional chi-squared test (Kumar & Song 2022)
<doi:10.1093/bioinformatics/btac206>,
and an exact functional test (Zhong & Song 2019)
<doi:10.1109/TCBB.2018.2809743> (Nguyen et al. 2020)
<doi:10.24963/ijcai.2020/372>. The normalized functional
chi-squared test was used by Best Performer 'NMSUSongLab'
in HPN-DREAM (DREAM8) Breast Cancer Network Inference
Challenges (Hill et al. 2016) <doi:10.1038/nmeth.3773>. A
function index (Zhong & Song 2019)
<doi:10.1186/s12920-019-0565-9> (Kumar et al. 2018)
<doi:10.1109/BIBM.2018.8621502> derived from the
functional test statistic offers a new effect size measure
for the strength of functional dependency, a better
alternative to conditional entropy in many aspects. For

continuous data, these tests offer an advantage over regression analysis when a parametric functional form cannot be assumed; for categorical data, they provide a novel means to assess directional dependency not possible with symmetrical Pearson's chi-squared or Fisher's exact tests.

License LGPL (>= 3)

Encoding UTF-8

Depends R (>= 3.0.0)

Imports Rcpp, Rdpack (>= 0.6-1), stats, dqrng

LinkingTo BH, Rcpp

RdMacros Rdpack

Suggests Ckmeans.1d.dp, DescTools, DiffXTables, GridOnClusters, infotheo, knitr, rmarkdown, testthat (>= 3.0.0)

NeedsCompilation yes

URL https://www.cs.nmsu.edu/~joemsong/publications/

Config/testthat/edition 3

VignetteBuilder knitr

Repository CRAN

Date/Publication 2024-05-10 19:03:04 UTC

Contents

FunChisq-package . . . 3
add.noise . . . 5
cond.fun.chisq.test . . . 7
cp.fun.chisq.test . . . 9
Exact Functional Test . . . 11
fun.chisq.test . . . 12
FunChisq-deprecated . . . 16
plot_table . . . 16
simulate_tables . . . 17
test.interactions . . . 23

Index **26**

FunChisq-package *Model-Free Functional Chi-Squared and Exact Tests*

Description

Statistical hypothesis testing methods for model-free functional dependency using asymptotic chi-squared or exact distributions. Functional chi-squared test statistics (Zhang and Song 2013; Zhang 2014; Nguyen 2018; Zhong 2019; Zhong and Song 2019a; Nguyen et al. 2020) are asymmetric, functionally optimal, and model-free, unique from other related statistical measures.

Tests in this package reveal evidence for causality based on the causality-by-functionality principle (Simon and Rescher 1966). The tests require data from two or more variables be formatted as a contingency table. Continuous variables need to be discretized first, for example, using R packages **Ckmeans.1d.dp** or **GridOnClusters**.

The package implements an asymptotic functional chi-squared test (Zhang and Song 2013; Zhang 2014), an adapted functional chi-squared test (Kumar2022AFT), and an exact functional test (Nguyen 2018; Zhong 2019; Zhong and Song 2019a; Nguyen et al. 2020). The normalized functional chi-squared test was used by Best Performer NMSUSongLab in HPN-DREAM (DREAM8) Breast Cancer Network Inference Challenges (Hill et al. 2016).

A function index derived from the functional chi-squared offers a new effect size measure for the strength of function dependency. It is asymmetrically functionally optimal, different from the symmetric Cramer's V, also a better alternative to conditional entropy in many aspects.

A simulator is provided to generate functional, dependent non-functional, and independent patterns (Sharma et al. 2017).

For continuous data, these tests offer an advantage over regression analysis when a parametric form cannot be reliably assumed for the underlying function. For categorical data, they provide a novel means to assess directional dependency not possible with symmetrical Pearson's chi-squared test, G-test, or Fisher's exact test.

Details

Package:	FunChisq
Type:	Package
Current version:	2.5.3
Initial release version:	1.0
Initial release date:	2014-03-08
License:	LGPL (>= 3)

Author(s)

Yang Zhang, Hua Zhong, Hien Nguyen, Ruby Sharma, Sajal Kumar, Yiyi Li, and Joe Song

References

Hill SM, Heiser LM, Cokelaer T, Unger M, Nesser NK, Carlin DE, Zhang Y, Sokolov A, Paull EO, Wong CK, Graim K, Bivol A, Wang H, Zhu F, Afsari B, Danilova LV, Favorov AV, Lee WS, Taylor D, Hu CW, Long BL, Noren DP, Bisberg AJ, The HPN-DREAM Consortium, Mills GB, Gray JW, Kellen M, Norman T, Friend S, Qutub AA, Fertig EJ, Guan Y, Song M, Stuart JM, Spellman PT, Koeppl H, Stolovitzky G, Saez-Rodriguez J, Mukherjee S (2016). "Inferring causal molecular networks: empirical assessment through a community-based effort." *Nat Methods*, **13**, 310–318. doi:10.1038/nmeth.3773.

Nguyen HH (2018). *Inference of Functional Dependency via Asymmetric, Optimal, and Model-free Statistics*. Ph.D. thesis, Department of Computer Science, New Mexico State University, Las Cruces, NM, USA.

Nguyen HH, Zhong H, Song M (2020). "Optimality, accuracy, and efficiency of an exact functional test." In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 2683–2689. doi:10.24963/ijcai.2020/372.

Sharma R, Kumar S, Zhong H, Song M (2017). "Simulating noisy, nonparametric, and multivariate discrete patterns." *The R Journal*, **9**(2), 366–377. doi:10.32614/RJ2017053.

Simon HA, Rescher N (1966). "Cause and counterfactual." *Philosophy of Science*, **33**(4), 323–340.

Zhang Y (2014). *Nonparametric Statistical Methods for Biological Network Inference*. Ph.D. thesis, Department of Computer Science, New Mexico State University, Las Cruces, NM, USA.

Zhang Y, Song M (2013). "Deciphering interactions in causal networks without parametric assumptions." *arXiv Molecular Networks*, arXiv:1311.2707. https://arxiv.org/abs/1311.2707.

Zhong H (2019). *Model-free Gene-to-zone Network Inference of Molecular Mechanisms in Biology*. Ph.D. thesis, Department of Computer Science, New Mexico State University, Las Cruces, NM, USA.

Zhong H, Song M (2019a). "A fast exact functional test for directional association and cancer biology applications." *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **16**(3), 818–826. doi:10.1109/TCBB.2018.2809743.

See Also

For data discretization, an option is optimal univariate clustering via package **Ckmeans.1d.dp**. A second option is joint multivariate discretization via package **GridOnClusters**.

For symmetric dependency tests on discrete data, see Pearson's chi-squared test (`chisq.test`), Fisher's exact test (`fisher.test`), mutual information (package **entropy**), and G-test, implemented in packages **DescTools** and **RVAideMemoire**.

`add.noise` *Apply Noise to Discrete-Valued Tables*

Description

The function can apply two types of noise to contingency tables of discrete values. A house noise model is designed for ordinal variables; a candle noise model is for categorical variables. Noise is applied independently for each data point in a table.

Usage

```
add.noise(tables, u, noise.model, margin=0)
add.house.noise(tables, u, margin=0)
add.candle.noise(tables, u, margin=0)
```

Arguments

`tables`	a list of tables or one table. A table can be either a matrix or a data frame of integer values.
`u`	a numeric value between 0 and 1 to specify the noise level to be applied to the input tables. See Details.
`noise.model`	a character string indicating the noise model of either `"house"` for ordinal variables or `"candle"` for categorical variables. See Details.
`margin`	a value of either 0, 1, or 2. Default is 0. 0: noise is applied along both rows and columns in a table. The sum of values in the table is the same before and after noise application. 1: noise is applied along each row. The sum of each row is the same before and after noise application. 2: noise is applied along each column. The sum of each column is the same before and after noise application.

Details

Each noise model defines a conditional probability function of a noisy version given an original discrete value and a noise level. In the house noise model for ordinal variables, defined in (Zhang et al. 2015), the probability decreases as the noisy version deviates from the original ordinal value. The shape of the function is like a pitched house roof. In the candle noise model for categorical variables, the probability of the noisy version for any value other than the original categorical value is the same given the noise level. The function shape is like a candle.

At a minimum level of 0, no noise is applied on the input table(s). A maximum level of 1 indicates that the original sample will be changed to some other values with a probability of 1. For a discrete random variable of two possible values, a noise level of 1 will flip the values and create a non-random pattern; a noise level of 0.5 creates the most random pattern.

Value

If `tables` is a list, the function returns a list of tables with noised applied. If `tables` is a numeric matrix or a data frame, the function returns one table with noise applied.

Author(s)

Hua Zhong, Yang Zhang, and Joe Song.

References

Zhang Y, Liu ZL, Song M (2015). "ChiNet uncovers rewired transcription subnetworks in tolerant yeast for advanced biofuels conversion." *Nucleic Acids Research*, **43**(9), 4393–4407. doi:10.1093/nar/gkv358.

See Also

`simulate_tables`.

Examples

```
# Example 1. Add house noise to a single table

# Create a 4x4 table
t <- matrix(c(3,0,0,0,
              0,2,2,0,
              0,0,0,4,
              3,3,2,0),
            nrow=4, ncol=4, byrow=TRUE)
# Two ways to apply house noise at level 0.1 along both rows
# and columns of the table:
add.noise(t, 0.1, "house", 0)
add.house.noise(t, 0.1, 0)


# Example 2. Add candle noise to a list of tables

# Create a list of tables
t.list <- list(t+5, t*10, t*2)
# Two ways to apply candle noise at level 0.2 along the rows
# of the table:
add.noise(t.list, 0.2, "candle", 1)
add.candle.noise(t.list, 0.2, 1)
```

`cond.fun.chisq.test` *Conditional Functional Chi-Squared Test*

Description

Asymptotic chi-squared test to determine the model-free functional dependency of effect variable Y on a cause variable X, conditioned on a third variable Z.

Usage

```
cond.fun.chisq.test(x, y, z=NULL, data=NULL, log.p = FALSE,
                    method = c("fchisq", "nfchisq"))
```

Arguments

`x`	vector or character; either a discrete random variable (cause) represented as vector or a character column name in `data`.
`y`	vector or character; either a discrete random variable (effect) represented as vector or a character column name in `data`.
`z`	vector or character; either a discrete random variable (condition) represented as vector or a character column name in `data`. In case of `NULL`, a `fun.chisq.test` on a contingency table, with `x` as row variable and `y` as column variable, is returned. See `fun.chisq.test` for details. The default is `NULL`.
`data`	a data frame containing three or more columns whose names can be used as values for `x`, `y` and `z`. In case of `NULL`, `x`, `y` and `z` must be vectors. The default is `NULL`.
`log.p`	logical; if `TRUE`, the p-value is given as `log(p)`. Taking the log improves numerical precision when the p-value is close to zero. The default is `FALSE`.
`method`	a character string to specify the method to compute the conditional functional chi-squared test statistic and its p-value. The options are `"fchisq"` (default) and `"nfchisq"`. See Details.

Details

The conditional functional chi-squared test introduces the concept of conditional functional depedency, where the functional association between two variables (`x` and `y`) is tested conditioned on a third variable (`z`) (Zhang 2014). Two methods are provided to compute the chi-squared statistic and its p-value. When `method = "fchisq"`, the p-value is computed using the chi-squared distribution; when `method = "nfchisq"`, a normalized statistic is obtained by shifting and scaling the original chi-squared statistic and a p-value is computed using the standard normal distribution (Box et al. 2005). The normalized test is more conservative on the degrees of freedom.

Value

A list with class "htest" containing the following components:

statistic	the conditional functional chi-squared statistic if method = "fchisq"; or the normalized conditional functional chi-squared statistic if method = "nfchisq".
parameter	degrees of freedom for the conditional functional chi-squared statistic.
p.value	p-value of the conditional functional test. If method = "fchisq", the p-value is computed by an asymptotic chi-squared distribution; if method = "nfchisq", the p-value is computed by the standard normal distribution.
estimate	an estimate of the conditional function index between 0 and 1. The value of 1 indicates strong functional dependency between x and y, given z. It is asymmetrical with respect to whether x was chosen as the cause of effect y or vice versa.

Author(s)

Sajal Kumar and Mingzhou Song

References

Box GE, Hunter JS, Hunter WG (2005). *Statistics for Experimenters: Design, Innovation and Discovery*, 2nd edition. Wiley-Interscience, New York.

Zhang Y (2014). *Nonparametric Statistical Methods for Biological Network Inference*. Ph.D. thesis, Department of Computer Science, New Mexico State University, Las Cruces, NM, USA.

See Also

See (unconditional) functional chi-squared test fun.chisq.test.

Examples

```
# Generate a relationship between variables X and Z
xz = matrix(c(30,2,2, 2,2,40, 2,30,2),ncol=3,nrow=3,
            byrow = TRUE)
# Re-construct X
x = rep(c(1:nrow(xz)),rowSums(xz))
# Re-construct Z
z = c()
for(i in 1:nrow(xz))
  z = c(z,rep(c(1:ncol(xz)),xz[i,]))

# Generate a relationship between variables Z and Y
# Make sure Z retains its distribution
zy = matrix(c(4,30, 30,4, 4,40),ncol=2,nrow=3,
            byrow = TRUE)
# Re-construct Y
y = rep(0,length(z))
for(i in unique(z))
  y[z==i] = rep(c(1:ncol(zy)),zy[i,])
```

```
# Tables
table(x,z)
table(z,y)
table(x,y)

# Conditional functional dependency
# Y = f(X) | Z should be false
cond.fun.chisq.test(x=x,y=y,z=z)
# Z = f(X) | Y should be true
cond.fun.chisq.test(x=x,y=z,z=y)
# Y = f(Z) | X should be true
cond.fun.chisq.test(x=z,y=y,z=x)
```

`cp.fun.chisq.test` *Comparative Chi-Squared Test for Model-Free Functional Heterogeneity*

Description

Comparative functional chi-squared tests on two or more contingency tables.

Usage

```
cp.fun.chisq.test(
  x, method = c("fchisq", "nfchisq", "default", "normalized"),
  log.p = FALSE
)
```

Arguments

`x`	a list of at least two matrices representing contingency tables of the same dimensionality.
`method`	a character string to specify the method to compute the functional chi-squared statistic and its p-value. The default is `"fchisq"` (equivalent to `"default"`). See Details. Note: `"default"` and `"normalized"` are deprecated.
`log.p`	logical; if `TRUE`, the p-value is given as `log(p)`. Taking the log improves the accuracy when p-value is close to zero. The default is `FALSE`.

Details

The comparative functional chi-squared test determines whether the patterns underlying the contingency tables are heterogeneous in a functional way (Zhang 2014). Specifically, it evaluates whether the column variable is a changed function of the row variable across the contingency tables.

Two methods are provided to compute the functional chi-squared statistic and its p-value. When `method = "fchisq"` (or `"default"`), the p-value is computed using the chi-squared distribution;

when method = "nfchisq" (or "normalized") a normalized statistic is obtained by shifting and scaling the original statistic and a p-value is computed using the standard normal distribution (Box et al. 2005) (Box et al., 2005). The normalized test is more conservative on the degrees of freedom.

Value

A list with class "htest" containing the following components:

statistic	functional heterogeneity statistic if method = "fchisq" (equivalent to "default"), or normalized statistic if method = "nfchisq" (equivalent to "normalized").
parameter	degrees of freedom.
p.value	p-value of the comparative functional chi-squared test. By default, it is computed by the chi-squared distribution. If method = "normalized", it is the p-value of the normalized statistic computed by the standard normal distribution.

Author(s)

Yang Zhang and Joe Song

References

Box GE, Hunter JS, Hunter WG (2005). *Statistics for Experimenters: Design, Innovation and Discovery*, 2nd edition. Wiley-Interscience, New York.

Zhang Y (2014). *Nonparametric Statistical Methods for Biological Network Inference*. Ph.D. thesis, Department of Computer Science, New Mexico State University, Las Cruces, NM, USA.

See Also

For comparative chi-squared test that does not consider functional dependencies, cp.chisq.test.

Examples

```
x <- matrix(c(4,0,4,0,4,0,1,0,1), 3)
y <- t(x)
z <- matrix(c(1,0,1,4,0,4,0,4,0), 3)
data <- list(x,y,z)
cp.fun.chisq.test(data)
cp.fun.chisq.test(data, method="nfchisq")
```

`Exact Functional Test` *Exact Functional Test on Two Discrete Random Variables*

Description

Perform the exact functional test on a contingency table to determine if the column variable is a function of the row variable. The null population includes tables with fixed row and column sums as in the observed table. The null distribution follows an exact multivariate hypergeometric distribution.

Usage

```
EFTDP(nm)
EFTDQP(nm)
```

Arguments

`nm`	a matrix of nonnegative integers representing a contingency table.

Details

The exact functional test is performed using branch-and-bound with two algorithms (DP and DQP) to avoid re-calculation of bounds (Nguyen 2018; Nguyen et al. 2020).

Value

The exact p-value of the test.

Note

The functions provide a direct entry into the C++ implementations of the exact functional test (Nguyen 2018; Nguyen et al. 2020).

Author(s)

Hien Nguyen, Hua Zhong, Yiyi Li, and Joe Song

References

Nguyen HH (2018). *Inference of Functional Dependency via Asymmetric, Optimal, and Model-free Statistics*. Ph.D. thesis, Department of Computer Science, New Mexico State University, Las Cruces, NM, USA.

Nguyen HH, Zhong H, Song M (2020). "Optimality, accuracy, and efficiency of an exact functional test." In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 2683–2689. doi:10.24963/ijcai.2020/372.

See Also

fun.chisq.test

Examples

```
x = matrix(c(0, 6, 3, 0, 10, 5, 4, 4, 1), nrow=3)
EFTDQP(x)
EFTDQP(t(x))

EFTDP(x)
EFTDP(t(x))
```

`fun.chisq.test` *Model-Free Functional Chi-Squared and Exact Tests*

Description

Asymptotic chi-squared, normalized chi-squared or exact tests on contingency tables to determine model-free functional dependency of the column variable on the row variable.

Usage

```
fun.chisq.test(
  x,
  method = c("fchisq", "nfchisq", "adapted",
             "exact", "exact.qp", "exact.dp", "exact.dqp",
             "default", "normalized", "simulate.p.value"),
  alternative = c("non-constant", "all"), log.p=FALSE,
  index.kind = c("conditional", "unconditional"),
  simulate.nruns = 2000,
  exact.mode.bound=TRUE
)
```

Arguments

`x`	a matrix representing a contingency table. The row variable represents the independent variable or all unique combinations of multiple independent variables. The column variable is the dependent variable.
`method`	a character string to specify the method to compute the functional chi-squared test statistic and its p-value. The options are `"fchisq"` (equivalent to `"default"`, the default), `"nfchisq"` (equivalent to `"normalized"`), `"exact"`, `"adapted"`, `"exact.qp"`, `"exact.dp"`, `"exact.dqp"` or `"simulate.p.value"`. See Details. Note: `"default"` and `"normalized"` are deprecated.
`alternative`	a character string to specify the alternative hypothesis. The options are `"non-constant"` (default, non-constant functions) and `"all"` (all types of functions including constant ones).

log.p	logical; if TRUE, the p-value is given as log(p). Taking the log improves the accuracy when p-value is close to zero. The default is FALSE.
index.kind	a character string to specify the kind of function index xi.f to be estimated. The options are "conditional" (default) and "unconditional". See Details.
simulate.nruns	A number to specify the number of tables generated to simulate the null distribution. Default is 2000. Only used when method="simulate.p.value".
exact.mode.bound	logical; if TRUE, a fast branch-and-bound algorithm is used for the exact functional test (method="exact"). If FALSE, a slow brute-force enumeration method is used to provide a reference for runtime analysis. Both options provide the same exact p-value. The default is TRUE.

Details

The functional chi-squared test determines whether the column variable is a function of the row variable in contingency table x (Zhang and Song 2013; Zhang 2014). This function supports three hypothesis testing methods:

When method="fchisq" (equivalent to "default", the default), the test statistic is computed as described in (Zhang and Song 2013; Zhang 2014) and the p-value is computed using the chi-squared distribution.

When method="nfchisq" (equivalent to "normalized"), the test statistic is obtained by shifting and scaling the original test statistic (Zhang and Song 2013; Zhang 2014); and the p-value is computed using the standard normal distribution (Box et al. 2005). The normalized chi-squared, more conservative on the degrees of freedom, was used by the Best Performer NMSUSongLab in HPN-DREAM (DREAM8) Breast Cancer Network Inference Challenges.

When method="exact", "exact.qp" (quadratic programming) (Zhong and Song 2019a; Zhong 2019), "exact.dp" (dynamic programming) (Nguyen 2018; Nguyen et al. 2020), or "exact.dqp" (dynamic and quadratic programming) (Nguyen 2018; Nguyen et al. 2020), an exact functional test is performed. The option of "exact" uses "exact.dqp", the fastest method. All methods compute an exact p-value.

When method="adapted", the adapted functional chi-squared test (Kumar and Song 2022) is used. The test statistic is obtained by evaluating the most populous portrait or square (number of rows <= number of columns) table in the contingency table x. The p-value is computed using the chi-squared distribution. This option should be used to determine the functional direction between variables in x.

For the "exact.qp" and "exact.dp" options, if the sample size is no more than 200 or the average cell count is less than five, and the table size is no more than 10 in either row or column, the exact test will not be called and the asymptotic functional chi-squared test (method="fchisq") is used instead.

For "exact.dqp", the exact functional test will always be performed.

For 2-by-2 contingency tables, the asymptotic test options (method="fchisq" or "nfchisq") are recommended to test functional dependency, instead of the exact functional test.

When method="simulate.p.value", a simulated null distribution is used to calculate p-value. The null distribution is a multinomial distribution that is the product of two marginal distributions. Like other Monte Carlo based methods, this method is slower but may be more accurate than other methods based on asymptotic distributions.

index.kind specifies the kind of function index to be computed. If the experimental design controls neither the row nor column marginal sums, index.kind = "unconditional" is recommended; If the column marginal sums are controlled, index.kind = "conditional" is recommended. The conditional function index is the square root of Goodman-Kruskal's tau (Goodman and Kruskal 1954). The choice of index.kind affects only the function index xi.f value, but not the test statistic or p-value.

Value

A list with class "htest" containing the following components:

statistic	the functional chi-squared statistic if method = "fchisq", "default", or "exact"; or the normalized functional chi-squared statistic if method = "nfchisq" or "normalized".
parameter	degrees of freedom for the functional chi-squared statistic.
p.value	p-value of the functional test. If method = "fchisq" (or "default"), it is computed by an asymptotic chi-squared distribution; if method = "nfchisq" (or "normalized"), it is computed by the standard normal distribution; if method = "exact", it is computed by an exact hypergeometric distribution.
estimate	an estimate of function index between 0 and 1. The value of 1 indicates a strictly mathematical function. It is asymmetrical with respect to transpose of the input contingency table, different from the symmetrical Cramer's V based on the Pearson's chi-squared test statistic. See (Zhong and Song 2019b; Kumar et al. 2018) for the definition of function index.

Author(s)

Yang Zhang, Hua Zhong, Hien Nguyen, Sajal Kumar, and Joe Song

References

Box GE, Hunter JS, Hunter WG (2005). *Statistics for Experimenters: Design, Innovation and Discovery*, 2nd edition. Wiley-Interscience, New York.

Goodman LA, Kruskal WH (1954). "Measures of Association for Cross Classifications." *Journal of the American Statistical Association*, **49**(268), 732–764.

Kumar S, Song M (2022). "Overcoming biases in causal inference of molecular interactions." *Bioinformatics*, **38**(10), 2818–2825. doi:10.1093/bioinformatics/btac206.

Kumar S, Zhong H, Sharma R, Li Y, Song M (2018). "Scrutinizing functional interaction networks from RNA-binding proteins to their targets in cancer." In *IEEE International Conference on Bioinformatics and Biomedicine*, 185–190. doi:10.1109/BIBM.2018.8621502.

Nguyen HH (2018). *Inference of Functional Dependency via Asymmetric, Optimal, and Model-free Statistics*. Ph.D. thesis, Department of Computer Science, New Mexico State University, Las Cruces, NM, USA.

Nguyen HH, Zhong H, Song M (2020). "Optimality, accuracy, and efficiency of an exact functional test." In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence,*

IJCAI-20, 2683–2689. doi:10.24963/ijcai.2020/372.

Zhang Y (2014). *Nonparametric Statistical Methods for Biological Network Inference*. Ph.D. thesis, Department of Computer Science, New Mexico State University, Las Cruces, NM, USA.

Zhang Y, Song M (2013). "Deciphering interactions in causal networks without parametric assumptions." *arXiv Molecular Networks*, arXiv:1311.2707. https://arxiv.org/abs/1311.2707.

Zhong H (2019). *Model-free Gene-to-zone Network Inference of Molecular Mechanisms in Biology*. Ph.D. thesis, Department of Computer Science, New Mexico State University, Las Cruces, NM, USA.

Zhong H, Song M (2019a). "A fast exact functional test for directional association and cancer biology applications." *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **16**(3), 818–826. doi:10.1109/TCBB.2018.2809743.

Zhong H, Song M (2019b). "Directional association test reveals high-quality putative cancer driver biomarkers including noncoding RNAs." *BMC Med Genomics*, **12**(7), 129. doi:10.1186/s12920-01905659.

See Also

For data discretization, an option is optimal univariate clustering via package **Ckmeans.1d.dp**. A second option is joint multivariate discretization via package **GridOnClusters**.

For symmetrical dependency tests on discrete data, see Pearson's chi-squared test `chisq.test`, Fisher's exact test `fisher.test`, and mutual information methods in package **entropy**.

Examples

```
# Example 1. Asymptotic functional chi-squared test
x <- matrix(c(20,0,20,0,20,0,5,0,5), 3)
fun.chisq.test(x) # strong functional dependency
fun.chisq.test(t(x)) # weak functional dependency

# Example 2. Normalized functional chi-squared test
x <- matrix(c(8,0,8,0,8,0,2,0,2), 3)
fun.chisq.test(x, method="nfchisq") # strong functional dependency
fun.chisq.test(t(x), method="nfchisq") # weak functional dependency

# Example 3. Exact functional chi-squared test
x <- matrix(c(4,0,4,0,4,0,1,0,1), 3)
fun.chisq.test(x, method="exact") # strong functional dependency
fun.chisq.test(t(x), method="exact") # weak functional dependency

# Example 4. Exact functional chi-squared test on a real data set
#            (Shen et al., 2002)
# x is a contingency table with row variable for p53 mutation and
#   column variable for CIMP
x <- matrix(c(12,26,18,0,8,12), nrow=2, ncol=3, byrow=TRUE)
```

```
# Example 5. Adpated functional chi-squared test
x <- matrix(c(20, 0, 1, 0, 1, 20, 3, 2, 15, 2, 5, 2), 3, 4, byrow=TRUE)
fun.chisq.test(x, method="adapted") # strong functional dependency
fun.chisq.test(t(x), method="adapted") # weak functional dependency

# Test the functional dependency: p53 mutation -> CIMP
fun.chisq.test(x, method="exact")

# Test the functional dependency CIMP -> p53 mutation
fun.chisq.test(t(x), method="exact")

# Example 6. Asymptotic functional chi-squared test with simulated distribution
x <- matrix(c(20,0,20,0,20,0,5,0,5), 3)
fun.chisq.test(x, method="simulate.p.value")
fun.chisq.test(x, method="simulate.p.value", simulate.n = 1000)
```

`FunChisq-deprecated` *Deprecated Functions in Package* **FunChisq**

Description

These functions are provided for compatibility with older versions of package **FunChisq** only, and may be removed eventually.

Details

The following functions are deprecated and will be made defunct; use the replacement indicated below:

- `cp.chisq.test`: now available as `cp.chisq.test` in package **DiffXTables**

`plot_table` *Plot a Table Using Color Intensity for Counts*

Description

A table is visualized as a matrix whose cells are shown with intensity of a given color proportional to the count in each cell. The count in a cell must be real: negative numbers or non-integers are acceptable. It provides a global understanding of the underlying pattern.

Usage

```
plot_table(table, xlab = "Column", ylab = "Row", col = "green3",
           xaxt = "n", yaxt = "n", main = NULL,
           show.value = TRUE, value.cex = 2,
           highlight=c("row.maxima", "none"),
           highlight.col=col,
           mgp=c(0.5,0,0), mar=c(2,2,3,1.5), ...)
```

Arguments

table	A data frame or a matrix.
xlab	The lable of the horizontal axis.
ylab	The lable of the vertical axis.
col	The color corresponding to the maximum value in the table.
xaxt	The style of the horizontal axis. See par.
yaxt	The style of the vertical axis. See par.
main	The title of the plot.
show.value	logical. Show the value of each cell in the table on the plot.
value.cex	Relative magnification factor if values are to be put in the cell.
...	Parameters acceptable to image function in the **graphics** package.
highlight	Specify to highlight row maxima or no highlight. When highlighted, a box is placed around each row maximum.
highlight.col	The color used to highlight a cell in the table.
mgp	The margin (in mex units) for the axis title, labels and line. See par.
mar	The margins of the four sides of the plot. See par.

Author(s)

Joe Song

Examples

```
opar <- par(mfrow=c(2,2))
plot_table(matrix(1:6, nrow=2), col="seagreen2")

plot_table(matrix(rnorm(20), nrow=5), col="orange", show.value=FALSE)

plot_table(matrix(rpois(16, 2), nrow=4), col="cornflowerblue", highlight="none")

plot_table(matrix(rbinom(15, 8, 0.5), nrow=3), col="sienna2", highlight="none")
par(opar)
```

---

simulate_tables *Simulate Noisy Contingency Tables to Represent Diverse Discrete Patterns*

Description

Generate random contingency tables representing various functional, non-functional, dependent, or independent patterns, without specifying a parametric model for the patterns.

Usage

```
simulate_tables(
  n = 100, nrow = 3, ncol = 3,
  type = c("functional", "many.to.one",
           "discontinuous", "independent",
           "dependent.non.functional"),
  n.tables = 1,
  row.marginal = NULL,
  col.marginal = NULL,
  noise = 0.0, noise.model = c("house", "candle"),
  margin = 0
)
```

Arguments

n	a positive integer specifying the sample size to be distributed in each table. For "functional", "many.to.one", and "discontinuous" tables, n must be no less than nrow. For "dependent.non.functional" tables, n must be no less than nrow*ncol. For "independent" tables, n must be a positive integer.
nrow	a positive integer specifying the number of rows in each table. The value must be no less than 2. For "many.to.one" tables, nrow must be no less than 3.
ncol	a positive integer specifying the number of columns in output table. ncol must be no less than 2.
type	a character string to specify the type of pattern underlying the table. The options are "functional" (default), "many.to.one", "discontinuous", "independent", and "dependent.non.functional". See Details.
n.tables	a positive integer value specifying the number of tables to be generated.
row.marginal	a non-negative numeric vector of length nrow specifying row marginal probabilities. The vector is linearly scaled so that the sum is 1. The default is a uniform distribution.
col.marginal	a non-negative numeric vector of length ncol specifying column marginal probabilities. The vector is linearly scaled so that the sum is 1. This argument is ignored by "dependent.non.functional" tables.
noise	a numeric value between 0 and 1 specifying the noise level to be added to a table using function add.noise. The noise can be applied along row, column, or both, which can be specified by the margin argument. See add.noise for details.
noise.model	a character string indicating the noise model of either "house" for ordinal variables (Zhang et al. 2015) or "candle" for categorical variables. See add.noise for details.
margin	a numeric value of either 0, 1 or 2. Default is 0. 0: noise is applied along both rows and columns. 1: noise is applied along each row. 2: noise is applied along each column. See add.noise for details.

Details

This function generates five types of table representing different interaction patterns between row and column discrete random variables X and Y. Three of the five types are non-constant functional patterns (Y is a non-constant function of X):

`type="functional"`: Y is a function of X but X may or may not be a function of Y.

`type="many.to.one"`: Y is a many-to-one function of X but X is not a function of Y.

`type="discontinuous"`: Y is a function of X, where the function value of X must differ from its neighbors. X may or may not be a function of Y. A discontinuous function forms a contrast with those that are close to constant functions.

The fourth `type` `"dependent.non.functional"` is non-functional patterns where X and Y are statistically dependent but not function of each other. The samples are distributed according to `row.marginal` probabilities.

The fifth `type` `"independent"` represents patterns where X and Y are statistically independent whose joint probability mass function is the product of their marginal probability mass functions.

For all functional tables (`type="functional"`, `type="many.to.one"`, `type="discontinuous"`), the samples are distributed using either the given row or column marginal probabilities. Theoretically, it is not always possible to enforce both marginals in a functional pattern. If both marginals are provided, one will be randomly selected to generate a table; about half of the time each equested marginal is used. If neither is provided, either row or column uniform marginal will be randomly selected to generate a table; half of the time a table will have a uniform row marginal and the other half a uniform column marginal.

Random noise can be optionally applied to the tables using either the house or the candle noise model. See `add.noise` for details.

Sharma et al. (2017) provide full mathematical and statistical details of the simulation strategies for the above table types except the `"discontinuous"` type which was introduced after the publication.

Value

A list containing the following components:

`pattern.list` a list of tables containing binary patterns in 0's and 1's. Each table is created by setting all non-zero entries in the corresponding sampled contingency table from `sample.list` to 1. Each table strictly satisfies the mathematical relationship required for a given pattern `type` requested, but it does not meet the statistical requirements. As each table represents the truth regarding the mathematical relationship between the row and column variables, they can be used as the ground truth or gold standard for benchmarking.

`sample.list` a list of tables satisfying both the mathematical and statistical requirements. These tables are noise free.

`noise.list` a list of tables after applying noise to the corresponding tables in `sample.list`. Each table is the noisy version of the corresponding sampled contingency table. Due to the added noise, each table may no longer strictly satisfy the required mathematical or statistical relationships. These tables are the main output to be used for the evaluation of a discrete pattern discovery algorithm.

`pvalue.list` a list of p-values reporting the statistical significance of the generated tables for the required type. When the pattern type specifies a functional relationship, the p-values are computed by the functional chi-square test (Zhang and Song 2013); otherwise, the Pearson's chi-square test of independence is used to calculate the p-value.

Author(s)

Ruby Sharma, Sajal Kumar, Hua Zhong, and Joe Song

References

Sharma R, Kumar S, Zhong H, Song M (2017). "Simulating noisy, nonparametric, and multivariate discrete patterns." *The R Journal*, **9**(2), 366–377. doi:10.32614/RJ2017053.

Zhang Y, Liu ZL, Song M (2015). "ChiNet uncovers rewired transcription subnetworks in tolerant yeast for advanced biofuels conversion." *Nucleic Acids Research*, **43**(9), 4393–4407. doi:10.1093/nar/gkv358.

Zhang Y, Song M (2013). "Deciphering interactions in causal networks without parametric assumptions." *arXiv Molecular Networks*, arXiv:1311.2707. https://arxiv.org/abs/1311.2707.

See Also

`add.noise` for details of the noise model.

Examples

```
# In all examples, x is the row variable and y is the column
#    variable of a table.

# Example 1. Simulating a noisy function where y=f(x),
#            x may or may not be g(y) with given row.marginal.

tbls <- simulate_tables(n=100, nrow=4, ncol=5, type="functional",
                noise=0.2, n.tables = 1,
                row.marginal = c(0.3,0.2,0.3,0.2))

par(mfrow=c(2,2))
plot_table(tbls$pattern.list[[1]], main="Ex 1. Functional pattern")
plot_table(tbls$sample.list[[1]], main="Ex 1. Sampled pattern (noise free)")
plot_table(tbls$noise.list[[1]], main="Ex 1. Sampled pattern with 0.2 noise")
plot.new()

# Example 2. Simulating a noisy functional pattern where
#            y=f(x), x may or may not be g(y) with given row.marginal.

tbls <- simulate_tables(n=100, nrow=4, ncol=5, type="functional",
                noise=0.5, n.tables = 1,
                row.marginal = c(0.3,0.2,0.3,0.2))
```

```
par(mfrow=c(2,2))
plot_table(tbls$pattern.list[[1]], main="Ex 2. Functioal pattern", col="seagreen2")
plot_table(tbls$sample.list[[1]], main="Ex 2. Sampled pattern (noise free)", col="seagreen2")
plot_table(tbls$noise.list[[1]], main="Ex 2. Sampled pattern with 0.5 noise", col="seagreen2")
plot.new()


# Example 3. Simulating a noisy many.to.one function where
#            y=f(x), x!=f(y) with given row.marginal.

tbls <- simulate_tables(n=100, nrow=4, ncol=5, type="many.to.one",
                noise=0.2, n.tables = 1,
                row.marginal = c(0.4,0.3,0.1,0.2))
par(mfrow=c(2,2))
plot_table(tbls$pattern.list[[1]], main="Ex 3. Many-to-one pattern", col="limegreen")
plot_table(tbls$sample.list[[1]], main="Ex 3. Sampled pattern (noise free)", col="limegreen")
plot_table(tbls$noise.list[[1]], main="Ex 3. Sampled pattern with 0.2 noise", col="limegreen")
plot.new()

# Example 4. Simulating noisy discontinuous
#   pattern where y=f(x), x may or may not be g(y) with given row.marginal.

tbls <- simulate_tables(n=100, nrow=4, ncol=5,
                type="discontinuous", noise=0.2,
                n.tables = 1, row.marginal = c(0.2,0.4,0.2,0.2))

par(mfrow=c(2,2))
plot_table(tbls$pattern.list[[1]], main="Ex 4. Discontinuous pattern", col="springgreen3")
plot_table(tbls$sample.list[[1]], main="Ex 4. Sampled pattern (noise free)", col="springgreen3")
plot_table(tbls$noise.list[[1]], main="Ex 4. Sampled pattern with 0.2 noise", col="springgreen3")
plot.new()


# Example 5. Simulating noisy dependent.non.functional
#            pattern where y!=f(x) and x and y are statistically
#            dependent.

tbls <- simulate_tables(n=100, nrow=4, ncol=5,
                type="dependent.non.functional", noise=0.3,
                n.tables = 1, row.marginal = c(0.2,0.4,0.2,0.2))

par(mfrow=c(2,2))
plot_table(tbls$pattern.list[[1]], main="Ex 5. Dependent.non.functional pattern",
col="sienna2", highlight="none")
plot_table(tbls$sample.list[[1]], main="Ex 5. Sampled pattern (noise free)",
col="sienna2", highlight="none")
plot_table(tbls$noise.list[[1]], main="Ex 5. Sampled pattern with 0.3 noise",
col="sienna2", highlight="none")
plot.new()

# Example 6. Simulating a pattern where x and y are
#            statistically independent.
```

```
tbls <- simulate_tables(n=100, nrow=4, ncol=5, type="independent",
                noise=0.3, n.tables = 1,
                row.marginal = c(0.4,0.3,0.1,0.2),
                col.marginal = c(0.1,0.2,0.4,0.2,0.1))

par(mfrow=c(2,2))
plot_table(tbls$pattern.list[[1]], main="Ex 6. Independent pattern",
col="cornflowerblue", highlight="none")
plot_table(tbls$sample.list[[1]], main="Ex 6. Sampled pattern (noise free)",
col="cornflowerblue", highlight="none")
plot_table(tbls$noise.list[[1]], main="Ex 6. Sampled pattern with 0.3 noise",
col="cornflowerblue", highlight="none")
plot.new()


# Example 7. Simulating a noisy function where y=f(x),
#            x may or may not be g(y), with given column marginal


tbls <- simulate_tables(n=100, nrow=4, ncol=5, type="functional",
                noise=0.2, n.tables = 1,
                col.marginal = c(0.2,0.1,0.4,0.2,0.1))

par(mfrow=c(2,2))
plot_table(tbls$pattern.list[[1]], main="Ex 7. Functional pattern")
plot_table(tbls$sample.list[[1]], main="Ex 7. Sampled pattern (noise free)")
plot_table(tbls$noise.list[[1]], main="Ex 7. Sampled pattern with 0.2 noise")
plot.new()


# Example 8. Simulating a noisy many.to.one function where
#            y=f(x), x!=f(y) with given column marginal.

tbls <- simulate_tables(n=100, nrow=4, ncol=4, type="many.to.one",
                noise=0.2, n.tables = 1,
                col.marginal = c(0.4,0.3,0.1,0.2))
par(mfrow=c(2,2))
plot_table(tbls$pattern.list[[1]], main="Ex 8. Many-to-one pattern", col="limegreen")
plot_table(tbls$sample.list[[1]], main="Ex 8. Sampled pattern (noise free)", col="limegreen")
plot_table(tbls$noise.list[[1]], main="Ex 8. Sampled pattern with 0.2 noise", col="limegreen")
plot.new()


# Example 9. Simulating noisy discontinuous
#   pattern where y=f(x), x may or may not be g(y) with given column marginal

tbls <- simulate_tables(n=100, nrow=4, ncol=4,
                type="discontinuous", noise=0.2,
                n.tables = 1, col.marginal = c(0.1,0.4,0.2,0.3))

par(mfrow=c(2,2))
plot_table(tbls$pattern.list[[1]], main="Ex 9. Discontinuous pattern", col="springgreen3")
plot_table(tbls$sample.list[[1]], main="Ex 9. Sampled pattern (noise free)", col="springgreen3")
```

```
plot_table(tbls$noise.list[[1]], main="Ex 9. Sampled pattern with 0.2 noise", col="springgreen3")
plot.new()
```

`test.interactions` *Functional Chi-Squared Test of Functional Dependency among Many Variables in a Data Set*

Description

Apply functional chi-squared tests on many-to-one combinatorial relationships for functional dependency using multivariate discrete data.

Usage

```
test.interactions(
  x, list.ind.vars, dep.vars, var.names = rownames(x),
  index.kind = c("conditional", "unconditional")
)
```

Arguments

`x`	A numeric matrix or data frame of discrete values. Rows represent variables and columns represent samples. Thus, each row index is a variable index, used by `list.ind.vars` and `dep.vars`.
`list.ind.vars`	A list of numeric or integer vectors, each vector representing independent variable indices in one interaction. Each vector (parents) forms a pair with a dependent variable (child) of the same position in `dep.vars` to represent a many-to-one directional interaction.
`dep.vars`	A numeric vector representing indices of dependent variables (children) in multiple interactions.
`var.names`	Optional. A character vector specifying names of all variables (rows). If not provided, the default is the row names of `x`; or `1:nrow(x)` if `x` does not have row names.
`index.kind`	A character string to specify the kind of function index to return, identical to the same argument in `fun.chisq.test`. The value can be `"unconditional"` (default) or `"conditional"`.

Details

`test.interactions` tests functional dependencies in multiple directional interactions. Each interaction, either one-to-one or many-to-one, is a parents-child pair representing a relationship from independent variables (parents) to a dependent variable (child). The parents-child pairs are specified in two input arguments `list.ind.vars` (a list of parents for each interaction) and `dep.vars` (vector of children in each interaction).

The function automatically creates contingency tables for interactions of interest, thus convenient to use on multivariate data sets. As the function is implemented in C++ and capable of testing multiple many-to-one interactions in one call, it is much faster than calling the R function `fun.chisq.test` multiple times.

`test.interactions` implements only the `method="fchisq"` option in `fun.chisq.test`.

When a contingency table is created for each interaction, all combinations of unique values of the independent variables (parents) form the rows and the unique values of dependent variable (child) form the columns in the contingency table. The table entries are the counts of the corresponding combination of parent and child values. Either rows or columns with all zero counts are removed from the contingency table before functional chi-squared test is applied.

Value

A data frame with five columns. Each row represents the testing result of each directional interaction. The 1st column is either the indices or names (if `var.names` is not `NULL`) of independent variables (parents); The 2nd column is the indices or names of the dependent variable (child); The 3rd column named `p.value` are p-values; The 4th column named `statistic` is chi-squared values; and the 5th column named `estimate` is the function indices for each interaction.

Author(s)

Hua Zhong and Joe Song

See Also

This function calls functional chi-squared test implemented in C++ and is thus much faster than the R version `fun.chisq.test`.

For data discretization by optimal univariate k-means clustering, see **Ckmeans.1d.dp**.

Examples

```
x <- matrix(
  c(0,0,1,0,1,
    1,0,2,1,0,
    2,2,0,0,0,
    1,2,1,1,2,
    1,0,2,1,2),
  nrow = 5, ncol = 5, byrow = TRUE)

list.ind.vars <-list(
  c(1),c(1),c(1),
  c(2),c(2),c(2),
```

```
  c(1,2), c(2,3),
  c(3,4), c(4,5))
dep.vars <- c(
  3,4,5,
  3,4,5,
  3,4,
  5,1)

# list.ind.vars and dep.vars together specify
#   the following ten interactions:
#   1 -> 3
#   1 -> 4
#   1 -> 5
#   2 -> 3
#   2 -> 4
#   2 -> 5
# 1,2 -> 3
# 2,3 -> 4
# 3,4 -> 5
# 4,5 -> 1

var.names <- paste0("var", 1:5)

test.interactions(
  x = x,
  list.ind.vars = list.ind.vars,
  dep.vars = dep.vars,
  var.names = var.names,
  index.kind = "unconditional")
```

Index

∗ **conditional functional dependency**
    cond.fun.chisq.test, 7
∗ **datagen**
    add.noise, 5
    FunChisq-package, 3
    simulate_tables, 17
∗ **hplot**
    plot_table, 16
∗ **htest**
    cp.fun.chisq.test, 9
    Exact Functional Test, 11
    fun.chisq.test, 12
    FunChisq-package, 3
    test.interactions, 23
∗ **noise model**
    add.noise, 5
∗ **nonparametric**
    cp.fun.chisq.test, 9
    Exact Functional Test, 11
    fun.chisq.test, 12
    FunChisq-package, 3
    test.interactions, 23
∗ **package**
    FunChisq-package, 3

add.candle.noise *(*add.noise*)*, 5
add.house.noise *(*add.noise*)*, 5
add.noise, 5, *18–20*

chisq.test, *4*, *15*
cond.fun.chisq.test, 7
cp.chisq.test, *10*, *16*
cp.chisq.test *(*FunChisq-deprecated*)*, 16
cp.fun.chisq.test, 9

EFTDP *(*Exact Functional Test*)*, 11
EFTDQP *(*Exact Functional Test*)*, 11
Exact Functional Test, 11

fisher.test, *4*, *15*
fun.chisq.test, *7*, *8*, *12*, 12, *24*
FunChisq-deprecated, 16
FunChisq-package, 3

image, *17*

par, *17*
plot_table, 16

simulate_tables, *6*, 17

test.interactions, 23