Package 'binGroup2'

July 22, 2025

Type Package

Title Identification and Estimation using Group Testing

Version 1.3.1

Date 2023-11-13

Maintainer Brianna Hitt <brianna.hitt@afacademy.af.edu>

Description Methods for the group testing identification problem: 1) Operating characteristics (e.g., expected number of tests) for commonly used hierarchical and array-based algorithms, and 2) Optimal testing configurations for these same algorithms. Methods for the group testing estimation problem: 1) Estimation and inference procedures for an overall prevalence, and 2) Regression modeling for commonly used hierarchical and array-based algorithms.

Imports ggplot2, graphics, grDevices, partitions, rBeta2009, Rcpp (>= 1.0.0), Rdpack, scales, stats, utils

RdMacros Rdpack

LinkingTo Rcpp, RcppArmadillo

License GPL (>= 3)

Encoding UTF-8

LazyData true

Author Brianna Hitt [aut, cre] (ORCID: <https://orcid.org/0000-0002-0645-0067>),
Christopher Bilder [aut] (ORCID: <https://orcid.org/0000-0002-2848-8576>),
Frank Schaarschmidt [aut] (ORCID: <https://orcid.org/0000-0002-6599-3803>),
Brad Biggerstaff [aut] (ORCID: <https://orcid.org/0000-0002-3105-3530>),
Christopher McMahan [aut] (ORCID: <https://orcid.org/0000-0001-5056-9615>),
Joshua Tebbs [aut] (ORCID: <https://orcid.org/0000-0002-6762-7241>),
Boan Zhang [ctb],
Michael Black [ctb],
Peijie Hou [ctb],

Peng Chen [ctb],
Minh Nguyen [ctb]

Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.2.3

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-11-14 01:43:21 UTC

Contents

`Accuracy` *Extract the accuracy measures from group testing results*

Description

Extract the accuracy measures from objects of class "opchar" returned by `operatingCharacteristics1` (`opChar1`) or `operatingCharacteristics2` (`opChar2`).

Usage

```
Accuracy(object, individual = TRUE, ...)
```

Arguments

`object`	An object of class "opChar", from which the accuracy measures are to be extracted.
`individual`	A logical argument that determines whether the accuracy measures for each individual (`individual=TRUE`) are to be included.
`...`	Additional arguments to be passed to `Accuracy` (e.g., `digits` to be passed to `round` or `signif` for appropriate rounding).

Details

The Accuracy function gives the individual accuracy measures for each individual in object and the overall accuracy measures for the algorithm. If individual=TRUE, individual accuracy measures are provided for each individual specified in the a argument of the call to operatingCharacteristics1 (opChar1) or operatingCharacteristics2 (opChar2).

Accuracy measures included are the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value. The overall accuracy measures displayed are weighted averages of the corresponding individual accuracy measures for all individuals in the algorithm. Expressions for these averages are provided in the Supplementary Material for Hitt et al. (2019). For more information, see the Details' section for the operatingCharacteristics1 (opChar1) or operatingCharacteristics2 (opChar2) function.

The rows in the matrices of individual accuracy measures correspond to each unique set of accuracy measures in the algorithm. Individuals with the same set of accuracy measures are displayed together in a single row of the matrix. The columns correspond to the pooling sensitivity, pooling specificity, pooling positive predictive value, pooling negative predictive value, and the indices for the individuals in each row of the matrix. Individual accuracy measures are provided only if individual=TRUE.

Value

A list containing:

Individual	matrix detailing the accuracy measures for each individual from object (for objects returned by opChar1).
Disease 1 Individual	matrix detailing the accuracy measures pertaining to disease 1 for each individual from object (for objects returned by opChar2).
Disease 2 Individual	matrix detailing the accuracy measures pertaining to disease 2 for each individual from object (for objects returned by opChar2).
Overall	matrix detailing the overall accuracy measures for the algorithm from object.

Author(s)

Brianna D. Hitt

Examples

```
config.mat <- matrix(data = c(rep(1, 10), 1:10),
                     nrow = 2, ncol = 10, byrow = TRUE)
res1 <- opChar1(algorithm = "D2", p = 0.05, Se = 0.99, Sp = 0.99,
        hier.config = config.mat)
Accuracy(res1, individual = FALSE)
Accuracy(res1, individual = TRUE)

res2 <- opChar2(algorithm = "A2M",
                p.vec = c(0.92, 0.05, 0.02, 0.01),
                Se = rep(0.95, 2), Sp = rep(0.99, 2),
                   rowcol.sz = 8)
Accuracy(res2)
```

`binGroup2` *binGroup2: Identification and Estimation using Group Testing*

Description

Methods for the group testing identification and estimation problems.

Details

Methods for identification of positive items in group testing designs: Operating characteristics (e.g., expected number of tests) are calculated for commonly used hierarchical and array-based algorithms. Optimal testing configurations for an algorithm can be found as well. Please see Hitt et al. (2019) for specific details.

Methods for estimation and inference for proportions in group testing designs: For estimating one proportion or the difference of proportions, confidence interval methods are included that account for different pool sizes. Functions for hypothesis testing of proportions, calculation of power, and calculation of the expected width of confidence intervals are also included. Furthermore, regression methods and simulation of group testing data are implemented for simple pooling (Dorfman testing with or without retests), halving, and array testing designs.

The `binGroup2` package is based upon the `binGroup` package that was originally designed for the group testing estimation problem. Over time, additional functions for estimation and for the group testing identification problem were included. Due to the diverse styles resulting from these additions, we have created `binGroup2` as a way to unify functions in a coherent structure and incorporate additional functions for identification. The `binGroup2` package provides all the main functionality from the `binGroup` package, and can be used in place of the `binGroup` package. The name "binGroup" originates from the assumption in basic estimation for group testing that the number of positive groups has a binomial distribution. While more advanced estimation methods no longer make this assumption, we continue with the `binGroup` name for consistency.

Bilder (2019a,b) provide introductions to group testing. These papers and additional details about group testing are available at `http://chrisbilder.com/grouptesting/`.

This research was supported by the National Institutes of Health under grant R01 AI121351.

Identification: The binGroup2 package focuses on the group testing identification problem using hierarchical and array-based group testing algorithms.

The `OTC1` function implements a number of group testing algorithms, described in Hitt et al. (2019), which calculate the operating characteristics and find the optimal testing configuration over a range of possible initial group sizes and/or testing configurations (sets of subsequent group sizes). The `OTC2` function does the same with a multiplex assay that tests for two diseases.

The `operatingCharacteristics1` (`opChar1`) and `operatingCharacteristics2` (`opChar2`) functions calculate operating characteristics for a specified testing configuration with assays that test for one and two diseases, respectively.

These functions allow the sensitivity and specificity to differ across stages of testing. This means that the accuracy of the diagnostic test can differ for stages in a hierarchical testing algorithm or between row/column testing and individual testing in an array testing algorithm.

Estimation: The binGroup2 package also provides functions for estimation and inference for proportions in group testing designs.

The `propCI` function calculates the point estimate and confidence intervals for a single proportion from group testing data. The `propDiffCI` function does the same for the difference of proportions. A number of confidence interval methods are available for groups of equal or different sizes.

The `gtWidth` function calculates the expected width of confidence intervals in group testing. The `gtTest` function calculates p-values for hypothesis tests of single proportions. The `gtPower` function calculates power to reject a hypothesis.

The `designPower` function iterates either the number of groups or group size in a one-parameter group testing design until a pre-specified power level is achieved. The `designEst` function finds the optimal group size corresponding to the minimal mean-squared error of the point estimator.

The `gtReg` function implements regression methods and the `gtSim` function simulates group testing data for simple pooling, halving, and array testing designs.

Author(s)

Maintainer: Brianna Hitt `<brianna.hitt@afacademy.af.edu>` (ORCID)

Authors:

- Christopher Bilder (ORCID)
- Frank Schaarschmidt (ORCID)
- Brad Biggerstaff (ORCID)
- Christopher McMahan (ORCID)
- Joshua Tebbs (ORCID)

Other contributors:

- Boan Zhang [contributor]
- Michael Black [contributor]
- Peijie Hou [contributor]
- Peng Chen [contributor]
- Minh Nguyen [contributor]

References

Altman, D., Bland, J. (1994). “Diagnostic tests 1: Sensitivity and specificity.” *BMJ*, **308**, 1552.

Altman, D., Bland, J. (1994). “Diagnostic tests 2: Predictive values.” *BMJ*, **309**, 102.

Biggerstaff, B. (2008). “Confidence intervals for the difference of proportions estimated from pooled samples.” *Journal of Agricultural, Biological, and Environmental Statistics*, **13**, 478–496.

Bilder, C., Tebbs, J., Chen, P. (2010). “Informative retesting.” *Journal of the American Statistical Association*, **105**, 942–955.

Bilder, C., Tebbs, J., McMahan, C. (2019). “Informative group testing for multiplex assays.” *Biometrics*, **75**, 278–288.

Bilder, C. (2019a). “Group Testing for Estimation.” *Wiley StatsRef: Statistics Reference Online*.

Bilder, C. (2019b). “Group Testing for Identification.” *Wiley StatsRef: Statistics Reference Online*.

Bilder, C., Iwen, P., Abdalhamid, B., Tebbs, J., McMahan, C. (2020). “Tests in short supply? Try group testing.” *Significance*, **17**, 15.

Black, M., Bilder, C., Tebbs, J. (2012). “Group testing in heterogeneous populations by using halving algorithms.” *Journal of the Royal Statistical Society. Series C: Applied Statistics*, **61**, 277–290.

Black, M., Bilder, C., Tebbs, J. (2015). “Optimal retesting configurations for hierarchical group testing.” *Journal of the Royal Statistical Society. Series C: Applied Statistics*, **64**, 693–710.

Graff, L., Roeloffs, R. (1972). “Group testing in the presence of test error; an extension of the Dorfman procedure.” *Technometrics*, **14**, 113–122.

Hepworth, G. (1996). “Exact confidence intervals for proportions estimated by group testing.” *Biometrics*, **52**, 1134–1146.

Hepworth, G., Biggerstaff, B. (2017). “Bias correction in estimating proportions by pooled testing.” *Journal of Agricultural, Biological, and Environmental Statistics*, **22**, 602–614.

Hitt, B., Bilder, C., Tebbs, J., McMahan, C. (2019). “The objective function controversy for group testing: Much ado about nothing?” *Statistics in Medicine*, **38**, 4912–4923.

Hou, P., Tebbs, J., Wang, D., McMahan, C., Bilder, C. (2021). “Array testing with multiplex assays.” *Biostatistics*, **21**, 417–431.

Malinovsky, Y., Albert, P., Roy, A. (2016). “Reader reaction: A note on the evaluation of group testing algorithms in the presence of misclassification.” *Biometrics*, **72**, 299–302.

McMahan, C., Tebbs, J., Bilder, C. (2012a). “Informative Dorfman Screening.” *Biometrics*, **68**, 287–296.

McMahan, C., Tebbs, J., Bilder, C. (2012b). “Two-Dimensional Informative Array Testing.” *Biometrics*, **68**, 793–804.

Schaarschmidt, F. (2007). “Experimental design for one-sided confidence intervals or hypothesis tests in binomial group testing.” *Communications in Biometry and Crop Science*, **2**, 32–40. ISSN 1896-0782.

Swallow, W. (1985). “Group testing for estimating infection rates and probabilities of disease transmission.” *Phytopathology*, **75**, 882–889.

Tebbs, J., Bilder, C. (2004). “Confidence interval procedures for the probability of disease transmission in multiple-vector-transfer designs.” *Journal of Agricultural, Biological, and Environmental Statistics*, **9**, 75–90.

Vansteelandt, S., Goetghebeur, E., Verstraeten, T. (2000). “Regression models for disease prevalence with diagnostic tests on pools of serum samples.” *Biometrics*, **56**, 1126–1133.

Verstraeten, T., Farah, B., Duchateau, L., Matu, R. (1998). “Pooling sera to reduce the cost of HIV surveillance: a feasibility study in a rural Kenyan district.” *Tropical Medicine & International Health*, **3**, 747–750.

Xie, M. (2001). “Regression analysis of group testing samples.” *Statistics in Medicine*, **20**, 1957–1969.

Examples

```
# 1) Identification using hierarchical and array-based
#   group testing algorithms with an assay that tests
#   for one disease.
# 1.1) Find the optimal testing configuration over a
#   range of initial group sizes, using informative
#   three-stage hierarchical testing, where
#   p denotes the overall prevalence of disease (mean
#    parameter of a beta distribution);
#   Se denotes the sensitivity of the diagnostic test;
#   Sp denotes the specificity of the diagnostic test;
#   group.sz denotes the range of initial pool sizes
#   for consideration;
#   obj.fn specifies the objective functions for which
#   to find results; and
#   alpha is the heterogeneity level.
set.seed(1002)
results1 <- OTC1(algorithm = "ID3", p = 0.025, Se = 0.95,
                 Sp = 0.95, group.sz = 3:20,
                 obj.fn = "ET", alpha = 2)
summary(results1)

# 1.2) Find the optimal testing configuration using
#   non-informative array testing without master pooling.
# The sensitivity and specificity differ for row/column
#   testing and individual testing.
results2 <- OTC1(algorithm = "A2", p = 0.05,
                 Se = c(0.95, 0.99), Sp = c(0.95, 0.98),
                 group.sz = 3:15, obj.fn = "ET")
summary(results2)

# 1.3) Calculate the operating characteristics using
#   informative two-stage hierarchical (Dorfman) testing,
#   implemented via the pool-specific optimal Dorfman
#   (PSOD) method described in McMahan et al. (2012a).
# Hierarchical testing configurations are specified by
#   a matrix in the hier.config argument. The rows of
#   the matrix correspond to the stages of the
#   hierarchical testing algorithm, the columns
#   correspond to the individuals to be tested, and the
#   cell values correspond to the group number of each
#   individual at each stage.
config.mat <- matrix(data = c(rep(1, 5), rep(2, 4), 3, 1:10),
                     nrow = 2, ncol = 10, byrow = TRUE)
set.seed(8791)
results3 <- opChar1(algorithm = "ID2", p = 0.02,
                    Se = 0.95, Sp = 0.99,
                    hier.config = config.mat, alpha = 0.5)
summary(results3)

# 1.4) Calculate the operating characteristics using
#   non-informative four-stage hierarchical testing.
config.mat <- matrix(data = c(rep(1, 15), rep(c(1, 2, 3), each = 5),
                              rep(1, 3), rep(2, 2),
                              rep(3, 3), rep(4, 2),
                              rep(5, 4), 6, 1:15),
                     nrow = 4, ncol = 15, byrow = TRUE)
results4 <- opChar1(algorithm = "D4", p = 0.008,
                    Se = 0.96, Sp = 0.98,
                    hier.config = config.mat,
                    a = c(1, 4, 6, 9, 11, 15))
summary(results4)


# 2) Identification using hierarchical and array-based
#   group testing algorithms with a multiplex assay that
#   tests for two diseases.

# 2.1) Find the optimal testing configuration using
#   non-informative two-stage hierarchical testing, given
#   p.vec, a vector of overall joint probabilities of disease;
#   Se, a vector of sensitivity values for each disease; and
#   Sp, a vector of specificity values for each disease.
# Se and Sp can also be specified as a matrix, where one
#   value is specified for each disease at each stage of
#   testing.
results5 <- OTC2(algorithm = "D2",
                 p.vec = c(0.90, 0.04, 0.04, 0.02),
                 Se = c(0.99, 0.99), Sp = c(0.99, 0.99),
                 group.sz = 3:20)
summary(results5)

# 2.2) Calculate the operating characteristics for
#   informative five-stage hierarchical testing, given
#   alpha.vec, a vector of shape parameters for the
#   Dirichlet distribution;
#   Se, a matrix of sensitivity values; and
#   Sp, a matrix of specificity values.
Se <- matrix(data = rep(0.95, 10), nrow = 2, ncol = 5, byrow = TRUE)
Sp <- matrix(data = rep(0.99, 10), nrow = 2, ncol = 5, byrow = TRUE)
config.mat <- matrix(data = c(rep(1, 24), rep(1, 18),
                              rep(2, 6), rep(1, 9),
                              rep(2, 9), rep(3, 4), 4, 5,
                              rep(1, 6), rep(2, 3),
                              rep(3, 5), rep(4, 4),
                              rep(5, 3), 6, rep(NA, 2),
                              1:21, rep(NA, 3)),
                     nrow = 5, ncol = 24, byrow = TRUE)
results6 <- opChar2(algorithm = "ID5",
                    alpha = c(18.25, 0.75, 0.75, 0.25),
                    Se = Se, Sp = Sp,
                    hier.config = config.mat)
summary(results6)

# 3) Estimation of the overall disease prevalence and
#   calculation of confidence intervals.

# 3.1) Suppose 3 groups out of 24 test positively.
#   Each group has a size of 7.
```

```
propCI(x = 3, m = 7, n = 24, ci.method = "CP")
propCI(x = 3, m = 7, n = 24, ci.method = "Blaker")
propCI(x = 3, m = 7, n = 24, ci.method = "score")
propCI(x = 3, m = 7, n = 24, ci.method = "soc")

# 3.2) Consider the following situation:
#   0 out of 5 groups test positively with groups
#   of size 1 (individual testing),
#   0 out of 5 groups test positively with groups of size 5,
#   1 out of 5 groups test positively with groups of size 10,
#   2 out of 5 groups test positively with groups of size 50
propCI(x = c(0, 0, 1, 2), m = c(1, 5, 10, 50),
       n = c(5, 5, 5, 5), pt.method = "Gart",
       ci.method = "skew-score")

# 4) Estimate a group testing regression model.

# 4.1) Fit a group testing regression model with
#   simple pooling using the "hivsurv" dataset.
data(hivsurv)
fit1 <- gtReg(type = "sp",
              formula = groupres ~ AGE + EDUC.,
              data = hivsurv, groupn = gnum,
              sens = 0.9, spec = 0.9, method = "Xie")
summary(fit1)

# 4.2) Simulate data for the halving protocol, and
#   fit a group testing regression model.
set.seed(46)
gt.data <- gtSim(type = "halving", par = c(-6, 0.1),
                 gshape = 17, gscale = 1.4,
                 size1 = 1000, size2 = 5,
                 sens = 0.95, spec = 0.95)
fit2 <- gtReg(type = "halving", formula = gres ~ x,
              data = gt.data, groupn = groupn,
              subg = subgroup, retest = retest,
              sens = 0.95, spec = 0.95,
              start = c(-6, 0.1), trace = TRUE)
summary(fit2)
```

coef.gtReg *Extract coefficients from a fitted group testing model*

Description

Extract coefficients from objects of class "gtReg" returned by gtReg.

Usage

```
## S3 method for class 'gtReg'
coef(object, digits = max(3, getOption("digits") - 3), ...)

## S3 method for class 'gtReg'
coefficients(object, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

`object`	An object of class "gtReg", created by `gtReg`, from which the coefficients are to be extracted.
`digits`	digits for rounding.
`...`	not currently used.

Value

Model coefficients extracted from the object `object`.

Author(s)

Brianna D. Hitt

Examples

```
data(hivsurv)
fit1 <- gtReg(formula = groupres ~ AGE * EDUC.,
              data = hivsurv, groupn = gnum,
              linkf = "probit")
coefficients(object = fit1)
```

`CompareConfig` *Compare group testing results*

Description

Compare group testing results from objects of class "opchar" returned by `operatingCharacteristics1` (`opChar1`) or `operatingCharacteristics2` (`opChar2`).

Usage

```
CompareConfig(object1, object2)
```

Arguments

`object1`	An object of class "opChar" containing group testing results.
`object2`	A second object of class "opChar" containing group testing results.

Details

The `CompareConfig` function compares group testing results from two objects of class "opChar". The function creates a data frame with these comparisons.

Value

A data frame with the expected percent reduction in tests (PercentReductionTests) and the expected increase in testing capacity (PercentIncreaseTestCap) when using the second testing configuration rather than the first testing configuration. Positive values for these quantities indicate that the second testing configuration is more efficient than the first.

Author(s)

Brianna D. Hitt and Christopher R. Bilder

Examples

```
config.mat1 <- matrix(data = c(rep(1, 10), rep(1:2, each = 5), 1:10),
                      nrow = 3, ncol = 10, byrow = TRUE)
res1 <- opChar1(algorithm = "D3", p = 0.05, Se = 0.99, Sp = 0.99,
                hier.config = config.mat1)
config.mat2 <- matrix(data = c(rep(1, 10), 1:10),
                      nrow = 2, ncol = 10, byrow = TRUE)
res2 <- opChar1(algorithm = "D2", p = 0.05, Se = 0.99, Sp = 0.99,
        hier.config = config.mat2)
CompareConfig(res2, res1)

config.mat3 <- matrix(data = c(rep(1, 10), rep(1, 5),
                               rep(2, 4), 3, 1:9, NA),
                      nrow = 3, ncol = 10, byrow = TRUE)
Se <- matrix(data = rep(0.95, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
Sp <- matrix(data = rep(0.99, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
res3 <- opChar2(algorithm = "D3", p.vec = c(0.95, 0.02, 0.02, 0.01),
                Se = Se, Sp = Sp, hier.config = config.mat3)
config.mat4 <- matrix(data = c(rep(1, 12), rep(1, 6), rep(2, 6),
                               rep(1, 4), rep(2, 2), rep(3, 3),
                               rep(4, 3), 1:12),
                    nrow = 4, ncol = 12, byrow = TRUE)
Se <- matrix(data = rep(0.95, 8), nrow = 2, ncol = 4,
             dimnames = list(Infection = 1:2, Stage = 1:4))
Sp <- matrix(data = rep(0.99, 8), nrow = 2, ncol = 4,
             dimnames = list(Infection = 1:2, Stage = 1:4))
res4 <- opChar2(algorithm = "D4", p.vec = c(0.92, 0.05, 0.02, 0.01),
                Se = Se, Sp = Sp, hier.config = config.mat4)
CompareConfig(res4, res3)
```

`Config` *Access the testing configurations returned from an object*

Description

`Config` is a generic function that extracts testing configurations from an object

Usage

```
Config(object, ...)
```

Arguments

`object`	An object from which the testing configurations are to be extracted.
`...`	Additional arguments to be passed to `Config`.

Author(s)

Christopher R. Bilder

See Also

`Config.opChar` and `Config.OTC`

Examples

```
# Find the optimal testing configuration for
#   non-informative two-stage hierarchical testing.
res1 <- OTC1(algorithm = "D2", p = 0.01, Se = 0.99, Sp = 0.99,
             group.sz = 2:100, obj.fn = c("ET", "MAR", "GR1"),
             weights = matrix(data = c(1,1), nrow = 1, ncol = 2))
Config(res1)
```

`Config.opChar` *Extract the testing configuration from group testing results*

Description

Extract the testing configuration from objects of class "opchar" returned by `operatingCharacteristics1` (opChar1) or `operatingCharacteristics2` (opChar2).

Usage

```
## S3 method for class 'opChar'
Config(object, ...)
```

Arguments

`object`	An object of class "opChar", from which the testing configuration is to be extracted.
`...`	currently not used.

Value

A data frame specifying elements of the testing configuration.

Author(s)

Brianna D. Hitt

Examples

```
config.mat <- matrix(data = c(rep(1, 10), 1:10),
                     nrow = 2, ncol = 10, byrow = TRUE)
res1 <- opChar1(algorithm = "D2", p = 0.05, Se = 0.99, Sp = 0.99,
        hier.config = config.mat)
Config(res1)

config.mat <- matrix(data = c(rep(1, 20), rep(1, 10), rep(2, 10),
                             rep(c(1, 2, 3, 4), each = 5),
                             rep(1, 3), rep(2, 2), rep(3, 3),
                             rep(4, 2), rep(5, 3), rep(6, 2),
                             rep(7, 3), rep(8, 2), 1:20),
                    nrow = 5, ncol = 20, byrow = TRUE)
Se <- matrix(data = rep(0.95, 10), nrow = 2, ncol = 5,
             dimnames = list(Infection = 1:2, Stage = 1:5))
Sp <- matrix(data = rep(0.99, 10), nrow = 2, ncol = 5,
             dimnames = list(Infection = 1:2, Stage = 1:5))
res2 <- opChar2(algorithm = "ID5",
                alpha = c(18.25, 0.75, 0.75, 0.25),
                Se = Se, Sp = Sp, hier.config = config.mat)
Config(res2)
```

`Config.OTC` *Extract the testing configuration from group testing results*

Description

Extract the testing configuration from objects of class "OTC" returned by `OTC1` (`OTC1`) or `OTC2` (`OTC2`).

Usage

```
## S3 method for class 'OTC'
Config(object, n = 5, top.overall = FALSE, ...)
```

Arguments

`object`	An object of class "OTC", from which the testing configuration is to be extracted.
`n`	Number of testing configurations.
`top.overall`	logical; if TRUE, best overall testing configurations; if FALSE, best testing configurations by initial group size
`...`	currently not used.

Value

A data frame providing the best testing configurations.

Author(s)

Christopher R. Bilder

Examples

```
res1 <- OTC1(algorithm = "D3", p = 0.05, Se = 0.99, Sp = 0.99,
             group.sz = 3:15, obj.fn = "ET")
Config(res1)
```

`designEst` *Optimal group size determination based on minimal MSE when estimating an overall prevalence*

Description

Find the group size `s` for a fixed number of groups `n` and an assumed true proportion `p.tr`, for which the mean squared error (MSE) of the point estimator is minimal and bias is within a restriction.

Usage

```
designEst(n, smax, p.tr, biasrest = 0.05)
```

Arguments

`n`	integer specifying the fixed number of groups.
`smax`	integer specifying the maximum group size allowed in the planning of the design.
`p.tr`	assumed true proportion of the "positive" trait in the population, specified as a value between 0 and 1.
`biasrest`	a value between 0 and 1 specifying the absolute bias maximally allowed.

Details

Swallow (1985) recommends the use of the upper bound of the expected range of the true proportion `p.tr` for optimization of the design. For further details, see Swallow (1985). Note that the specified number of groups must be less than $n = 1020$.

Value

A list containing:

`call`	the function call
`result`	a data frame containing: **mse** the mean squared error of the estimator. **sout** the group size `s` for which the MSE of the estimator is minimal for the given `n` and `p.tr` and for which the bias restriction `biasrest` is not violated. In the case that the minimum MSE is achieved for a group size $s >= smax$, the value of `smax` is returned. **exp** the expected value of the estimator. **varp** the variance of the estimator. **bias** the bias of the estimator.
`bias.reached`	a logical value indicating whether the bias restriction `biasrest` was violated.
`smax.reached`	a logical value indicating whether the maximum group size allowed `smax` was reached.

Author(s)

This function was originally written by Frank Schaarschmidt as the `estDesign` function for the `binGroup` package. Minor modifications were made for inclusion in the `binGroup2` package.

References

Swallow, W. (1985). "Group testing for estimating infection rates and probabilities of disease transmission." *Phytopathology*, **75**, 882–889.

See Also

`designPower` for choice of the group testing design according to the power in a hypothesis test.

Other estimation functions: `designPower()`, `gtPower()`, `gtTest()`, `gtWidth()`, `propCI()`, `propDiffCI()`

Examples

```
# Compare to Table 1 in Swallow (1985):
designEst(n = 10, smax = 100, p.tr = 0.001)
designEst(n = 10, smax = 100, p.tr = 0.01)
designEst(n = 25, smax = 100, p.tr = 0.05)
designEst(n = 40, smax = 100, p.tr = 0.25)
designEst(n = 200, smax = 100, p.tr = 0.30)
```

designPower — *Number of groups or group size needed to achieve a power level in one parameter group testing*

Description

For a fixed number of groups (group size), determine the group size (number of groups) needed to obtain a specified power level to reject a hypothesis for a proportion in one parameter group testing.

Usage

```
designPower(
  n,
  s,
  fixed = "s",
  delta,
  p.hyp,
  conf.level = 0.95,
  power = 0.8,
  alternative = "two.sided",
  method = "CP",
  biasrest = 0.05
)
```

Arguments

`n`	integer specifying the maximum number of groups `n` allowed when `fixed="s"` or the fixed number of groups when `fixed="n"`. When `fixed="s"`, a vector of two integers giving the range of `n` which power shall be iterated over is also allowed.
`s`	integer specifying the fixed group size (number of units per group) when `fixed="s"` or the maximum group size allowed in the planning of the design when `fixed="n"`.
`fixed`	character string specifying whether the number of groups `"n"` or the group size `"s"` is to be held at a fixed value.
`delta`	the absolute difference between the true proportion and the hypothesized proportion which shall be detectable with the specified power.
`p.hyp`	the proportion in the hypotheses, specified as a value between 0 and 1.
`conf.level`	confidence level of the decision. The default confidence level is 0.95.
`power`	level of power to be achieved, specified as a probability between 0 and 1.
`alternative`	character string defining the alternative hypothesis, either `"two.sided"`, `"less"`, or `"greater"`.
`method`	character string specifying the confidence interval method (see `propCI`) to be used.
`biasrest`	a value between 0 and 1, specifying the absolute bias maximally allowed for a point estimate.

Details

The power of a hypothesis test performed by a confidence interval is defined as the probability that a confidence interval excludes the threshold parameter (`p.hyp`) of the hypothesis.

When `fixed="s"`, this function increases the number of groups until a pre-specified level of power is reached or the maximum number of groups `n` is reached. Since the power does not increase monotonically with increasing `n` for single proportions but oscillates between local maxima and minima, the simple iteration given here will generally result in selecting `n` for which the given confidence interval method shows a local minimum of coverage if the null hypothesis is true. Bias decreases monotonically with increasing the number of groups (if other parameters are fixed). The resulting problems of choosing a number of groups which results in satisfactory power are solved in the following manner:

In the case that the pre-specified power is reached within the given range of `n`, the smallest `n` is returned for which at least this power is reached, as well as the actual power for this `n`.

In the case that the pre-specified power is not reached within the given value, that `n` is returned for which maximum power is achieved, and the corresponding value of power.

In the case that the bias restriction is violated even for the largest `n` within the given range of `n`, simply that `n` will be returned for which power was largest in the given range.

Especially for large `n`, the calculation time may become large (particularly for the Blaker interval). Alternatively, the function `gtPower` might be used to calculate power and bias only for some particular combinations of the input arguments.

When `fixed="n"`, this function increases the size of groups until a pre-specified level of power is reached. Since the power does not increase monotonically with increasing `s` for single proportions but oscillates between local maxima and minima, the simple iteration given here will generally result in selecting `s` for which the given confidence interval method shows a local minimum of coverage if the null hypothesis is true. Since the positive bias of the estimator in group testing increases with increasing group size, this function checks whether the bias is smaller than a pre-specified level (`bias.rest`). If the bias violates this restriction for a given combination `n`, `s`, and `delta`, `s` will not be further increased and the actual power of the last acceptable group size `s` is returned.

Value

A list containing:

`nout`	the number of groups necessary to reach the power with the specified parameters, when `fixed="s"` only.
`sout`	the group size necessary to meet the conditions, when `fixed="n"` only.
`powerout`	the power for the specified parameters and the selected number of groups `n` when `fixed="s"` or the selected group size `s` when `fixed="n"`.
`biasout`	the bias for the specified parameters and the selected number of groups `n` when `fixed="s"` or the selected group size `s` when `fixed="n"`.
`power.reached`	a logical value indicating whether the specified level of power was reached.
`bias.reached`	a logical value indicating whether the maximum allowed bias was reached.
`nit`	the number of groups for each iteration.
`sit`	the group size for each iteration.

`powerit`	the power achieved for each iteration.
`biasit`	the bias for each iteration.
`maxit`	the iteration at which the maximum power was reached, or the total number of iterations.
`alternative`	the alternative hypothesis specified by the user.
`p.hyp`	the hypothesized proportion specified by the user.
`delta`	the absolute difference between the true proportion and the hypothesized proportion specified by the user.
`power`	the desired power specified by the user.
`biasrest`	the maximum absolute bias specified by the user.

Author(s)

The `nDesign` and `sDesign` functions were originally written by Frank Schaarschmidt for the `binGroup` package. Minor modifications were made for inclusion in the `binGroup2` package.

References

Swallow, W. (1985). "Group testing for estimating infection rates and probabilities of disease transmission." *Phytopathology*, **75**, 882–889.

See Also

`gtPower` for calculation of power and bias depending on `n`, `s`, `delta`, `p.hyp`, `conf.level`, and `method`, and `designEst` to choose the group size `s` according to the minimal mse of the estimator, as given in Swallow (1985).

Other estimation functions: `designEst()`, `gtPower()`, `gtTest()`, `gtWidth()`, `propCI()`, `propDiffCI()`

Examples

```
# Assume the objective is to show that a proportion is
#   smaller than 0.005 (i.e. 0.5 percent) with a power
#   of 0.80 (i.e. 80 percent) if the unknown proportion
#   in the population is 0.003 (i.e. 0.3 percent);
#   thus, a delta of 0.002 shall be detected.
# A 95% Clopper Pearson CI shall be used.
# The maximum group size because of limited sensitivity
#   of the diagnostic test might be s=20 and we can
#   only afford to perform maximally 100 tests:
designPower(n = 100, s = 20, delta = 0.002,
            p.hyp = 0.005, fixed = "s",
            alternative = "less", method = "CP",
            power = 0.8)

# One might accept to detect delta=0.004,
#   i.e. reject H0: p>=0.005 with power 80 percent
#   when the true proportion is 0.001:
designPower(n = 100, s = 20, delta = 0.004, p.hyp = 0.005, fixed = "s",
             alternative = "less", method = "CP", power = 0.8)

# Power for a design with a fixed group size of s = 1
#   (individual testing).
designPower(n = 200, s = 1, delta = 0.05, p.hyp = 0.10,
            fixed = "s", method = "CP", power = 0.80)

# Assume that objective is to show that a proportion
#   is smaller than 0.005 (i.e. 0.5%) with a
#   power of 0.80 (i.e. 80%) if the unknown proportion
#   in the population is 0.003 (i.e. 0.3%); thus, a
#   delta = 0.002 shall be detected.
# A 95% Clopper-Pearson CI shall be used.
# The maximum number of groups might be 30, where the
#   overall sensitivity is not limited until group
#   size s=100.
designPower(s = 100, n = 30, delta = 0.002, p.hyp = 0.005, fixed = "n",
             alternative = "less", method = "CP", power = 0.8)

# One might accept to detect delta=0.004,
#   i.e. reject H0: p>=0.005 with power 80 percent
#   when the true proportion is 0.001:
designPower(s = 100, n = 30, delta = 0.004, p.hyp = 0.005, fixed = "n",
             alternative = "less", method = "CP", power = 0.8)
designPower(s = 100, n = 30, delta = 0.004, p.hyp = 0.005, fixed = "n",
             alternative = "less", method = "score", power = 0.8)
```

---

expectOrderBeta *Determine a vector of probabilities for informative group testing algorithms*

Description

Find the expected value of order statistics from a beta distribution. This function is used to provide a set of individual risk probabilities for informative group testing.

Usage

```
expectOrderBeta(
  p,
  alpha,
  size,
  grp.sz,
  num.sim = 10000,
 rel.tol = ifelse(alpha >= 1, .Machine$double.eps^0.25, .Machine$double.eps^0.1),
  ...
)
```

Arguments

`p`	overall probability of disease that will be used to determine a vector of individual risk probabilities. This is the expected value of a random variable with a beta distribution, $\frac{\alpha}{\alpha+\beta}$.
`alpha`	a shape parameter for the beta distribution that specifies the degree of heterogeneity for the determined probability vector.
`size`	the size of the vector of individual risk probabilities to be generated. This is also the number of total individuals for which to determine risk probabilities.
`grp.sz`	the number of total individuals for which to determine risk probabilities. This argument is deprecated; the `size` argument should be used instead.
`num.sim`	the number of simulations. This argument is used only when simulation is necessary.
`rel.tol`	relative tolerance used for integration.
`...`	arguments to be passed to the `beta.dist` function written by Michael Black for Black et al. (2015).

Details

This function uses the `beta.dist` function from Black et al. (2015) to determine a vector of individual risk probabilities, ordered from least to greatest. Depending on the specified probability, α level, and overall group size, simulation may be necessary in order to determine the probabilities. For this reason, the user should set a seed in order to reproduce results. The number of simulations (default = 10,000) and relative tolerance for integration can be specified by the user. The `expectOrderBeta` function augments the `beta.dist` function by checking whether simulation is needed before attempting to determine the probabilities, and by allowing the number of simulations to be specified by the user. See Black et al. (2015) for additional details on the original `beta.dist` function.

Value

A vector of individual risk probabilities.

Author(s)

Brianna D. Hitt

References

Black, M., Bilder, C., Tebbs, J. (2015). “Optimal retesting configurations for hierarchical group testing.” *Journal of the Royal Statistical Society. Series C: Applied Statistics*, **64**, 693–710.

See Also

`informativeArrayProb` for arranging a vector of individual risk probabilities in a matrix for informative array testing without master pooling.

Examples

```
set.seed(8791)
expectOrderBeta(p = 0.03, alpha = 0.5, size = 100, rel.tol = 0.0001)

expectOrderBeta(p = 0.05, alpha = 2, size = 40)
```

ExpTests *Access the expected number of tests from an object*

Description

`ExpTests` is a generic function that extracts the expected number of tests from an object that contains information aboout a testing configuration.

Usage

```
ExpTests(object, ...)
```

Arguments

`object`	An object for which a summary of the expected number of tests is desired.
`...`	Additional arguments to be passed to `ExpTests`.

Value

The value return depends on the class of its object. See the documentation for the corresponding method functions.

Author(s)

Christopher R. Bilder

See Also

`ExpTests.opChar` and `ExpTests.OTC`

Examples

```
# Find the optimal testing configuration for
#   non-informative two-stage hierarchical testing.
res1 <- OTC1(algorithm = "D2", p = 0.01, Se = 0.99, Sp = 0.99,
             group.sz = 2:100, obj.fn = c("ET", "MAR", "GR1"),
             weights = matrix(data = c(1,1), nrow = 1, ncol = 2))
ExpTests(res1)
```

`ExpTests.halving` *Extract the expected number of tests from testing configuration results*

Description

Extract the expected number of tests from objects of class "halving" returned by `halving` (`halving`).

Usage

```
## S3 method for class 'halving'
ExpTests(object, ...)
```

Arguments

`object`	An object of class "halving", from which the expected number of tests is to be extracted.
`...`	Additional arguments to be passed to `ExpTests` (e.g., `digits` to be passed to `round` for appropriate rounding).

Value

A data frame containing the columns:

`ExpTests`	the expected number of tests required to decode all individuals in the algorithm.
`ExpTestsPerIndividual`	the expected number of tests per individual.
`PercentReductionTests`	The percent reduction in the number of tests; 100 * (1 - ExpTestsPerIndividual).
`PercentIncreaseTestCap`	The percent increase in testing capacity when the algorithm is applied to a continuous stream of specimens; 100 * (1/ExpTestsPerIndividual - 1).

Author(s)

Christopher R. Bilder

References

Bilder, C., Iwen, P., Abdalhamid, B., Tebbs, J., McMahan, C. (2020). “Tests in short supply? Try group testing.” *Significance*, **17**, 15.

Examples

```
save.it1 <- halving(p = rep(0.01, 10), Sp = 1, Se = 1, stages = 2,
        order.p = TRUE)
ExpTests(save.it1)
```

ExpTests.opChar — *Extract the expected number of tests from testing configuration results*

Description

Extract the expected number of tests and expected number of tests per individual from objects of class "opchar" returned by `operatingCharacteristics1` (`opChar1`) or `operatingCharacteristics2` (`opChar2`).

Usage

```
## S3 method for class 'opChar'
ExpTests(object, ...)
```

Arguments

`object`	An object of class "opChar", from which the expected number of tests and expected number of tests per individual are to be extracted.
`...`	Additional arguments to be passed to `ExpTests` (e.g., `digits` to be passed to `round` for appropriate rounding).

Value

A data frame containing the columns:

`ExpTests`	the expected number of tests required to decode all individuals in the algorithm.
`ExpTestsPerIndividual`	the expected number of tests per individual.
`PercentReductionTests`	The percent reduction in the number of tests; 100 * (1 - ExpTestsPerIndividual).
`PercentIncreaseTestCap`	The percent increase in testing capacity when the algorithm is applied to a continuous stream of specimens; 100 * (1/ExpTestsPerIndividual - 1).

Author(s)

Brianna D. Hitt and Christopher R. Bilder

References

Bilder, C., Iwen, P., Abdalhamid, B., Tebbs, J., McMahan, C. (2020). “Tests in short supply? Try group testing.” *Significance*, **17**, 15.

Examples

```
config.mat <- matrix(data = c(rep(1, 10), 1:10),
                     nrow = 2, ncol = 10, byrow = TRUE)
res1 <- opChar1(algorithm = "D2", p = 0.05, Se = 0.99, Sp = 0.99,
                hier.config = config.mat)
ExpTests(res1)

res2 <- opChar2(algorithm = "A2M", p.vec = c(0.92, 0.05, 0.02, 0.01),
                Se = rep(0.95, 2), Sp = rep(0.99, 2), rowcol.sz = 8)
ExpTests(res2)
```

`ExpTests.OTC` *Extract the expected number of tests from optimal testing configuration results*

Description

Extract the expected number of tests and expected number of tests per individual from objects of class "OTC" returned by `OTC1` or `OTC2`.

Usage

```
## S3 method for class 'OTC'
ExpTests(object, ...)
```

Arguments

`object`	An object of class "OTC", from which the expected number of tests and expected number of tests per individual are to be extracted.
`...`	Additional arguments to be passed to `ExpTests` (e.g., `digits` to be passed to `round` for appropriate rounding).

Value

A data frame containing the columns:

`ExpTests`	the expected number of tests required by the optimal testing configuration.
`ExpTestsPerInd`	the expected number of tests per individual for the optimal testing configuration.
`PercentReductionTests`	The percent reduction in the number of tests; 100 * (1 - ExpTestsPerIndividual).
`PercentIncreaseTestCap`	The percent increase in testing capacity when the algorithm is applied to a continuous stream of specimens; 100 * (1/ExpTestsPerIndividual - 1).

Each row of the data frame represents an objective function specified in the call to `OTC1` or `OTC2`.

Author(s)

Brianna D. Hitt and Christopher R. Bilder

References

Bilder, C., Iwen, P., Abdalhamid, B., Tebbs, J., McMahan, C. (2020). "Tests in short supply? Try group testing." *Significance*, **17**, 15.

Examples

```
res1 <- OTC1(algorithm = "D2", p = 0.05, Se = 0.99, Sp = 0.99,
             group.sz = 2:100, obj.fn = c("ET", "MAR"),
             trace = TRUE)
ExpTests.OTC(res1)
```

`ExpTests.Sterrett` *Extract the expected number of tests from testing configuration results*

Description

Extract the expected number of tests from objects of class "Sterrett" returned by `Sterrett` (`Sterrett`).

Usage

```
## S3 method for class 'Sterrett'
ExpTests(object, ...)
```

Arguments

`object`	An object of class "Sterrett", from which the expected number of tests is to be extracted.
`...`	Additional arguments to be passed to `ExpTests` (e.g., `digits` to be passed to `round` for appropriate rounding).

Value

A data frame containing the columns:

`ExpTests`	the expected number of tests required to decode all individuals in the algorithm.
`ExpTestsPerIndividual`	the expected number of tests per individual.
`PercentReductionTests`	The percent reduction in the number of tests; 100 * (1 - ExpTestsPerIndividual).
`PercentIncreaseTestCap`	The percent increase in testing capacity when the algorithm is applied to a continuous stream of specimens; 100 * (1/ExpTestsPerIndividual - 1).

Author(s)

Christopher R. Bilder

References

Bilder, C., Iwen, P., Abdalhamid, B., Tebbs, J., McMahan, C. (2020). “Tests in short supply? Try group testing.” *Significance*, **17**, 15.

Examples

```
set.seed(1231)
p.vec1 <- rbeta(n = 8, shape1 = 1, shape2 = 10)
save.it1 <- Sterrett(p = p.vec1, Sp = 0.90, Se = 0.95)
ExpTests(save.it1)
```

`ExpTests.TOD` *Extract the expected number of tests from testing configuration results*

Description

Extract the expected number of tests from objects of class "TOD" returned by `TOD` (`TOD`).

Usage

```
## S3 method for class 'TOD'
ExpTests(object, ...)
```

Arguments

`object`	An object of class "TOD", from which the expected number of tests is to be extracted.
`...`	Additional arguments to be passed to `ExpTests` (e.g., `digits` to be passed to `round` for appropriate rounding).

Value

A data frame containing the columns:

`ExpTests`	the expected number of tests required to decode all individuals in the algorithm.
`ExpTestsPerIndividual`	the expected number of tests per individual.
`PercentReductionTests`	The percent reduction in the number of tests; 100 * (1 - ExpTestsPerIndividual).
`PercentIncreaseTestCap`	The percent increase in testing capacity when the algorithm is applied to a continuous stream of specimens; 100 * (1/ExpTestsPerIndividual - 1).

Author(s)

Christopher R. Bilder

References

Bilder, C., Iwen, P., Abdalhamid, B., Tebbs, J., McMahan, C. (2020). “Tests in short supply? Try group testing.” *Significance*, **17**, 15.

Examples

```
set.seed(1002)
p.vec <- expectOrderBeta(p = 0.01, alpha = 2, size = 20)
save.it1 <- TOD(p = p.vec, Se = 0.95, Sp = 0.95, max = 5, threshold = 0.015)
ExpTests(save.it1)
```

---

formula.gtReg *Extract the model formula from a fitted group testing model*

Description

Extract the model formula from objects of class "gtReg" returned by gtReg.

Usage

```
## S3 method for class 'gtReg'
formula(x, ...)
```

Arguments

x	An object of class "gtReg", created by gtReg, from which the model formula is to be extracted.
...	not currently used.

Value

Model formula extracted from the object `object`.

Author(s)

Brianna D. Hitt

Examples

```
data(hivsurv)
fit1 <- gtReg(formula = groupres ~ AGE * EDUC.,
              data = hivsurv, groupn = gnum,
              linkf = "probit")
formula(x = fit1)
```

`GroupMembershipMatrix` *Construct a group membership matrix for hierarchical algorithms*

Description

Construct a group membership matrix for two-, three-, or four-stage hierarchical algorithms.

Usage

```
GroupMembershipMatrix(stage1, stage2 = NULL, stage3 = NULL, stage4 = NULL)
```

Arguments

`stage1`	the group size in stage one of testing. This also corresponds to the number of individuals to be tested and will specify the number of columns in the resulting group membership matrix.
`stage2`	a vector of group sizes in stage two of testing. The group sizes specified here should sum to the number of individuals/group size specified in `stage1`. If `NULL`, a group membership matrix will be constructed for a two-stage hierarchical algorithm. Further details are given under 'Details'.
`stage3`	a vector of group sizes in stage three of testing. The group sizes specified here should sum to the number of individuals/group size specified in `stage1`. If group sizes are provided in `stage2` and `stage3` is `NULL`, a group membership matrix will be constructed for a three-stage hierarchical algorithm. Further details are given under 'Details'.
`stage4`	a vector of group sizes in stage four of testing. The group sizes specified here should sum to the number of individuals/group size specified in `stage1`. If group sizes are provided in `stage3` and `stage4` is `NULL`, a group membership matrix will be constructed for a four-stage hierarchical algorithm. Further details are given under 'Details'.

Details

This function constructs a group membership matrix for two-, three-, four-, or five-stage hierarchical algorithms. The resulting group membership matrix has rows corresponding to the number of stages of testing and columns corresponding to each individual to be tested. The value specified in `stage1` corresponds to the number of individuals to be tested.

For group membership matrices when only `stage1` is specified, a two-stage hierarchical algorithm is used and the second stage will consist of individual testing. For group membership matrices when `stage1` and `stage2` are specified, a three-stage hierarchical algorithm is used and the third stage will consist of individual testing. Group membership matrices for four- and five-stage hierarchical algorithms follow a similar structure. There should never be group sizes specified for later stages of testing without also providing group sizes for all earlier stages of testing (i.e., to provide group sizes for `stage3`, group sizes must also be provided for `stage1` and `stage2`).

Value

A matrix specifying the group membership for each individual. The rows of the matrix correspond to the stages of testing and the columns of the matrix correspond to the individuals to be tested.

Author(s)

Minh Nguyen and Christopher Bilder

See Also

Other operating characteristic functions: `Sterrett()`, `TOD()`, `halving()`, `operatingCharacteristics1()`, `operatingCharacteristics2()`

Examples

```
# Generate a group membership matrix for a two-stage
#   hierarchical algorithm, within the opChar1() function
#   and calculate operating characteristics
opChar1(algorithm = "D2", p = 0.0193, Se = 0.99, Sp = 0.99,
        hier.config = GroupMembershipMatrix(stage1 = 16),
        print.time = FALSE)

# Generate a group membership matrix for a five-stage
#   hierarchical algorithm and calculate the
#   operating characteristics for a two-disease assay
config.mat <- GroupMembershipMatrix(stage1 = 16,
                                    stage2 = c(8,8),
                                    stage3 = c(4,4,4,4),
                                    stage4 = rep(2, times = 8))
Se <- matrix(data = rep(0.95, 10), nrow = 2, ncol = 5,
             dimnames = list(Infection = 1:2, Stage = 1:5))
Sp <- matrix(data = rep(0.99, 10), nrow = 2, ncol = 5,
             dimnames = list(Infection = 1:2, Stage = 1:5))
opChar2(algorithm = "D5", p.vec = c(0.92, 0.05, 0.02, 0.01),
        Se = Se, Sp = Sp, hier.config = config.mat)
```

---

gtPower *Power to reject a hypothesis for one proportion in group testing*

Description

This function calculates the power to reject a hypothesis in a group testing experiment, using confidence intervals for the decision. This function also calculates the bias of the point estimator for a given n, s, and true, unknown proportion.

Usage

```
gtPower(
  n,
  s,
  delta,
  p.hyp,
  conf.level = 0.95,
  method = "CP",
  alternative = "two.sided"
)
```

Arguments

`n`	integer specifying the number of groups. A vector of integers is also allowed.
`s`	integer specifying the common group size. A vector of integers is also allowed.
`delta`	the absolute difference between the true proportion and the hypothesized proportion. A vector is also allowed.
`p.hyp`	the proportion in the hypotheses, specified as a value between 0 and 1.
`conf.level`	confidence level required for the decision on the hypotheses.
`method`	character string specifying the confidence interval method (see `propCI`) to be used.
`alternative`	character string defining the alternative hypothesis, either `"two.sided"`, `"less"`, or `"greater"`.

Details

The power of a hypothesis test performed by a confidence interval is defined as the probability that a confidence interval excludes the threshold parameter (`p.hyp`) of the null hypothesis, as described in Schaarschmidt (2007). Due to discreteness, the power does not increase monotonically for an increasing number of groups n or group size s, but exhibits local maxima and minima, depending on n, s, `p.hyp`, and `conf.level`.

Additional to the power, the bias of the point estimator is calculated according to Swallow (1985). If vectors are specified for n, s, and (or) delta, a matrix will be constructed and power and bias are calculated for each line in this matrix.

Value

A matrix containing the following columns:

`ns`	a vector of the total sample size, $n * s$.
`n`	a vector of the number of groups.
`s`	a vector of the group sizes.
`delta`	a vector of the delta values.
`power`	the power to reject the given null hypothesis.
`bias`	the bias of the estimator for the specified n, s, and the true proportion.

Author(s)

This function was originally written as `bgtPower` by Frank Schaarschmidt for the `binGroup` package. Minor modifications have been made for inclusion of the function in the `binGroup2` package.

References

Schaarschmidt, F. (2007). “Experimental design for one-sided confidence intervals or hypothesis tests in binomial group testing.” *Communications in Biometry and Crop Science*, **2**, 32–40. ISSN 1896-0782.

Swallow, W. (1985). “Group testing for estimating infection rates and probabilities of disease transmission.” *Phytopathology*, **75**, 882–889.

See Also

`propCI` for confidence intervals and `gtTest` for hypothesis tests for one proportion from a group testing experiment.

Other estimation functions: `designEst()`, `designPower()`, `gtTest()`, `gtWidth()`, `propCI()`, `propDiffCI()`

Examples

```
# Calculate the power for the design
#   in the example given in Tebbs and Bilder(2004):
#   n=24 groups each containing 7 insects
#   if the true proportion of virus vectors
#   in the population is 0.04 (4 percent),
#   the power to reject H0: p>=0.1 using an
#   upper Clopper-Pearson ("CP") confidence interval
#   is calculated with the following call:
gtPower(n = 24, s = 7, delta = 0.06, p.hyp = 0.1,
        conf.level = 0.95, alternative = "less",
        method = "CP")

# Explore development of power and bias for varying n,
#   s, and delta. How much can we decrease the number of
#   groups (costly tests to be performed) by pooling the
#   same number of 320 individuals to groups of
#   increasing size without largely decreasing power?
gtPower(n = c(320, 160, 80, 64, 40, 32, 20, 10, 5),
        s = c(1, 2, 4, 5, 8, 10, 16, 32, 64),
        delta = 0.01,  p.hyp = 0.02)

# What happens to the power for increasing differences
#   between the true proportion and the threshold
#   proportion?
gtPower(n = 50, s = 10,
        delta = seq(from = 0, to = 0.01, by = 0.001),
        p.hyp = 0.01, method = "CP")

# Calculate power with a group size of 1 (individual
#   testing).
gtPower(n = 100, s = 1,
        delta = seq(from = 0, to = 0.01, by = 0.001),
        p.hyp = 0.01, method = "CP")
```

---

gtReg *Fitting group testing regression models*

Description

Fits the group testing regression model specified through a symbolic description of the linear predictor and descriptions of the group testing setting. This function allows for fitting regression models with simple pooling, halving, or array testing data.

Usage

```
gtReg(
  type = "sp",
  formula,
  data,
  groupn = NULL,
  subg = NULL,
  coln = NULL,
  rown = NULL,
  arrayn = NULL,
  retest = NULL,
  sens = 1,
  spec = 1,
  linkf = c("logit", "probit", "cloglog"),
  method = c("Vansteelandt", "Xie"),
  sens.ind = NULL,
  spec.ind = NULL,
  start = NULL,
  control = gtRegControl(...),
  ...
)
```

Arguments

`type`	`"sp"` for simple pooling (Dorfman testing with or without retests), `"halving"` for halving protocol, or `"array"` for array testing. See 'Details' for descriptions of the group testing algorithms.
`formula`	an object of class "formula" (or one that can be coerced to that class); a symbolic description of the model to be fitted. The details of model specification are under 'Details'.

`data`	an optional data frame, list, or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which `gtReg` is called.
`groupn`	a vector, list, or data frame of the group numbers that designates individuals to groups (for use with simple pooling, `type = "sp"`, or the halving protocol, `type = "halving"`).
`subg`	a vector, list, or data frame of the group numbers that designates individuals to subgroups (for use with the halving protocol, `type = "halving"`).
`coln`	a vector, list, or data frame that specifies the column group number for each sample (for use with array testing, `type = "array"`).
`rown`	a vector, list, or data frame that specifies the row group number for each sample (for use with array testing, `type = "array"`).
`arrayn`	a vector, list, or data frame that specifies the array number for each sample (for use with array testing, `type = "array"`).
`retest`	a vector, list, or data frame of individual retest results. Default value is `NULL` for no retests. See 'Details' for details on how to specify `retest`.
`sens`	sensitivity of the test. Default value is set to 1.
`spec`	specificity of the test. Default value is set to 1.
`linkf`	a character string specifying one of the three link functions for a binomial model: `"logit"` (default), `"probit"`, or `"cloglog"`.
`method`	the method to fit the regression model. Options include `"Vansteelandt"` (default) or `"Xie"`. The `"Vansteelandt"` option finds estimates by directly maximizing the likelihood function based on the group responses, while the `"Xie"` option uses the EM algorithm to maximize the likelihood function in terms of the unobserved individual responses.
`sens.ind`	sensitivity of the individual retests. If NULL, set to be equal to `sens`.
`spec.ind`	specificity of the individual retests. If NULL, set to be equal to `spec`.
`start`	starting values for the parameters in the linear predictor.
`control`	a list of parameters for controlling the fitting process in method `"Xie"`. These parameters will be passed to the `gtRegControl` function for use.
`...`	arguments to be passed to `gtRegControl` by default. See argument `control`.

Details

With simple pooling and halving, a typical predictor has the form `groupresp ~ covariates` where `groupresp` is the (numeric) group response vector. With array testing, individual samples are placed in a matrix-like grid where samples are pooled within each row and within each column. This leads to two kinds of group responses: row and column group responses. Thus, a typical predictor has the form `cbind(col.resp, row.resp) ~ covariates`, where `col.resp` is the (numeric) column group response vector and `row.resp` is the (numeric) row group response vector. For all methods, `covariates` is a series of terms which specifies a linear predictor for individual responses. Note that it is actually the unobserved individual responses, not the observed group responses, which are modeled by the covariates. When denoting group responses (`groupresp`, col.resp, and row.resp), a 0 denotes a negative response and a 1 denotes a positive response, where the probability of an individual positive response is being modeled directly.

A terms specification of the form first + second indicates all the terms in first together with all the terms in second with duplicates removed. A specification of the form first:second indicates the set of terms obtained by taking the interactions of all terms in first with all terms in second. The specification first*second indicates the cross of first and second. This is the same as first + second + first:second. The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order, and so on; to avoid this, pass a terms object as the formula.

For simple pooling (type = "sp"), the functions gtreg.fit, EM, and EM.ret, where the first corresponds to Vansteelandt's method described in Vansteelandt et al. (2000) and the last two correspond to Xie's method described in Xie (2001), are called to carry out the model fitting. The gtreg.fit function uses the optim function with default method "Nelder-Mead" to maximize the likelihood function of the observed group responses. If this optimization method produces a Hessian matrix of all zero elements, the "SANN" method in optim is employed to find the coefficients and Hessian matrix. For the "SANN" method, the number of iterations in optim is set to be 10000. For the background on the use of optim, see help(optim).

The EM and EM.ret functions apply Xie's EM algorithm to the likelihood function written in terms of the unobserved individual responses; the functions use glm.fit to update the parameter estimates within each M step. The EM function is used when there are no retests and EM.ret is used when individual retests are available. Thus, within the retest argument, individual observations in observed positive groups are 0 (negative) or 1 (positive); the remaining individual observations are NAs, meaning that no retest is performed for them. Retests cannot be used with Vansteelandt's method; a warning message will be given in this case, and the individual retests will be ignored in the model fitting. There could be slight differences in the estimates between Vansteelandt's and Xie's methods (when retests are not available) due to different convergence criteria.

With simple pooling (i.e., Dorfman testing, two-stage hierarchical testing), each individual appears in exactly one pool. When only the group responses are observed, the null degrees of freedom are the number of groups minus 1 and the residual degrees of freedom are the number of groups minus the number of parameters. When individual retests are observed too, it is an open research question for what the degrees of freedom and the deviance for the null model should be; therefore, the degrees of freedom and null.deviance will not be displayed.

Under the halving protocol, the EM.halving function applies Xie's EM algorithm to the likelihood function written in terms of the unobserved individual responses; the functions use glm.fit to update the parameter estimates within each M step. In the halving protocol, if the initial group tests positive, it is split into two subgroups. The two subgroups are subsequently tested and if either subgroup tests positive, the third and final step is to test all individuals within the subgroup. Thus, within subg, subgroup responses in observed positive groups are 0 (negative) or 1 (positive); the remaining subgroup responses are NAs, meaning that no tests are performed for them. The individual retests are similarly coded.

With array testing (also known as matrix pooling), the EM.mp function applies Xie's EM algorithm to the likelihood function written in terms of the unobserved individual responses. In each E step, the Gibbs sampling technique is used to estimate the conditional probabilities. Because of the large number of Gibbs samples needed to achieve convergence, the model fitting process could be quite slow, especially when multiple positive rows and columns are observed. In this case, we can either increase the Gibbs sample size to help achieve convergence or loosen the convergence criteria by increasing tol at the expense of perhaps poorer estimates. If follow-up retests are performed, the retest results going into the model will help achieve convergence faster with the same Gibbs sample size and convergence criteria. In each M step, we use `glm.fit` to update the parameter estimates.

For simple pooling, `retest` provides individual retest results for Dorfman's retesting procedure. Under the halving protocol, `retest` provides individual retest results within a subgroup that tests positive. The `retest` argument provides individual retest results, where a 0 denotes negative and 1 denotes positive status. An `NA` denotes that no retest is performed for that individual. The default value is `NULL` for no retests.

For simple pooling, `control` provides parameters for controlling the fitting process in the `"Xie"` method only.

`gtReg` returns an object of class `"gtReg"`. The function `summary` (i.e., `summary.gtReg` is used to obtain or print a summary of the results. The group testing function `predict` (i.e., `predict.gtReg`) is used to make predictions on `"gtReg"` objects.

Value

An object of class `"gtReg"`, a list which may include:

`coefficients`	a named vector of coefficients.
`hessian`	estimated Hessian matrix of the negative log-likelihood function. This serves as an estimate of the information matrix.
`residuals`	the response residuals. This is the difference of the observed group responses and the fitted group responses. Not included for array testing.
`fitted.values`	the fitted mean values of group responses. Not included for array testing.
`deviance`	the deviance between the fitted model and the saturated model. Not included for array testing.
`aic`	Akaike's Information Criterion. This is minus twice the maximized log-likelihood plus twice the number of coefficients. Not included for array testing.
`null.deviance`	the deviance for the null model, comparable with `deviance`. The null model will include only the intercept, if there is one in the model. Provided for simple pooling, `type = "sp"`, only.
`counts`	the number of iterations in `optim` (Vansteelandt's method) or the number of iterations in the EM algorithm (Xie's method, halving, and array testing).
`Gibbs.sample.size`	the number of Gibbs samples generated in each E step. Provided for array testing, `type = "array"`, only.
`df.residual`	the residual degrees of freedom. Provided for simple pooling, `type = "sp"`, only.
`df.null`	the residual degrees of freedom for the null model. Provided for simple pooling, `type = "sp"`, only.
`z`	the vector of group responses. Not included for array testing.
`call`	the matched call.
`formula`	the formula supplied.
`terms`	the terms object used.
`method`	the method (`"Vansteelandt"` or `"Xie"`) used to fit the model. For the halving protocol, the `"Xie"` method is used. Not included for array testing.
`link`	the link function used in the model.

Author(s)

The majority of this function was originally written as `gtreg.sp`, `gtreg.halving`, and `gtreg.mp` by Boan Zhang for the `binGroup` package. Minor modifications have been made for inclusion of the functions in the `binGroup2` package.

References

Vansteelandt, S., Goetghebeur, E., Verstraeten, T. (2000). “Regression models for disease prevalence with diagnostic tests on pools of serum samples.” *Biometrics*, **56**, 1126–1133.

Xie, M. (2001). “Regression analysis of group testing samples.” *Statistics in Medicine*, **20**, 1957–1969.

See Also

`gtSim` for simulation of data in the group testing form to be used by `gtReg`, `summary.gtReg` and `predict.gtReg` for `gtreg` methods.

Examples

```
data(hivsurv)
fit1 <- gtReg(type = "sp", formula  =  groupres ~ AGE + EDUC.,
              data  =  hivsurv, groupn  =  gnum, sens  =  0.9,
              spec  =  0.9, method  =  "Xie")
fit1

set.seed(46)
gt.data <- gtSim(type = "sp", par = c(-12, 0.2),
                 size1 = 700, size2 = 5)
fit2 <- gtReg(type = "sp", formula = gres ~ x, data = gt.data,
              groupn = groupn)
fit2

set.seed(21)
gt.data <- gtSim(type = "sp", par = c(-12, 0.2),
                 size1 = 700, size2 = 6, sens = 0.95, spec = 0.95,
                 sens.ind = 0.98, spec.ind = 0.98)
fit3 <- gtReg(type = "sp", formula = gres ~ x, data = gt.data,
              groupn = groupn, retest = retest, method = "Xie",
              sens = 0.95, spec = 0.95, sens.ind = 0.98,
              spec.ind = 0.98, trace = TRUE)
summary(fit3)

set.seed(46)
gt.data <- gtSim(type = "halving", par = c(-6, 0.1), gshape = 17,
                 gscale = 1.4, size1 = 5000, size2 = 5,
                 sens = 0.95, spec = 0.95)
fit4 <- gtReg(type = "halving", formula = gres ~ x,
              data = gt.data, groupn = groupn, subg = subgroup,
              retest = retest, sens = 0.95, spec = 0.95,
              start = c(-6, 0.1), trace = TRUE)
summary(fit4)
```

```
# 5x6 and 4x5 array
set.seed(9128)
sa1a <- gtSim(type = "array", par = c(-7, 0.1), size1 = c(5, 4),
              size2 = c(6, 5), sens = 0.95, spec = 0.95)
sa1 <- sa1a$dframe
fit5 <- gtReg(type = "array",
              formula = cbind(col.resp, row.resp) ~ x,
              data = sa1, coln = coln, rown = rown,
              arrayn = arrayn, sens = 0.95, spec = 0.95,
              tol = 0.005, n.gibbs = 2000, trace = TRUE)
fit5
summary(fit5)
```

gtRegControl *Auxiliary for controlling group testing regression*

Description

Auxiliary function to control fitting parameters of the EM algorithm used internally in gtReg for simple pooling (type = "sp") with method = "Xie" or for array testing (type = "array").

Usage

```
gtRegControl(
  tol = 1e-04,
  n.gibbs = 1000,
  n.burnin = 20,
  maxit = 500,
  trace = FALSE,
  time = TRUE
)
```

Arguments

tol	convergence criterion.
n.gibbs	the Gibbs sample size to be used in each E step of the EM algorithm, for array testing. The default is 1000.
n.burnin	the number of samples in the burn-in period, for array testing. The default is 20.
maxit	maximum number of iterations in the EM algorithm.
trace	a logical value indicating whether the output should be printed for each iteration. The default is FALSE.
time	a logical value indicating whether the length of time for the model fitting should be printed. The default is TRUE.

Value

A list with components named as the input arguments.

Author(s)

This function was originally written as the `gt.control` function for the binGroup package. Minor modifications have been made for inclusion in the binGroup2 package.

Examples

```
# The default settings:
gtRegControl()
```

`gtSim` *Simulation function for group testing data*

Description

Simulates data in group testing form ready to be fit by `gtReg`.

Usage

```
gtSim(
  type = "sp",
  x = NULL,
  gshape = 20,
  gscale = 2,
  par,
  linkf = c("logit", "probit", "cloglog"),
  size1,
  size2,
  sens = 1,
  spec = 1,
  sens.ind = NULL,
  spec.ind = NULL
)
```

Arguments

`type`	`"sp"` for simple pooling (Dorfman testing with or without retests), `"halving"` for halving protocol, and `"array"` for array testing (also known as matrix pooling).
`x`	a matrix of user-submitted covariates with which to simulate the data. Default is `NULL`, in which case a gamma distribution is used to generate the covariates automatically.
`gshape`	shape parameter for the gamma distribution. The value must be non-negative. Default value is set to 20.

`gscale`	scale parameter for the gamma distribution. The value must be strictly positive. Default value is set to 2.
`par`	the true coefficients in the linear predictor.
`linkf`	a character string specifying one of the three link functions to be used: `"logit"` (default), `"probit"`, or `"cloglog"`.
`size1`	sample size of the simulated data (for use with `"sp"` and `"halving"` methods) or a vector that specifies the number of rows in each matrix (for use with `"array"` method). If only one matrix is simulated, this value is a scalar.
`size2`	group size in pooling individual samples (for use with `"sp"` and `"halving"` methods) or a vector that specifies the number of columns in each matrix (for use with `"array"` method). If only one matrix is simulated, this value is a scalar.
`sens`	sensitivity of the group tests. Default value is set to 1.
`spec`	specificity of the group tests. Default value is set to 1.
`sens.ind`	sensitivity of the individual retests. If NULL, set to be equal to sens.
`spec.ind`	specificity of the individual retests. If NULL, set to be equal to spec.

Details

Generates group testing data in simple pooling form (`type = "sp"`), for the halving protocol (`type = "halving"`), or in array testing form (`type = "array"`). The covariates are either specified by the `x` argument or they are generated from a gamma distribution with the given `gshape` and `gscale` parameters. The individual probabilities are calculated from the covariates, the coefficients given in `par`, and the link function specified through `linkf`. The true binary individual responses are then simulated from the individual probabilities.

Under the matrix pooling protocol (`type = "array"`), the individuals are first organized into (by column) one or more matrices specified by the number of rows (`size1`) and the number of columns (`size2`).

Then, for all pooling protocols, the true group responses are found from the individual responses within groups or within rows/columns for matrix pooling (i.e., if at least one response is positive, the group is positive; otherwise, the group response is negative). Finally, the observed group (`method = "sp"`) and subgroup `method = "halving"` only), or row and column responses `method = "array"` are simulated using the given `sens` and `spec`.

For the simple pooling and halving protocols, individual retests are simulated from `sens.ind` and `spec.ind` for samples in observed positive groups. Note that with a given group size (specified by `size2` with `method = "sp"` or `method = "halving"`), the last group may have fewer individuals. For the matrix pooling protocol, individual retests are simulated from `sens.ind` and `spec.ind` for individuals that lie on the intersection of an observed positive row and and observed positive column. In the case where no column (row) tests positive in a matrix, all the individuals in any observed positive rows (columns) will be assigned a simulated retest result. If no column or row is observed positive, NULL is returned.

Value

For simple pooling (`type = "sp"`) and the halving protocol (`type = "halving"`), a data frame or for array testing (`type = "array"`), a list, which may include the following:

`gres`	the group response, for simple pooling and the halving protocol only.
`col.resp`	the column group response, for array testing only.
`row.resp`	the row group response, for array testing only.
`x`	the covariate.
`groupn`	the group number, for simple pooling and the halving protocol only.
`arrayn`	the array number, for array testing only.
`coln`	the column group number, for array testing only.
`rown`	the row group number, for array testing only.
`ind`	the true individual responses. For simple pooling and the halving protocol, these are included in the data frame of results. For array testing, these are included in the list of results, with individual responses presented in matrices.
`retest`	the results of individual retests.
`subgroup`	the subgroup number, for the halving protocol.
`prob`	the individual probabilities, for array testing only.

Author(s)

This function is a combination of `sim.gt`, `sim.halving`, and `sim.mp` written by Boan Zhang for the `binGroup` package. Minor modifications have been made for inclusion of the functions in the `binGroup2` package.

See Also

`gtReg` to fit simulated group testing data.

Examples

```
set.seed(46)
gt.data <- gtSim(type = "sp", par = c(-12, 0.2),
                 size1 = 700, size2 = 5)

x1 <- sort(runif(100, 0, 30))
x2 <- rgamma(100, shape = 17, scale = 1.5)
gt.data <- gtSim(type = "sp", x = cbind(x1, x2),
                 par = c(-14, 0.2, 0.3), size2 = 4,
                 sens = 0.98, spec = 0.98)

set.seed(46)
gt.data <- gtSim(type = "halving", par = c(-6, 0.1),
                 gshape = 17, gscale = 1.4, size1 = 5000,
                 size2 = 5, sens = 0.95, spec = 0.95)

# 5x6 and 4x5 matrix
set.seed(9128)
sa1a <- gtSim(type = "array", par = c(-7, 0.1),
              size1 = c(5, 4), size2 = c(6, 5),
              sens = 0.95, spec = 0.95)
sa1a$dframe
```

gtTest *Hypothesis test for one proportion in group testing*

Description

Calculates p-values for hypothesis tests of single proportions estimated from group testing experiments against a threshold proportion in the hypotheses. Available methods include the exact test, score test, and Wald test.

Usage

```
gtTest(n, y, s, p.hyp, alternative = "two.sided", method = "exact")
```

Arguments

`n`	integer specifying the number of groups.
`y`	integer specifying the number of positive groups.
`s`	integer specifying the common size of groups.
`p.hyp`	the hypothetical threshold proportion against which to test, specified as a number between 0 and 1.
`alternative`	character string defining the alternative hypothesis, either `"two.sided"`, `"less"`, or `"greater"`.
`method`	character string defining the test method to be used. Options include "exact" for an exact test corresponding to the Clopper-Pearson confidence interval, "score" for a score test corresponding to the Wilson confidence interval, and "Wald" for a Wald test corresponding to the Wald confidence interval. The Wald method is not recommended. The "exact" method uses `binom.test{stats}`.

Details

This function assumes equal group sizes, no testing error (i.e., 100 percent sensitivity and specificity) to test the groups, and individual units randomly assigned to the groups with identical true probability of success.

Value

A list containing:

`p.value`	the p-value of the test
`estimate`	the estimated proportion
`p.hyp`	the threshold proportion provided by the user.
`alternative`	the alternative provided by the user.
`method`	the test method provided by the user.

Author(s)

This function was originally written as `bgtTest` by Frank Schaarschmidt for the `binGroup` package. Minor modifications have been made for inclusion of the function in the `binGroup2` package.

See Also

`propCI` for confidence intervals in group testing and `binom.test(stats)` for the exact test and corresponding confidence interval.

Other estimation functions: `designEst()`, `designPower()`, `gtPower()`, `gtWidth()`, `propCI()`, `propDiffCI()`

Examples

```
# Consider the following the experiment: Tests are
#   performed on n=10 groups, each group has a size
#   of s=100 individuals. The aim is to show that less
#   than 0.5 percent (\eqn{p < 0.005}) of the units in
#   the population show a detrimental trait (positive test).
#   y=1 positive test and 9 negative tests are observed.
gtTest(n = 10, y = 1, s = 100, p.hyp = 0.005,
       alternative = "less", method = "exact")

# The exact test corresponds to the
#   limits of the Clopper-Pearson confidence interval
#   in the example of Tebbs & Bilder (2004):
gtTest(n = 24, y = 3, s = 7, alternative = "two.sided",
       method = "exact", p.hyp = 0.0543)

gtTest(n = 24, y = 3, s = 7, alternative = "two.sided",
       method = "exact", p.hyp = 0.0038)

# Hypothesis test with a group size of 1.
gtTest(n = 24, y = 3, s = 1, alternative = "two.sided",
       method = "exact", p.hyp = 0.1)

# Further methods:
gtTest(n = 24, y = 3, s = 7, alternative = "two.sided",
       method = "score", p.hyp = 0.0516)

gtTest(n = 24, y = 3, s = 7, alternative = "two.sided",
       method = "Wald", p.hyp = 0.0401)
```

`gtWidth` *Expected width of confidence intervals in group testing*

Description

Calculation of the expected value of the width of confidence intervals for one proportion in group testing. Calculations are available for the confidence interval methods in `propCI`.

Usage

```
gtWidth(n, s, p, conf.level = 0.95, alternative = "two.sided", method = "CP")
```

Arguments

`n`	integer specifying the number of groups. A vector of integers is also allowed.
`s`	integer specifying the common size of groups. A vector of integers is also allowed.
`p`	the assumed true proportion of individuals showing the trait to be estimated. A vector is also allowed.
`conf.level`	the required confidence level of the interval.
`alternative`	character string specifying the alternative hypothesis, either `"two.sided"`, `"less"`, or `"greater"`.
`method`	character string specifying the confidence interval method. Available options include those in `propCI`.

Details

The two-sided (`alternative="two.sided"`) option calculates the expected width between the lower and upper bound of a two-sided $conf.level * 100$ percent confidence interval. See Tebbs & Bilder (2004) for expression. The one-sided (`alternative="less"` or `alternative="greater"`) options calculate the expected distance between the one-sided limit and the assumed true proportion `p` for a one-sided $conf.level * 100$ percent confidence interval.

Value

A matrix containing the columns:

`ns`	the resulting total number of units, $n * s$.
`n`	the number of groups.
`s`	the group size.
`p`	the assumed true proportion.
`expCIWidth`	the expected value of the confidence interval width as defined under the argument `alternative`.

Author(s)

This function was originally written as `bgtWidth` by Frank Schaarschmidt for the `binGroup` package. Minor modifications have been made for inclusion of the function in the `binGroup2` package.

References

Tebbs, J., Bilder, C. (2004). “Confidence interval procedures for the probability of disease transmission in multiple-vector-transfer designs.” *Journal of Agricultural, Biological, and Environmental Statistics*, **9**, 75–90.

See Also

propCI for confidence intervals in group testing.

Other estimation functions: designEst(), designPower(), gtPower(), gtTest(), propCI(), propDiffCI()

Examples

```
# Examine different group sizes to determine
#   the shortest expected width.
gtWidth(n = 20, s = seq(from = 1, to = 200, by = 10),
        p = 0.01, alternative = "less", method = "CP")

# Calculate the expected width of the confidence
#   interval with a group size of 1 (individual testing).
gtWidth(n = 20, s = 1, p = 0.005, alternative = "less", method = "CP")
```

---

halving *Probability mass function for halving*

Description

Calculate the probability mass function for the number of tests from using the halving algorithm.

Usage

```
halving(p, Se = 1, Sp = 1, stages = 2, order.p = TRUE)
```

Arguments

p	a vector of individual risk probabilities.
Se	sensitivity of the diagnostic test.
Sp	specificity of the diagnostic test.
stages	the number of stages for the halving algorithm.
order.p	logical; if TRUE, the vector of individual risk probabilities will be sorted.

Details

Halving algorithms involve successively splitting a positive testing group into two equal-sized halves (or as close to equal as possible) until all individuals have been identified as positive or negative. S-stage halving begins by testing the whole group of I individuals. Positive groups are split in half until the final stage of the algorithm, which consists of individual testing. For example, consider an initial group of size $I = 16$ individuals. Three-stage halving (3H) begins by testing the whole group of 16 individuals. If this group tests positive, the second stage involves splitting into two groups of size 8. If either of these groups test positive, a third stage involves testing each individual rather than halving again. Four-stage halving (4H) would continue with halving into groups of size 4 before individual testing. Five-stage halving (5H) would continue with halving into groups of size 2 before individual testing. 3H requires more than 2 individuals, 4H requires more than 4 individuals, and 5H requires more than 8 individuals.

This function calculates the probability mass function, expected testing expenditure, and variance of the testing expenditure for halving algorithms with 3 to 5 stages.

Value

A list containing:

`pmf`	the probability mass function for the halving algorithm.
`et`	the expected testing expenditure for the halving algorithm.
`vt`	the variance of the testing expenditure for the halving algorithm.
`p`	a vector containing the probabilities of positivity for each individual.

Author(s)

This function was originally written by Michael Black for Black et al. (2012). The function was obtained from `http://chrisbilder.com/grouptesting/`. Minor modifications have been made for inclusion of the function in the `binGroup2` package.

References

Black, M., Bilder, C., Tebbs, J. (2012). “Group testing in heterogeneous populations by using halving algorithms.” *Journal of the Royal Statistical Society. Series C: Applied Statistics*, **61**, 277–290.

See Also

`expectOrderBeta` for generating a vector of individual risk probabilities for informative group testing.

Other operating characteristic functions: `GroupMembershipMatrix()`, `Sterrett()`, `TOD()`, `operatingCharacteristics1(`
`operatingCharacteristics2()`

Examples

```
# Equivalent to Dorfman testing (two-stage hierarchical)
halving(p = rep(0.01, 10), Se = 1, Sp = 1, stages = 2,
        order.p = TRUE)

# Halving over three stages; each individual has a
#   different probability of being positive
set.seed(12895)
p.vec <- expectOrderBeta(p = 0.05, alpha = 2, size = 20)
halving(p = p.vec, Se = 0.95, Sp = 0.95, stages = 3,
        order.p = TRUE)
```

hivsurv *Data from an HIV surveillance project*

Description

The `hivsurv` data set comes from an HIV surveillance project discussed in Verstraeten et al. (1998) and Vansteelandt et al. (2000). The purpose of the study was to estimate the HIV prevalence among pregnant Kenyan women in four rural locations of the country, using both individual and group testing responses. Blood tests were administered to each participating woman, and 4 covariates were obtained on each woman. Because the original group responses are unavailable, individuals are artificially put into groups of 5 here to form group responses. Only the 428 complete observations are given.

Usage

```
data(hivsurv)
```

Format

A data frame with 428 observations on the following 8 variables.

`DATE` the date when each sample was collected.

`PAR.` parity (number of children).

`AGE` age (in years).

`MA.ST.` marital status (1: single; 2: married (polygamous); 3: married (monogamous); 4: divorced; 5: widow).

`EDUC.` highest attained education level (1: no schooling; 2: primary school; 3: secondary school; 4: higher).

`HIV` individual response of HIV diagnosis (0: negative; 1: positive).

`gnum` the group number that designates individuals into groups.

`groupres` the group response calculated from artificially formed groups.

Source

Vansteelandt, S., Goetghebeur, E., Verstraeten, T. (2000). “Regression models for disease prevalence with diagnostic tests on pools of serum samples.” *Biometrics*, **56**, 1126–1133.

Verstraeten, T., Farah, B., Duchateau, L., Matu, R. (1998). “Pooling sera to reduce the cost of HIV surveillance: a feasibility study in a rural Kenyan district.” *Tropical Medicine & International Health*, **3**, 747–750.

Examples

```
data(hivsurv)

str(hivsurv)
```

IndProb *Extract the individual probabilities used to calculate group testing results*

Description

Extract the individual probabilities from objects of class "opchar" returned by `operatingCharacteristics1` (`opChar1`) or `operatingCharacteristics2` (`opChar2`).

Usage

```
IndProb(object, ...)
```

Arguments

`object`	An object of class "opChar", from which the individual probabilities are to be extracted.
`...`	Additional arguments to be passed to `IndProb` (e.g., `digits` to be passed to `signif` for appropriate rounding).

Value

Either `p.vec`, the sorted vector of individual probabilities (for hierarchical group testing algorithms) or `p.mat`, the sorted matrix of individual probabilities in gradient arrangement (for array testing algorithms). Further details are given under the 'Details' section for the `operatingCharacteristics1` (`opChar1`) or `operatingCharacteristics2` (`opChar2`) functions.

Author(s)

Brianna D. Hitt

Examples

```
config.mat <- matrix(data = c(rep(1, 10), 1:10),
                     nrow = 2, ncol = 10, byrow = TRUE)
res1 <- opChar1(algorithm = "D2", p = 0.05, Se = 0.99, Sp = 0.99,
        hier.config = config.mat)
IndProb(res1)

config.mat <- matrix(data = c(rep(1, 20), rep(1, 10), rep(2, 10),
                             rep(c(1, 2, 3, 4), each = 5),
                             rep(1, 3), rep(2, 2), rep(3, 3),
                             rep(4, 2), rep(5, 3), rep(6, 2),
                             rep(7, 3), rep(8, 2), 1:20),
                    nrow = 5, ncol = 20, byrow = TRUE)
Se <- matrix(data = rep(0.95, 10), nrow = 2, ncol = 5,
             dimnames = list(Infection = 1:2, Stage = 1:5))
Sp <- matrix(data = rep(0.99, 10), nrow = 2, ncol = 5,
             dimnames = list(Infection = 1:2, Stage = 1:5))
```

```
res2 <- opChar2(algorithm = "ID5",
                alpha = c(18.25, 0.75, 0.75, 0.25),
                Se = Se, Sp = Sp, hier.config = config.mat)
IndProb(res2)
```

`informativeArrayProb` *Arrange a matrix of probabilities for informative array testing*

Description

Arrange a vector of individual risk probabilities in a matrix for informative array testing without master pooling.

Usage

```
informativeArrayProb(prob.vec, nr, nc, method = "sd")
```

Arguments

`prob.vec`	vector of individual risk probabilities, of length nr * nc.
`nr`	number of rows in the array.
`nc`	number of columns in the array.
`method`	character string defining the method to be used for matrix arrangement. Options include spiral ("sd") and gradient ("gd") arrangement. See McMahan et al. (2012) for additional details.

Value

A matrix of probabilities arranged according to the specified method.

Author(s)

This function was originally written by Christopher McMahan for McMahan et al. (2012). The function was obtained from `http://chrisbilder.com/grouptesting/`.

References

McMahan, C., Tebbs, J., Bilder, C. (2012b). “Two-Dimensional Informative Array Testing.” *Biometrics*, **68**, 793–804.

See Also

`expectOrderBeta` for generating a vector of individual risk probabilities.

Examples

```
# Use the gradient arrangement method to create a matrix
#   of individual risk probabilities for a 10x10 array.
# Depending on the specified probability, alpha level,
#   and overall group size, simulation may be necessary
#   in order to generate the vector of individual
#   probabilities. This is done using the expectOrderBeta()
#   function and requires the user to set a seed in order
#   to reproduce results.
set.seed(1107)
p.vec1 <- expectOrderBeta(p = 0.05, alpha = 2, size = 100)
informativeArrayProb(prob.vec = p.vec1, nr = 10, nc = 10,
                     method = "gd")

# Use the spiral arrangement method to create a matrix
#   of individual risk probabilities for a 5x5 array.
set.seed(8791)
p.vec2 <- expectOrderBeta(p = 0.02, alpha = 0.5, size = 25)
informativeArrayProb(prob.vec = p.vec2, nr = 5, nc = 5,
                     method = "sd")
```

---

operatingCharacteristics1

Calculate operating characteristics for group testing algorithms that use a single-disease assay

---

Description

Calculate operating characteristics, such as the expected number of tests, for a specified testing configuration using non-informative and informative hierarchical and array-based group testing algorithms. Single-disease assays are used at each stage of the algorithms.

Usage

```
operatingCharacteristics1(
  algorithm,
  p = NULL,
  probabilities = NULL,
  Se = 0.99,
  Sp = 0.99,
  hier.config = NULL,
  rowcol.sz = NULL,
  alpha = 2,
  a = NULL,
  print.time = TRUE,
  ...
)
```

```
opChar1(
  algorithm,
  p = NULL,
  probabilities = NULL,
  Se = 0.99,
  Sp = 0.99,
  hier.config = NULL,
  rowcol.sz = NULL,
  alpha = 2,
  a = NULL,
  print.time = TRUE,
  ...
)
```

Arguments

`algorithm`	character string defining the group testing algorithm to be used. Non-informative testing options include two-stage hierarchical (`"D2"`), three-stage hierarchical (`"D3"`), four-stage hierarchical (`"D4"`), square array testing without master pooling (`"A2"`), and square array testing with master pooling (`"A2M"`). Informative testing options include two-stage hierarchical (`"ID2"`), three-stage hierarchical (`"ID3"`), four-stage hierarchical (`"ID4"`), and square array testing without master pooling (`"IA2"`).
`p`	overall probability of disease that will be used to generate a vector/matrix of individual probabilities. For non-informative algorithms, a homogeneous set of probabilities will be used. For informative algorithms, the `expectOrderBeta` function will be used to generate a heterogeneous set of probabilities. Further details are given under 'Details'. Either `p` or `probabilities` should be specified, but not both.
`probabilities`	a vector of individual probabilities, which is homogeneous for non-informative testing algorithms and heterogeneous for informative testing algorithms. Either `p` or `probabilities` should be specified, but not both.
`Se`	a vector of sensitivity values, where one value is given for each stage of testing (in order). If a single value is provided, sensitivity values are assumed to be equal to this value for all stages of testing. Further details are given under 'Details'.
`Sp`	a vector of specificity values, where one value is given for each stage of testing (in order). If a single value is provided, specificity values are assumed to be equal to this value for all stages of testing. Further details are given under 'Details'.
`hier.config`	a matrix specifying the configuration for a hierarchical testing algorithm. The rows correspond to the stages of testing, the columns correspond to each individual to be tested, and the cell values specify the group number of each individual at each stage. Further details are given under 'Details'. For array testing algorithms, this argument will be ignored.
`rowcol.sz`	the row/column size for array testing algorithms. For hierarchical testing algorithms, this argument will be ignored.

`alpha`	a shape parameter for the beta distribution that specifies the degree of heterogeneity for the generated probability vector (for informative testing only).
`a`	a vector containing indices indicating which individuals to calculate individual accuracy measures for. If `NULL`, individual accuracy measures will be displayed for all individuals in the algorithm.
`print.time`	a logical value indicating whether the length of time for calculations should be printed. The default is `TRUE`.
`...`	arguments to be passed to the `expectOrderBeta` function, which generates a vector of probabilities for informative testing algorithms. Further details are given under 'Details'.

Details

This function computes the operating characteristics for group testing algorithms with an assay that tests for one disease, as described in Hitt et al. (2019).

Available algorithms include two-, three-, and four-stage hierarchical testing and array testing with and without master pooling. Both non-informative and informative group testing settings are allowed for each algorithm, except informative array testing with master pooling is unavailable because this method has not appeared in the group testing literature. Operating characteristics calculated are expected number of tests, pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for each individual.

For informative algorithms where the `p` argument is specified, the expected value of order statistics from a beta distribution are found. These values are used to represent disease risk probabilities for each individual to be tested. The beta distribution has two parameters: a mean parameter `p` (overall disease prevalence) and a shape parameter `alpha` (heterogeneity level). Depending on the specified `p`, `alpha`, and overall group size, simulation may be necessary to generate the vector of individual probabilities. This is done using `expectOrderBeta` and requires the user to set a seed to reproduce results.

The sensitivity/specificity values are allowed to vary across stages of testing. For hierarchical testing, a different sensitivity/specificity value may be used for each stage of testing. For array testing, a different sensitivity/specificity value may be used for master pool testing (if included), row/column testing, and individual testing. The values must be specified in order of the testing performed. For example, values are specified as (stage 1, stage 2, stage 3) for three-stage hierarchical testing or (master pool testing, row/column testing, individual testing) for array testing with master pooling. A single sensitivity/specificity value may be specified instead. In this situation, sensitivity/specificity values for all stages are assumed to be equal.

The matrix specified by `hier.config` defines the hierarchical group testing algorithm for I individuals. The rows of the matrix correspond to the stages $s = 1, ..., S$ in the testing algorithm, and the columns correspond to individuals $i = 1, ...I$. The cell values within the matrix represent the group number of individual i at stage s. For three-stage, four-stage, and non-informative two-stage hierarchical testing, the first row of the matrix consists of all ones. This indicates that all individuals in the algorithm are tested together in a single group in the first stage of testing. For informative two-stage hierarchical testing, the initial group (block) is not tested. Thus, the first row of the matrix consists of the group numbers for each individual in the first stage of testing. For all hierarchical algorithms, the final row of the matrix denotes individual testing. Individuals who are not tested in a particular stage are represented by "NA" (e.g., an individual tested in a group of size 1 in the second stage of testing would not be tested again in a third stage of testing). It is important to note that this matrix represents the testing that could be performed if each group tests positively at each stage prior to the last. For more details on this matrix (called a group membership matrix), see Bilder et al. (2019).

For array testing without master pooling, the `rowcol.sz` specified represents the row/column size for initial (stage 1) testing. For array testing with master pooling, the `rowcol.sz` specified represents the row/column size for stage 2 testing. This is because the master pool size is the overall array size, given by the square of the row/column size.

The displayed overall pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value are weighted averages of the corresponding individual accuracy measures for all individuals within the initial group (or block) for a hierarchical algorithm, or within the entire array for an array-based algorithm. Expressions for these averages are provided in the Supplementary Material for Hitt et al. (2019). These expressions are based on accuracy definitions given by Altman and Bland (1994a, 1994b).

The `operatingCharacteristics1` function accepts additional arguments, namely `num.sim`, to be passed to the `expectOrderBeta` function, which generates a vector of probabilities for informative group testing algorithms. The `num.sim` argument specifies the number of simulations from the beta distribution when simulation is used. By default, 10,000 simulations are used.

Value

A list containing:

`algorithm`	the group testing algorithm used for calculations.
`prob`	the probability of disease or the vector of individual probabilities, as specified by the user.
`alpha`	level of heterogeneity for the generated probability vector (for informative testing only).
`Se`	the vector of sensitivity values for each stage of testing.
`Sp`	the vector of specificity values for each stage of testing.
`Config`	a list specifying elements of the specified testing configuration, which may include: **Stage1** group size for the first stage of hierarchical testing, if applicable. **Stage2** group sizes for the second stage of hierarchical testing, if applicable. **Stage3** group sizes for the third stage of hierarchical testing, if applicable. **Block.sz** the block size/initial group size for informative Dorfman testing, which is not tested. **pool.szs** group sizes for the first stage of testing for informative Dorfman testing. **Array.dim** the row/column size for array testing. **Array.sz** the overall array size for array testing (the square of the row/column size).
`p.vec`	the sorted vector of individual probabilities, if applicable.
`p.mat`	the sorted matrix of individual probabilities in gradient arrangement, if applicable. Further details are given under 'Details'.

`ET`	the expected testing expenditure to decode all individuals in the algorithm; this includes all individuals in all groups for hierarchical algorithms or in the entire array for array testing.
`value`	the value of the expected number of tests per individual.
`Accuracy`	a list containing:

Individual a matrix of accuracy measures for each individual specified in `a`. The rows correspond to each unique set of accuracy measures in the algorithm. Individuals with the same set of accuracy measures are displayed together in a single row of the matrix. The columns correspond to the pooling sensitivity, pooling specificity, pooling positive predictive value, pooling negative predictive value, and the indices for the individuals in each row of the matrix.

Overall a matrix of overall accuracy measures for the algorithm. The columns correspond to the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for the overall algorithm. Further details are given under 'Details'.

Note

This function returns the pooling positive and negative predictive values for all individuals even though these measures are diagnostic specific; e.g., the pooling positive predictive value should only be considered for those individuals who have tested positive.

Additionally, only stage dependent sensitivity and specificity values are allowed within the program (no group within stage dependent values are allowed). See Bilder et al. (2019) for additional information.

Author(s)

Brianna D. Hitt

References

Altman, D., Bland, J. (1994). "Diagnostic tests 1: Sensitivity and specificity." *BMJ*, **308**, 1552.

Altman, D., Bland, J. (1994). "Diagnostic tests 2: Predictive values." *BMJ*, **309**, 102.

Bilder, C., Tebbs, J., McMahan, C. (2019). "Informative group testing for multiplex assays." *Biometrics*, **75**, 278–288.

Hitt, B., Bilder, C., Tebbs, J., McMahan, C. (2019). "The objective function controversy for group testing: Much ado about nothing?" *Statistics in Medicine*, **38**, 4912–4923.

McMahan, C., Tebbs, J., Bilder, C. (2012a). "Informative Dorfman Screening." *Biometrics*, **68**, 287–296.

McMahan, C., Tebbs, J., Bilder, C. (2012b). "Two-Dimensional Informative Array Testing." *Biometrics*, **68**, 793–804.

See Also

Other operating characteristic functions: `GroupMembershipMatrix()`, `Sterrett()`, `TOD()`, `halving()`, `operatingCharacteristics2()`

Examples

```
# Calculate the operating characteristics for non-informative
#   two-stage hierarchical (Dorfman) testing.
config.mat <- matrix(data = c(rep(1, 10), 1:10),
                     nrow = 2, ncol = 10, byrow = TRUE)
opChar1(algorithm = "D2", p = 0.05, Se = 0.99, Sp = 0.99,
        hier.config = config.mat, print.time = FALSE)

# Calculate the operating characteristics for informative
#   two-stage hierarchical (Dorfman) testing.
# A vector of individual probabilities is generated using
#   the expected value of order statistics from a beta
#   distribution with p = 0.01 and a heterogeneity level
#   of alpha = 0.5.
config.mat <- matrix(data = c(rep(1:3, each = 10), 1:30),
                     nrow = 2, ncol = 30, byrow = TRUE)
set.seed(52613)
opChar1(algorithm = "ID2", p = 0.01, Se = 0.95, Sp = 0.95,
        hier.config = config.mat, alpha = 0.5, num.sim = 10000)
# Equivalent code using a heterogeneous vector of
#   probabilities
set.seed(52613)
probs <- expectOrderBeta(p = 0.01, alpha = 0.5, size = 30)
opChar1(algorithm = "ID2", probabilities = probs,
        Se = 0.95, Sp = 0.95, hier.config = config.mat)

# Calculate the operating characteristics for
#   non-informative three-stage hierarchical testing.
config.mat <- matrix(data = c(rep(1, 18), rep(1:3, each = 5),
                              rep(4, 3), 1:18),
                    nrow = 3, ncol = 18, byrow = TRUE)
opChar1(algorithm = "D3", p = 0.001, Se = 0.95, Sp = 0.95,
        hier.config = config.mat)
opChar1(algorithm = "D3", p = 0.001, Se = c(0.95, 0.95, 0.99),
        Sp = c(0.96, 0.96, 0.98), hier.config = config.mat)

# Calculate the operating characteristics for
#   informative three-stage hierarchical testing,
#   given a heterogeneous vector of probabilities.
config.mat <- matrix(data = c(rep(1, 6), rep(1:2, each = 3),
                              1:6), nrow = 3, ncol = 6,
                     byrow = TRUE)
set.seed(52613)
opChar1(algorithm = "ID3",
         probabilities = c(0.012, 0.014, 0.011, 0.012, 0.010, 0.015),
         Se = 0.99, Sp = 0.99, hier.config = config.mat,
         alpha = 0.5, num.sim = 5000)

# Calculate the operating characteristics for
#   non-informative four-stage hierarchical testing.
config.mat <- matrix(data = c(rep(1, 12), rep(1, 8),
                              rep(2, 2), 3, 4, rep(1, 5),
                              rep(2, 3), 3, 4, rep(NA, 2),
                              1:8, rep(NA, 4)), nrow = 4,
                     ncol = 12, byrow = TRUE)
opChar1(algorithm = "D4", p = 0.041, Se = 0.99, Sp = 0.90,
        hier.config = config.mat)

# Calculate the operating characteristics for
#   informative four-stage hierarchical testing.
# A vector of individual probabilities is generated using
#   the expected value of order statistics from a beta
#   distribution with p = 0.041 and a heterogeneity level
#   of alpha = 0.5.
config.mat <- matrix(data = c(rep(1, 12), rep(1, 8),
                              rep(2, 2), 3, 4, rep(1, 5),
                              rep(2, 3), 3, 4, rep(NA, 2),
                              1:8, rep(NA, 4)), nrow = 4,
                     ncol = 12, byrow = TRUE)
set.seed(5678)
opChar1(algorithm = "ID4", p = 0.041, Se = 0.99, Sp = 0.90,
        hier.config = config.mat, alpha = 0.5)

# Calculate the operating characteristics for
#   non-informative array testing without master pooling.
opChar1(algorithm = "A2", p = 0.005, Se = c(0.95, 0.99),
        Sp = c(0.95, 0.99), rowcol.sz = 8, a = 1)

# Calculate the operating characteristics for
#   informative array testing without master pooling.
# A vector of individual probabilities is generated using
#   the expected value of order statistics from a beta
#   distribution with p = 0.03 and a heterogeneity level
#   of alpha = 2.
set.seed(1002)
opChar1(algorithm = "IA2", p = 0.03, Se = 0.95, Sp = 0.95,
         rowcol.sz = 8, alpha = 2, a = 1:10)

# Calculate the operating characteristics for
#   non-informative array testing with master pooling.
opChar1(algorithm = "A2M", p = 0.02, Se = c(0.95,0.95,0.99),
        Sp = c(0.98,0.98,0.99), rowcol.sz = 5)
```

`operatingCharacteristics2`

Calculate operating characteristics for group testing algorithms that use a multiplex assay for two diseases

---

Description

Calculate operating characteristics, such as the expected number of tests, for a specified testing configuration using non-informative and informative hierarchical and array-based group testing algorithms. Multiplex assays for two diseases are used at each stage of the algorithms.

Usage

```
operatingCharacteristics2(
  algorithm,
  p.vec = NULL,
  probabilities = NULL,
  alpha = NULL,
  Se,
  Sp,
  hier.config = NULL,
  rowcol.sz = NULL,
  ordering = matrix(data = c(0, 1, 0, 1, 0, 0, 1, 1), nrow = 4, ncol = 2),
  a = NULL,
  print.time = TRUE,
  ...
)

opChar2(
  algorithm,
  p.vec = NULL,
  probabilities = NULL,
  alpha = NULL,
  Se,
  Sp,
  hier.config = NULL,
  rowcol.sz = NULL,
  ordering = matrix(data = c(0, 1, 0, 1, 0, 0, 1, 1), nrow = 4, ncol = 2),
  a = NULL,
  print.time = TRUE,
  ...
)
```

Arguments

`algorithm`	character string defining the group testing algorithm to be used. Non-informative testing options include two-stage hierarchical (`"D2"`), three-stage hierarchical (`"D3"`), four-stage hierarchical (`"D4"`), five-stage hierarchical (`"D5"`), square array testing without master pooling (`"A2"`), and square array testing with master pooling (`"A2M"`). Informative testing options include two-stage hierarchical (`"ID2"`), three-stage hierarchical (`"ID3"`), four-stage hierarchical (`"ID4"`), and five-stage hierarchical (`"ID5"`) testing.
`p.vec`	vector of overall joint probabilities. The joint probabilities are assumed to be equal for all individuals in the algorithm (non-informative testing only). There are four joint probabilities to consider: p_{00}, the probability that an individual tests negative for both diseases; p_{10}, the probability that an individual tests positive only for the first disease; p_{01}, the probability that an individual tests positive only for the second disease; and p_{11}, the probability that an individual tests positive for both diseases. The joint probabilities must sum to 1. Only one of `p.vec`, `probabilities`, or `alpha` should be specified.

`probabilities`	matrix of joint probabilities for each individual, where rows correspond to the four joint probabilities and columns correspond to each individual in the algorithm. Only one of `p.vec`, `probabilities`, or `alpha` should be specified.
`alpha`	a vector containing positive shape parameters of the Dirichlet distribution (for informative testing only). The vector will be used to generate a heterogeneous matrix of joint probabilities for each individual. The vector must have length 4. Further details are given under 'Details'. Only one of `p.vec`, `probabilities`, or `alpha` should be specified.
`Se`	matrix of sensitivity values, where one value is given for each disease (or infection) at each stage of testing. The rows of the matrix correspond to each disease $k = 1, 2$, and the columns of the matrix correspond to each stage of testing $s = 1, ..., S$. If a vector of 2 values is provided, the sensitivity values associated with disease are assumed to be equal to the kth value in the vector for all stages of testing. Further details are given under 'Details'.
`Sp`	a matrix of specificity values, where one value is given for each disease (or infection) at each stage of testing. The rows of the matrix correspond to each disease $k = 1, 2$, and the columns of the matrix correspond to each stage of testing $s = 1, ..., S$. If a vector of 2 values is provided, the specificity values associated with disease k are assumed to be equal to the kth value in the vector for all stages of testing. Further details are given under 'Details'.
`hier.config`	a matrix specifying the configuration for a hierarchical testing algorithm. The rows correspond to the stages of testing, the columns correspond to each individual to be tested, and the cell values specify the group number of each individual at each stage. Further details are given under 'Details'. For array testing algorithms, this argument will be ignored.
`rowcol.sz`	the row/column size for array testing algorithms. For hierarchical testing algorithms, this argument will be ignored.
`ordering`	a matrix detailing the ordering for the binary responses of the diseases. The columns of the matrix correspond to each disease and the rows of the matrix correspond to each of the 4 sets of binary responses for two diseases. This ordering is used with the joint probabilities. The default ordering is (p_00, p_10, p_01, p_11).
`a`	a vector containing indices indicating which individuals to calculate individual accuracy measures for. If `NULL`, individual accuracy measures will be displayed for all individuals in the algorithm.
`print.time`	a logical value indicating whether the length of time for calculations should be printed. The default is `TRUE`.
`...`	additional arguments to be passed to functions for hierarchical testing with multiplex assays for two diseases.

Details

This function computes the operating characteristics for standard group testing algorithms with a multiplex assay that tests for two diseases. Calculations for hierarchical group testing algorithms are performed as described in Bilder et al. (2019) and calculations for array-based group testing algorithms are performed as described in Hou et al. (2019).

Available algorithms include two-, three-, four-, and five-stage hierarchical testing and array testing with and without master pooling. Both non-informative and informative group testing settings are allowed for hierarchical algorithms. Only non-informative group testing settings are allowed for array testing algorithms. Operating characteristics calculated are expected number of tests, pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for each individual.

For informative algorithms where the `alpha` argument is specified, a heterogeneous matrix of joint probabilities for each individual is generated using the Dirichlet distribution. This is done using `rBeta2009::rdirichlet` and requires the user to set a seed to reproduce results. See Bilder et al. (2019) for additional details on the use of the Dirichlet distribution for this purpose.

The sensitivity/specificity values are allowed to vary across stages of testing. For hierarchical testing, a different sensitivity/specificity value may be used for each stage of testing. For array testing, a different sensitivity/specificity value may be used for master pool testing (if included), row/column testing, and individual testing. The values must be specified in the order of the testing performed. For example, values are specified as (stage 1, stage 2, stage 3) for three-stage hierarchical testing or (master pool testing, row/column testing, individual testing) for array testing with master pooling. A vector of 2 sensitivity/specificity values may be specified, and sensitivity/specificity values for all stages of testing are assumed to be equal. The first value in the vector will be used at each stage of testing for the first disease, and the second value in the vector will be used at each stage of testing for the second disease.

The matrix specified by `hier.config` defines the hierarchical group testing algorithm for I individuals. The rows of the matrix correspond to the stages $s = 1, ..., S$ in the testing algorithm, and the columns correspond to individuals $i = 1, ...I$. The cell values within the matrix represent the group number of individual i at stage s. For three-stage, four-stage, five-stage, and non-informative two-stage hierarchical testing, the first row of the matrix consists of all ones. This indicates that all individuals in the algorithm are tested together in a single group in the first stage of testing. For informative two-stage hierarchical testing, the initial group (block) is not tested. Thus, the first row of the matrix consists of the group numbers for each individual in the first stage of testing. For all hierarchical algorithms, the final row of the matrix denotes individual testing. Individuals who are not tested in a particular stage are represented by "NA" (e.g., an individual tested in a group of size 1 in the second stage of testing would not be tested again in a third stage of testing). It is important to note that this matrix represents the testing that could be performed if each group tests positively at each stage prior to the last. For more details on this matrix (called a group membership matrix), see Bilder et al. (2019).

For array testing without master pooling, the `rowcol.sz` specified represents the row/column size for initial (stage 1) testing. For array testing with master pooling, the `rowcol.sz` specified represents the row/column size for stage 2 testing. This is because the master pool size is the overall array size, given by the square of the row/column size.

The displayed overall pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value are weighted averages of the corresponding individual accuracy measures for all individuals within the initial group (or block) for a hierarchical algorithm, or within the entire array for an array-based algorithm. Expressions for these averages are provided in the Supplementary Material for Hitt et al. (2019). These expressions are based on accuracy definitions given by Altman and Bland (1994a, 1994b).

Value

A list containing:

`algorithm`	the group testing algorithm used for calculations.
`prob.vec`	the vector of joint probabilities provided by the user, if applicable (for non-informative algorithms only).
`joint.p`	the matrix of joint probabilities for each individual provided by the user, if applicable.
`alpha.vec`	the alpha vector provided by the user, if applicable (for informative algorithms only).
`Se`	the matrix of sensitivity values for each disease at each stage of testing.
`Sp`	the matrix of specificity values for each disease at each stage of testing.
`Config`	a list specifying elements of the specified testing configuration, which may include: **Stage1** group size for the first stage of hierarchical testing, if applicable. **Stage2** group sizes for the second stage of hierarchical testing, if applicable. **Stage3** group sizes for the third stage of hierarchical testing, if applicable. **Stage4** group sizes for the fourth stage of hierarchical testing, if applicable. **Block.sz** the block size/initial group size for informative Dorfman testing, which is not tested. **pool.szs** group sizes for the first stage of testing for informative Dorfman testing. **Array.dim** the row/column size for array testing. **Array.sz** the overall array size for array testing (the square of the row/column size).
`p.mat`	the matrix of joint probabilities for each individual in the algorithm. Each row corresponds to one of the four joint probabilities. Each column corresponds to an individual in the testing algorithm.
`ET`	the expected testing expenditure for the OTC.
`value`	the value of the expected number of tests per individual.
`Accuracy`	a list containing: **Disease 1 Individual** a matrix of accuracy measures, pertaining to the first disease, for each individual specified in `a`. The rows correspond to each unique set of accuracy measures in the algorithm. Individuals with the same set of accuracy measures are displayed together in a single row of the matrix. The columns correspond to the pooling sensitivity, pooling specificity, pooling positive predictive value, pooling negative predictive value, and the indices for the individuals in each row of the matrix. Individual accuracy measures are not displayed for array testing algorithms. **Disease 2 Individual** a matrix of accuracy measures, pertaining to the second disease, for each individual specified in `a`. The rows correspond to each unique set of accuracy measures in the algorithm. Individuals with the same set of accuracy measures are displayed together in a single row of the matrix. The columns correspond to the pooling sensitivity, pooling specificity,

pooling positive predictive value, pooling negative predictive value, and the indices for the individuals in each row of the matrix. Individual accuracy measures are not displayed for array testing algorithms.

Overall a matrix of overall accuracy measures for the algorithm. The rows correspond to each disease. The columns correspond to the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for the overall algorithm. Further details are given under 'Details'.

Note

This function returns the pooling positive and negative predictive values for all individuals even though these measures are diagnostic specific; e.g., the pooling positive predictive value should only be considered for those individuals who have tested positive.

Additionally, only stage dependent sensitivity and specificity values are allowed within the program (no group within stage dependent values are allowed). See Bilder et al. (2019) for additional information.

Author(s)

This function was written by Brianna D. Hitt. It calls `ET.all.stages.new` and `PSePSpAllStages`, which were originally written by Christopher Bilder for Bilder et al. (2019), and `ARRAY`, which was originally written by Peijie Hou for Hou et al. (2020). The functions `ET.all.stages.new`, `PSePSpAllStages`, and `ARRAY` were obtained from `http://chrisbilder.com/grouptesting/`. Minor modifications were made to the functions for inclusion in the binGroup2 package.

References

Altman, D., Bland, J. (1994). “Diagnostic tests 1: Sensitivity and specificity.” *BMJ*, **308**, 1552.

Altman, D., Bland, J. (1994). “Diagnostic tests 2: Predictive values.” *BMJ*, **309**, 102.

Bilder, C., Tebbs, J., McMahan, C. (2019). “Informative group testing for multiplex assays.” *Biometrics*, **75**, 278–288.

Hitt, B., Bilder, C., Tebbs, J., McMahan, C. (2019). “The objective function controversy for group testing: Much ado about nothing?” *Statistics in Medicine*, **38**, 4912–4923.

Hou, P., Tebbs, J., Wang, D., McMahan, C., Bilder, C. (2021). “Array testing with multiplex assays.” *Biostatistics*, **21**, 417–431.

McMahan, C., Tebbs, J., Bilder, C. (2012a). “Informative Dorfman Screening.” *Biometrics*, **68**, 287–296.

See Also

Other operating characteristic functions: `GroupMembershipMatrix()`, `Sterrett()`, `TOD()`, `halving()`, `operatingCharacteristics1()`

Examples

```
# Calculate the operating characteristics for
#   non-informative two-stage hierarchical
#   (Dorfman) testing.
config.mat <- matrix(data = c(rep(1, 24), 1:24),
                     nrow = 2, ncol = 24, byrow = TRUE)
Se <- matrix(data = c(0.95, 0.95, 0.95, 0.95),
             nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
Sp <- matrix(data = c(0.99, 0.99, 0.99, 0.99),
             nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
opChar2(algorithm = "D2", p.vec = c(0.90, 0.04, 0.04, 0.02),
         Se = Se, Sp = Sp, hier.config = config.mat, print.time = FALSE)

# Calculate the operating characteristics for informative
#   two-stage hierarchical (Dorfman) testing.
# A matrix of joint probabilities for each individual is
#   generated using the Dirichlet distribution.
config.mat <- matrix(data = c(rep(1, 5), rep(2, 4), 3, 1:9, NA),
                     nrow = 2, ncol = 10, byrow = TRUE)
Se <- matrix(data = c(0.95, 0.95, 0.99, 0.99),
             nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
Sp <- matrix(data = c(0.96, 0.96, 0.98, 0.98),
             nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
set.seed(8791)
opChar2(algorithm = "ID2", alpha = c(18.25, 0.75, 0.75, 0.25),
         Se = Se, Sp = Sp, hier.config = config.mat)
# Equivalent code using a heterogeneous matrix of joint
#   probabilities for each individual
set.seed(8791)
p.unordered <- t(rBeta2009::rdirichlet(n = 10,
                            shape = c(18.25, 0.75, 0.75, 0.25)))
p.ordered <- p.unordered[, order(1 - p.unordered[1,])]
opChar2(algorithm = "ID2", probabilities = p.ordered,
        Se = Se, Sp = Sp, hier.config = config.mat)

# Calculate the operating characteristics for
#   non-informative three-stage hierarchical testing.
config.mat <- matrix(data = c(rep(1, 10), rep(1, 5),
                              rep(2, 4), 3, 1:9, NA),
                     nrow = 3, ncol = 10, byrow = TRUE)
Se <- matrix(data = rep(0.95, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
Sp <- matrix(data = rep(0.99, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
opChar2(algorithm = "D3", p.vec = c(0.95, 0.02, 0.02, 0.01),
         Se = Se, Sp = Sp, hier.config = config.mat)
opChar2(algorithm = "D3", p.vec = c(0.95, 0.02, 0.02, 0.01),
        Se = Se, Sp = Sp, hier.config = config.mat,
        a = c(1, 6, 10))

# Calculate the operating characteristics for informative
#   three-stage hierarchical testing.
# A matrix of joint probabilities for each individual is
#   generated using the Dirichlet distribution.
config.mat <- matrix(data = c(rep(1, 15),
                              rep(c(1, 2, 3), each = 5), 1:15),
                     nrow = 3, ncol = 15, byrow = TRUE)
Se <- matrix(data = rep(0.95, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
Sp <- matrix(data = rep(0.99, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
opChar2(algorithm = "ID3", alpha = c(18.25, 0.75, 0.75, 0.25),
         Se = Se, Sp = Sp, hier.config = config.mat)

# Calculate the operating characteristics for
#   non-informative four-stage hierarchical testing.
config.mat <- matrix(data = c(rep(1, 12), rep(1, 6), rep(2, 6),
                              rep(1, 4), rep(2, 2), rep(3, 3),
                              rep(4, 3), 1:12),
                     nrow = 4, ncol = 12, byrow = TRUE)
Se <- matrix(data = rep(0.95, 8), nrow = 2, ncol = 4,
             dimnames = list(Infection = 1:2, Stage = 1:4))
Sp <- matrix(data = rep(0.99, 8), nrow = 2, ncol = 4,
             dimnames = list(Infection = 1:2, Stage = 1:4))
opChar2(algorithm = "D4", p.vec = c(0.92, 0.05, 0.02, 0.01),
         Se = Se, Sp = Sp, hier.config = config.mat)

# Calculate the operating characteristics for informative
#   five-stage hierarchical testing.
# A matrix of joint probabilities for each individual is
#   generated using the Dirichlet distribution.
config.mat <- matrix(data = c(rep(1, 20), rep(1, 10), rep(2, 10),
                              rep(c(1, 2, 3, 4), each = 5),
                              rep(1, 3), rep(2, 2), rep(3, 3),
                              rep(4, 2), rep(5, 3), rep(6, 2),
                              rep(7, 3), rep(8, 2), 1:20),
                     nrow = 5, ncol = 20, byrow = TRUE)
Se <- matrix(data = rep(0.95, 10), nrow = 2, ncol = 5,
             dimnames = list(Infection = 1:2, Stage = 1:5))
Sp <- matrix(data = rep(0.99, 10), nrow = 2, ncol = 5,
             dimnames = list(Infection = 1:2, Stage = 1:5))
opChar2(algorithm = "ID5", alpha = c(18.25, 0.75, 0.75, 0.25),
        Se = Se, Sp = Sp, hier.config = config.mat)

# Calculate the operating characteristics for
#   non-informative array testing without master pooling.
Se <- matrix(data = rep(0.95, 4), nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
Sp <- matrix(data = rep(0.99, 4), nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
opChar2(algorithm = "A2", p.vec = c(0.90, 0.04, 0.04, 0.02),
         Se = Se, Sp = Sp, rowcol.sz = 12)

# Calculate the operating characteristics for
#   non-informative array testing with master pooling.
Se <- matrix(data = rep(0.95, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
Sp <- matrix(data = rep(0.99, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
opChar2(algorithm = "A2M", p.vec = c(0.90, 0.04, 0.04, 0.02),
         Se = Se, Sp = Sp, rowcol.sz = 10)
```

`OTC1` *Find the optimal testing configuration for group testing algorithms that use a single-disease assay*

Description

Find the optimal testing configuration (OTC) using non-informative and informative hierarchical and array-based group testing algorithms. Single-disease assays are used at each stage of the algorithms.

Usage

```
OTC1(
  algorithm,
  p = NULL,
  probabilities = NULL,
  Se = 0.99,
  Sp = 0.99,
  group.sz,
  obj.fn = "ET",
  weights = NULL,
  alpha = 2,
  trace = TRUE,
  print.time = TRUE,
  ...
)
```

Arguments

`algorithm`	character string defining the group testing algorithm to be used. Non-informative testing options include two-stage hierarchical ("`D2`"), three-stage hierarchical ("`D3`"), square array testing without master pooling ("`A2`"), and square array testing with master pooling ("`A2M`"). Informative testing options include two-stage hierarchical ("`ID2`"), three-stage hierarchical ("`ID3`"), and square array testing without master pooling ("`IA2`").

`p`	overall probability of disease that will be used to generate a vector/matrix of individual probabilities. For non-informative algorithms, a homogeneous set of probabilities will be used. For informative algorithms, the `expectOrderBeta` function will be used to generate a heterogeneous set of probabilities. Further details are given under 'Details'. Either `p` or `probabilities` should be specified, but not both.
`probabilities`	a vector of individual probabilities, which is homogeneous for non-informative testing algorithms and heterogeneous for informative testing algorithms. Either `p` or `probabilities` should be specified, but not both.
`Se`	a vector of sensitivity values, where one value is given for each stage of testing (in order). If a single value is provided, sensitivity values are assumed to be equal to this value for all stages of testing. Further details are given under 'Details'.
`Sp`	a vector of specificity values, where one value is given for each stage of testing (in order). If a single value is provided, specificity values are assumed to be equal to this value for all stages of testing. Further details are given under 'Details'.
`group.sz`	a single group size or range of group sizes for which to calculate operating characteristics and/or find the OTC. The details of group size specification are given under 'Details'.
`obj.fn`	a list of objective functions which are minimized to find the OTC. The expected number of tests per individual, `"ET"`, will always be calculated. Additional options include `"MAR"` (the expected number of tests divided by the expected number of correct classifications, described in Malinovsky et al. (2016)), and `"GR"` (a linear combination of the expected number of tests, the number of misclassified negatives, and the number of misclassified positives, described in Graff & Roeloffs (1972)). See Hitt et al. (2019) for additional details. The first objective function specified in this list will be used to determine the results for the top configurations. Further details are given under 'Details'.
`weights`	a matrix of up to six sets of weights for the GR function. Each set of weights is specified by a row of the matrix.
`alpha`	a shape parameter for the betadistribution that specifies the degree of heterogeneity for the generated probability vector (for informative testing only).
`trace`	a logical value indicating whether the progress of calculations should be printed for each initial group size provided by the user. The default is `TRUE`.
`print.time`	a logical value indicating whether the length of time for calculations should be printed. The default is `TRUE`.
`...`	arguments to be passed to the `expectOrderBeta` function, which generates a vector of probabilities for informative testing algorithms. Further details are given under 'Details'.

Details

This function finds the OTC for group testing algorithms with an assay that tests for one disease and computes the associated operating characteristics, as described in Hitt et al. (2019).

Available algorithms include two- and three-stage hierarchical testing and array testing with and without master pooling. Both non-informative and informative group testing settings are allowed for each algorithm, except informative array testing with master pooling is unavailable because this method has not appeared in the group testing literature. Operating characteristics calculated are expected number of tests, pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for each individual.

For informative algorithms where the `p` argument is specified, the expected value of order statistics from a beta distribution are found. These values are used to represent disease risk probabilities for each individual to be tested. The beta distribution has two parameters: a mean parameter `p` (overall disease prevalence) and a shape parameter `alpha` (heterogeneity level). Depending on the specified `p`, `alpha`, and overall group size, simulation may be necessary to generate the vector of individual probabilities. This is done using `expectOrderBeta` and requires the user to set a seed to reproduce results.

Informative two-stage hierarchical (Dorfman) testing is implemented via the pool-specific optimal Dorfman (PSOD) method described in McMahan et al. (2012a), where the greedy algorithm proposed for PSOD is replaced by considering all possible testing configurations. Informative array testing is implemented via the gradient method (the most efficient array design), where higher-risk individuals are grouped in the left-most columns of the array. For additional details on the gradient arrangement method for informative array testing, see McMahan et al. (2012b).

The sensitivity/specificity values are allowed to vary across stages of testing. For hierarchical testing, a different sensitivity/specificity value may be used for each stage of testing. For array testing, a different sensitivity/specificity value may be used for master pool testing (if included), row/column testing, and individual testing. The values must be specified in order of the testing performed. For example, values are specified as (stage 1, stage 2, stage 3) for three-stage hierarchical testing or (master pool testing, row/column testing, individual testing) for array testing with master pooling. A single sensitivity/specificity value may be specified instead. In this situation, sensitivity/specificity values for all stages are assumed to be equal.

The value(s) specified by `group.sz` represent the initial (stage 1) group size for hierarchical testing and the row/column size for array testing. For informative two-stage hierarchical testing, the `group.sz` specified represents the block size used in the pool-specific optimal Dorfman (PSOD) method, where the initial group (block) is not tested. For more details on informative two-stage hierarchical testing implemented via the PSOD method, see Hitt et al. (2019) and McMahan et al. (2012a).

If a single value is provided for `group.sz` with array testing or non-informative two-stage hierarchical testing, operating characteristics will be calculated and no optimization will be performed. If a single value is provided for `group.sz` with three-stage hierarchical or informative two-stage hierarchical, the OTC will be found over all possible configurations. If a range of group sizes is specified, the OTC will be found over all group sizes.

In addition to the OTC, operating characteristics for some of the other configurations corresponding to each initial group size provided by the user will be displayed. These additional configurations are only determined for whichever objective function ("ET", "MAR", or "GR") is specified first in the function call. If "GR" is the objective function listed first, the first set of corresponding weights will be used. For algorithms where there is only one configuration for each initial group size (non-informative two-stage hierarchical and all array testing algorithms), results for each initial group size are provided. For algorithms where there is more than one possible configuration for each initial group size (informative two-stage hierarchical and all three-stage hierarchical algorithms), two sets of configurations are provided: 1) the best configuration for each initial group size, and 2) the top 10 configurations for each initial group size provided by the user. If a single value is provided for `group.sz` with array testing or non-informative two-stage hierarchical testing, operating characteristics will not be provided for configurations other than that specified by the user. Results are sorted by the value of the objective function per individual, `value`.

The displayed overall pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value are weighted averages of the corresponding individual accuracy measures for all individuals within the initial group (or block) for a hierarchical algorithm, or within the entire array for an array-based algorithm. Expressions for these averages are provided in the Supplementary Material for Hitt et al. (2019). These expressions are based on accuracy definitions given by Altman and Bland (1994a, 1994b). Individual accuracy measures can be calculated using the `operatingCharacteristics1` (`opChar1`) function.

The `OTC1` function accepts additional arguments, namely `num.sim`, to be passed to the `expectOrderBeta` function, which generates a vector of probabilities for informative group testing algorithms. The `num.sim` argument specifies the number of simulations from the beta distribution when simulation is used. By default, 10,000 simulations are used.

Value

A list containing:

`algorithm`	the group testing algorithm used for calculations.
`prob`	the probability of disease or the vector of individual probabilities, as specified by the user.
`alpha`	level of heterogeneity for the generated probability vector (for informative testing only).
`Se`	the vector of sensitivity values for each stage of testing.
`Sp`	the vector of specificity values for each stage of testing.

`opt.ET`, `opt.MAR`, `opt.GR`

a list of results for each objective function specified by the user, containing:

- **OTC** a list specifying elements of the optimal testing configuration, which may include:
  - **Stage1** group size for the first stage of hierarchical testing, if applicable.
  - **Stage2** group sizes for the second stage of hierarchical testing, if applicable.
  - **Block.sz** the block size/initial group size for informative Dorfman testing, which is not tested.
  - **pool.szs** group sizes for the first stage of testing for informative Dorfman testing.
  - **Array.dim** the row/column size for array testing.
  - **Array.sz** the overall array size for array testing (the square of the row/column size).
- **p.vec** the sorted vector of individual probabilities, if applicable.
- **p.mat** the sorted matrix of individual probabilities in gradient arrangement, if applicable. Further details are given under 'Details'.

ET the expected testing expenditure to decode all individuals in the algorithm; this includes all individuals in all groups for hierarchical algorithms or in the entire array for array testing.

value the value of the objective function per individual.

Accuracy a matrix of overall accuracy measures for the algorithm. The columns correspond to the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for the overall algorithm. Further details are given under 'Details'.

`Configs`	a data frame containing results for the best configuration for each initial group size provided by the user. The columns correspond to the initial group size, configuration (if applicable), overall array size (if applicable), expected number of tests, value of the objective function per individual, pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value. No results are displayed if a single `group.sz` is provided. Further details are given under 'Details'.
`Top.Configs`	a data frame containing results for the top overall configurations across all initial group sizes provided by the user. The columns correspond to the initial group size, configuration, expected number of tests, value of the objective function per individual, pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value. No results are displayed for non-informative two-stage hierarchical testing or for array testing algorithms. Further details are given under 'Details'.
`group.sz`	Initial group (or block) sizes examined to find the OTC.

Note

This function returns the pooling positive and negative predictive values for all individuals even though these measures are diagnostic specific; e.g., the pooling positive predictive value should only be considered for those individuals who have tested positive.

Additionally, only stage dependent sensitivity and specificity values are allowed within the program (no group within stage dependent values are allowed). See Bilder et al. (2019) for additional information.

Author(s)

Brianna D. Hitt

References

Altman, D., Bland, J. (1994). "Diagnostic tests 1: Sensitivity and specificity." *BMJ*, **308**, 1552.

Altman, D., Bland, J. (1994). "Diagnostic tests 2: Predictive values." *BMJ*, **309**, 102.

Bilder, C., Tebbs, J., McMahan, C. (2019). "Informative group testing for multiplex assays." *Biometrics*, **75**, 278–288.

Graff, L., Roeloffs, R. (1972). "Group testing in the presence of test error; an extension of the Dorfman procedure." *Technometrics*, **14**, 113–122.

Hitt, B., Bilder, C., Tebbs, J., McMahan, C. (2019). "The objective function controversy for group testing: Much ado about nothing?" *Statistics in Medicine*, **38**, 4912–4923.

Malinovsky, Y., Albert, P., Roy, A. (2016). "Reader reaction: A note on the evaluation of group testing algorithms in the presence of misclassification." *Biometrics*, **72**, 299–302.

McMahan, C., Tebbs, J., Bilder, C. (2012a). "Informative Dorfman Screening." *Biometrics*, **68**, 287–296.

McMahan, C., Tebbs, J., Bilder, C. (2012b). "Two-Dimensional Informative Array Testing." *Biometrics*, **68**, 793–804.

See Also

Other OTC functions: `OTC2()`

Examples

```
# Find the OTC for non-informative
#   two-stage hierarchical (Dorfman) testing.
OTC1(algorithm = "D2", p = 0.05, Se = 0.99, Sp = 0.99,
     group.sz = 2:100, obj.fn = "ET",
     trace = TRUE, print.time = TRUE)

# Find the OTC for informative two-stage hierarchical
#   (Dorfman) testing.
# A vector of individual probabilities is generated using
#   the expected value of order statistics from a beta
#   distribution with p = 0.01 and a heterogeneity level
#   of alpha = 0.5.
set.seed(52613)
OTC1(algorithm = "ID2", p = 0.01, Se = 0.95, Sp = 0.95,
     group.sz = 50, obj.fn = c("ET", "MAR", "GR"),
     weights = matrix(data = c(1, 1, 10, 10, 0.5, 0.5),
     nrow = 3, ncol = 2, byrow = TRUE), alpha = 0.5,
     trace = FALSE, print.time = TRUE, num.sim = 10000)

# Find the OTC over all possible testing configurations
#   for non-informative three-stage hierarchical testing
#   with a specified group size.
OTC1(algorithm = "D3", p = 0.001, Se = 0.95, Sp = 0.95,
     group.sz = 18, obj.fn = "ET",
     trace = FALSE, print.time = FALSE)

# Find the OTC for non-informative three-stage
#   hierarchical testing.
OTC1(algorithm = "D3", p = 0.06, Se = 0.90, Sp = 0.90,
     group.sz = 3:30, obj.fn = c("ET", "MAR", "GR"),
     weights = matrix(data = c(1, 1, 10, 10, 100, 100),
     nrow = 3, ncol = 2, byrow = TRUE))

# Find the OTC over all possible configurations
#   for informative three-stage hierarchical testing
#   with a specified group size and a heterogeneous
#   vector of probabilities.
set.seed(1234)
OTC1(algorithm = "ID3",
     probabilities = c(0.012, 0.014, 0.011,
                       0.012, 0.010, 0.015),
     Se = 0.99, Sp = 0.99, group.sz = 6,
     obj.fn = "ET",
     alpha = 0.5, num.sim = 5000, trace = FALSE)

# Calculate the operating characteristics for
#   non-informative array testing without master pooling
#   with a specified array size.
OTC1(algorithm = "A2", p = 0.005, Se = 0.95, Sp = 0.95,
     group.sz = 8, obj.fn = "ET", trace = FALSE)

# Find the OTC for informative array testing without
#   master pooling.
# A vector of individual probabilities is generated using
#   the expected value of order statistics from a beta
#   distribution with p = 0.03 and a heterogeneity level
#   of alpha = 2. The probabilities are then arranged in
#   a matrix using the gradient method.
set.seed(1002)
OTC1(algorithm = "IA2", p = 0.03, Se = 0.95, Sp = 0.95,
     group.sz = 2:20, obj.fn = c("ET", "MAR", "GR"),
     weights = matrix(data = c(1, 1, 10, 10, 100, 100),
                      nrow = 3, ncol = 2, byrow = TRUE),
     alpha = 2)

# Find the OTC for non-informative array testing
#   with master pooling. The calculations may not
#   be completed instantaneously.
OTC1(algorithm = "A2M", p = 0.04, Se = 0.90, Sp = 0.90,
     group.sz = 2:20, obj.fn = "ET")
```

OTC2 *Find the optimal testing configuration for group testing algorithms that use a multiplex assay for two diseases*

Description

Find the optimal testing configuration (OTC) using non-informative and informative hierarchical and array-based group testing algorithms. Multiplex assays for two diseases are used at each stage of the algorithms.

Usage

```
OTC2(
  algorithm,
  p.vec = NULL,
  probabilities = NULL,
  alpha = NULL,
  Se,
  Sp,
  ordering = matrix(data = c(0, 1, 0, 1, 0, 0, 1, 1), nrow = 4, ncol = 2),
  group.sz,
  trace = TRUE,
  print.time = TRUE,
  ...
)
```

Arguments

`algorithm`	character string defining the group testing algorithm to be used. Non-informative testing options include two-stage hierarchical (`"D2"`), three-stage hierarchical (`"D3"`), square array testing without master pooling (`"A2"`), and square array testing with master pooling (`"A2M"`). Informative testing options include two-stage hierarchical (`"ID2"`) and three-stage hierarchical (`"ID3"`) testing.
`p.vec`	vector of overall joint probabilities. The joint probabilities are assumed to be equal for all individuals in the algorithm (non-informative testing only). There are four joint probabilities to consider: p_{00}, the probability that an individual tests negative for both diseases; p_{10}, the probability that an individual tests positive only for the first disease; p_{01}, the probability that an individual tests positive only for the second disease; and p_{11}, the probability that an individual tests positive for both diseases. The joint probabilities must sum to 1. Only one of `p.vec`, `probabilities`, or `alpha` should be specified.
`probabilities`	matrix of joint probabilities for each individual, where rows correspond to the four joint probabilities and columns correspond to each individual in the algorithm. Only one of `p.vec`, `probabilities`, or `alpha` should be specified.
`alpha`	vector containing positive shape parameters of the Dirichlet distribution (for informative testing only). The vector will be used to generate a heterogeneous matrix of joint probabilities for each individual. The vector must have length 4. Further details are given under 'Details'. Only one of `p.vec`, `probabilities`, or `alpha` should be specified.
`Se`	matrix of sensitivity values, where one value is given for each disease (or infection) at each stage of testing. The rows of the matrix correspond to each disease $k = 1, 2$, and the columns of the matrix correspond to each stage of testing $s = 1, ..., S$. If a vector of 2 values is provided, the sensitivity values associated with disease k are assumed to be equal to the kth value in the vector for all stages of testing. Further details are given under 'Details'.
`Sp`	matrix of specificity values, where one value is given for each disease (or infection) at each stage of testing. The rows of the matrix correspond to each disease $k = 1, 2$, and the columns of the matrix correspond to each stage of testing $s = 1, ..., S$. If a vector of 2 values is provided, the specificity values associated with disease k are assumed to be equal to the kth value in the vector for all stages of testing. Further details are given under 'Details'.
`ordering`	matrix detailing the ordering for the binary responses of the diseases. The columns of the matrix correspond to each disease and the rows of the matrix

	correspond to each of the 4 sets of binary responses for two diseases. This ordering is used with the joint probabilities. The default ordering is (p_00, p_10, p_01, p_11).
`group.sz`	single group size or range of group sizes for which to calculate operating characteristics and/or find the OTC. The details of group size specification are given under 'Details'.
`trace`	a logical value indicating whether the progress of calculations should be printed for each initial group size provided by the user. The default is `TRUE`.
`print.time`	a logical value indicating whether the length of time for calculations should be printed. The default is `TRUE`.
`...`	additional arguments to be passed to functions for hierarchical testing with multiplex assays for two diseases.

Details

This function finds the OTC for standard group testing algorithms with a multiplex assay that tests for two diseases and computes the associated operating characteristics. Calculations for hierarchical group testing algorithms are performed as described in Bilder et al. (2019) and calculations for array-based group testing algorithms are performed as described in Hou et al. (2019).

Available algorithms include two- and three-stage hierarchical testing and array testing with and without master pooling. Both non-informative and informative group testing settings are allowed for hierarchical algorithms. Only non-informative group testing settings are allowed for array testing algorithms. Operating characteristics calculated are expected number of tests, pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for each individual.

For informative algorithms where the `alpha` argument is specified, a heterogeneous matrix of joint probabilities for each individual is generated using the Dirichlet distribution. This is done using `rBeta2009::rdirichlet` and requires the user to set a seed to reproduce results. See Bilder et al. (2019) for additional details on the use of the Dirichlet distribution for this purpose.

The sensitivity/specificity values are allowed to vary across stages of testing. For hierarchical testing, a different sensitivity/specificity value may be used for each stage of testing. For array testing, a different sensitivity/specificity value may be used for master pool testing (if included), row/column testing, and individual testing. The values must be specified in the order of the testing performed. For example, values are specified as (stage 1, stage 2, stage 3) for three-stage hierarchical testing or (master pool testing, row/column testing, individual testing) for array testing with master pooling. A vector of 2 sensitivity/specificity values may be specified, and sensitivity/specificity values for all stages of testing are assumed to be equal. The first value in the vector will be used at each stage of testing for the first disease, and the second value in the vector will be used at each stage of testing for the second disease.

The value(s) specified by `group.sz` represent the initial (stage 1) group size for hierarchical testing and the row/column size for array testing. If a single value is provided for `group.sz` with two-stage hierarchical or array testing, operating characteristics will be calculated and no optimization will be performed. If a single value is provided for `group.sz` with three-stage hierarchical, the OTC will be found over all possible configurations with this initial group size. If a range of group sizes is specified, the OTC will be found over all group sizes.

In addition to the OTC, operating characteristics for some of the other configurations corresponding to each initial group size provided by the user are displayed. For algorithms where there is only one configuration for each initial group size (non-informative two-stage hierarchical and all array testing algorithms), results for each initial group size are provided. For algorithms where there is more than one possible configuration for each initial group size (informative two-stage hierarchical and all three-stage hierarchical algorithms), two sets of configurations are provided: 1) the best configuration for each initial group size, and 2) the top 10 configurations for each initial group size provided by the user. If a single value is provided for `group.sz` with array testing or non-informative two-stage hierarchical testing, operating characteristics will not be provided for configurations other than that specified by the user. Results are sorted by the value of the objective function per individual, `value`.

The displayed overall pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value are weighted averages of the corresponding individual accuracy measures for all individuals within the initial group (or block) for a hierarchical algorithm, or within the entire array for an array-based algorithm. Expressions for these averages are provided in the Supplementary Material for Hitt et al. (2019). These expressions are based on accuracy definitions given by Altman and Bland (1994a, 1994b). Individual accuracy measures can be calculated using the `operatingCharacteristics2` (`opChar2`) function.

Value

A list containing:

`algorithm`	the group testing algorithm used for calculations.
`prob.vec`	the vector of joint probabilities provided by the user, if applicable (for non-informative algorithms only).
`joint.p`	the matrix of joint probabilities for each individual provided by the user, if applicable.
`alpha.vec`	the alpha vector provided by the user, if applicable (for informative algorithms only).
`Se`	the matrix of sensitivity values for each disease at each stage of testing.
`Sp`	the matrix of specificity values for each disease at each stage of testing.
`opt.ET`	a list containing:

OTC a list specifying elements of the optimal testing configuration, which may include:

- **Stage1** group size for the first stage of hierarchical testing, if applicable.
- **Stage2** group sizes for the second stage of hierarchical testing, if applicable.
- **Block.sz** the block size/initial group size for informative Dorfman testing, which is not tested.
- **pool.szs** group sizes for the first stage of testing for informative Dorfman testing.
- **Array.dim** the row/column size for array testing.
- **Array.sz** the overall array size for array testing (the square of the row/column size).

p.mat the matrix of joint probabilities for each individual in the algorithm. Each row corresponds to one of the four joint probabilities. Each column corresponds to an individual in the testing algorithm.

ET the expected testing expenditure for the OTC.

value the value of the expected number of tests per individual.

Accuracy the matrix of overall accuracy measures for the algorithm. The rows correspond to each disease. The columns correspond to the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for the overall algorithm. Further details are given under 'Details'.

`Configs`	a data frame containing results for the best configuration for each initial group size provided by the user. The columns correspond to the initial group size, configuration (if applicable), overall array size (if applicable), expected number of tests, value of the objective function per individual, and accuracy measures for each disease. Accuracy measures include the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value. No results are displayed if a single `group.sz` is provided. Further details are given under 'Details'.
`Top.Configs`	a data frame containing results for some of the top configurations for each initial group size provided by the user. The columns correspond to the initial group size, configuration, expected number of tests, value of the objective function per individual, and accuracy measures for each disease. Accuracy measures include the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value. No results are displayed for non-informative two-stage hierarchical testing or for array testing algorithms. Further details are given under 'Details'.
`group.sz`	Initial group (or block) sizes examined to find the OTC.

Note

This function returns the pooling positive and negative predictive values for all individuals even though these measures are diagnostic specific; e.g., the pooling positive predictive value should only be considered for those individuals who have tested positive.

Additionally, only stage dependent sensitivity and specificity values are allowed within the program (no group within stage dependent values are allowed). See Bilder et al. (2019) for additional information.

Author(s)

This function was written by Brianna D. Hitt. It calls `ET.all.stages.new` and `PSePSpAllStages`, which were originally written by Christopher Bilder for Bilder et al. (2019), and `ARRAY`, which was originally written by Peijie Hou for Hou et al. (2020). The functions `ET.all.stages.new`, `PSePSpAllStages`, and `ARRAY` were obtained from `http://chrisbilder.com/grouptesting/`. Minor modifications were made to the functions for inclusion in the binGroup2 package.

References

Altman, D., Bland, J. (1994). "Diagnostic tests 1: Sensitivity and specificity." *BMJ*, **308**, 1552.

Altman, D., Bland, J. (1994). “Diagnostic tests 2: Predictive values.” *BMJ*, **309**, 102.

Bilder, C., Tebbs, J., McMahan, C. (2019). “Informative group testing for multiplex assays.” *Biometrics*, **75**, 278–288.

Hitt, B., Bilder, C., Tebbs, J., McMahan, C. (2019). “The objective function controversy for group testing: Much ado about nothing?” *Statistics in Medicine*, **38**, 4912–4923.

Hou, P., Tebbs, J., Wang, D., McMahan, C., Bilder, C. (2021). “Array testing with multiplex assays.” *Biostatistics*, **21**, 417–431.

McMahan, C., Tebbs, J., Bilder, C. (2012a). “Informative Dorfman Screening.” *Biometrics*, **68**, 287–296.

See Also

Other OTC functions: `OTC1()`

Examples

```
# Find the OTC for non-informative two-stage
#   hierarchical (Dorfman) testing
Se <- matrix(data = c(0.95, 0.95, 0.99, 0.99), nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
Sp <- matrix(data = c(0.96, 0.96, 0.98, 0.98), nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
OTC2(algorithm = "D2", p.vec = c(0.90, 0.04, 0.04, 0.02),
     Se = Se, Sp = Sp, group.sz = 2:10)

# Find the OTC over all possible testing configurations
#   for informative two-stage hierarchical (Dorfman)
#   testing with a specified group size.
# A matrix of joint probabilities for each individual is
#   generated using the Dirichlet distribution.
Se <- matrix(data = rep(0.95, 4), nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
Sp <- matrix(data = rep(0.99, 4), nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
set.seed(1002)
OTC2(algorithm = "ID2", alpha = c(18.25, 0.75, 0.75, 0.25),
     Se = Se, Sp = Sp, group.sz = 18:22)

# Find the OTC for non-informative three-stage
#   hierarchical testing.
Se <- matrix(data = rep(0.95, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
Sp <- matrix(data = rep(0.99, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
OTC2(algorithm = "D3", p.vec = c(0.91, 0.04, 0.04, 0.01),
     Se = Se, Sp = Sp, group.sz = 3:12)

# Find the OTC over all possible configurations
#   for informative three-stage hierarchical
#   testing with a specified group size
#   and a heterogeneous matrix of joint
#   probabilities for each individual.
set.seed(8791)
Se <- matrix(data = rep(0.95, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
Sp <- matrix(data = rep(0.99, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
p.unordered <- t(rBeta2009::rdirichlet(n = 8,
                            shape = c(18.25, 0.75, 0.75, 0.25)))
p.ordered <- p.unordered[, order(1 - p.unordered[1,])]
OTC2(algorithm = "ID3", probabilities = p.ordered,
         Se = Se, Sp = Sp, group.sz = 8,
         trace = FALSE, print.time = FALSE)

# Find the OTC for non-informative array testing
#   without master pooling.
Se <- matrix(data = rep(0.95, 4), nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
Sp <- matrix(data = rep(0.99, 4), nrow = 2, ncol = 2,
             dimnames = list(Infection = 1:2, Stage = 1:2))
OTC2(algorithm = "A2", p.vec = c(0.90, 0.04, 0.04, 0.02),
     Se = Se, Sp = Sp, group.sz = 2:10)

# Find the OTC for non-informative array testing
#   with master pooling.
Se <- matrix(data = rep(0.95, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
Sp <- matrix(data = rep(0.99, 6), nrow = 2, ncol = 3,
             dimnames = list(Infection = 1:2, Stage = 1:3))
OTC2(algorithm = "A2M", p.vec = c(0.90, 0.04, 0.04, 0.02),
     Se = Se, Sp = Sp, group.sz = 10,
     trace = FALSE, print.time = FALSE)
```

plot.OTC *Plot method for optimal testing configuration results*

Description

Produce a plot for objects of class "OTC" returned by OTC1 or OTC2.

Usage

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

Arguments

x	an object of class "OTC", providing operating characteristics for the optimal testing configuration and similar configurations for a group testing algorithm.
...	currently not used.

Details

This function produces a plot for objects of class "OTC" returned by OTC1 or OTC2. It plots the expected number of tests per individual for each similar testing configuration in the object.

In addition to the OTC, the OTC1 and OTC2 functions provide operating characteristics for other configurations corresponding to each initial group size provided by the user. For algorithms where there is only one configuration for each initial group size (non-informative two-stage hierarchical and all array testing algorithms), results for each initial group size are plotted. For algorithms where there is more than one possible configuration for each initial group size (informative two-stage hierarchical and all three-stage hierarchical algorithms), the results corresponding to the best configuration for each initial group size are plotted.

If a single value is provided for the group.sz argument in the OTC1 or OTC2 functions, no plot will be produced.

The plot is produced using the ggplot2 package. Customization features from ggplot2 are available once the package is loaded. Examples are shown in the 'Examples' section.

Value

A plot of the expected number of tests per individual for similar configurations provided in the object.

Author(s)

Brianna D. Hitt

See Also

OTC1 and OTC2 for creating an object of class "OTC".

Examples

```
# Find the optimal testing configuration for
#   non-informative two-stage hierarchical testing.
res1 <- OTC1(algorithm = "D2", p = 0.01, Se = 0.99, Sp = 0.99,
             group.sz = 3:100, obj.fn = c("ET", "MAR", "GR1"),
             weights = matrix(data = c(1, 1), nrow = 1, ncol = 2))
plot(res1)

# Customize the plot using the ggplot2 package.
library(ggplot2)
plot(res1) + ylim(0,1) +
  ggtitle("Similar configurations for Dorfman testing") +
  theme(plot.title = element_text(hjust = 0.5))

# Find the optimal testing configuration for
#   informative three-stage hierarchical testing
res2 <- OTC1(algorithm = "ID3", p = 0.025,
             Se = c(0.95, 0.95, 0.99), Sp = c(0.96, 0.96, 0.98),
             group.sz = 3:15, obj.fn = "ET", alpha = 2)
plot(res2)
```

```
# Find the optimal testing configuration for
#   informative array testing without master pooling.
res3 <- OTC1(algorithm = "IA2", p = 0.09, alpha = 2,
             Se = 0.90, Sp = 0.90, group.sz = 3:20, obj.fn = "ET")
plot(res3)

# Find the optimal testing configuration for
#   informative two-stage hierarchical testing.
Se <- matrix(data = c(rep(0.95, 2), rep(0.99, 2)),
             nrow = 2, ncol = 2, byrow = FALSE)
Sp <- matrix(data = c(rep(0.96, 2), rep(0.98, 2)),
             nrow = 2, ncol = 2, byrow = FALSE)
res4 <- OTC2(algorithm = "ID2", alpha = c(18.25, 0.75, 0.75, 0.25),
                Se = Se, Sp = Sp, group.sz = 12:20)
plot(res4)


# Find the optimal testing configuration for
#   non-informative array testing with master pooling.
res5 <- OTC2(algorithm = "A2M", p.vec = c(0.90, 0.04, 0.04, 0.02),
             Se = rep(0.99, 2), Sp = rep(0.99, 2), group.sz = 3:20)
plot(res5)
```

pmf — *Access the testing probability mass function returned from an object*

Description

pmf is a generic function that extracts the probability mass function from an object (if available) that contains information aboout a testing configuration.

Usage

```
pmf(object, ...)
```

Arguments

object	An object from which the probability mass function is to be extracted.
...	Additional arguments to be passed to pmf.

Author(s)

Christopher R. Bilder

See Also

pmf.halving and pmf.Sterrett

Examples

```
res <- halving(p = rep(0.01, 10), Sp = 1, Se = 1,
               stages = 2, order.p = TRUE)
pmf.halving(res)
```

`pmf.halving` *Extract probability mass function (PMF) from group testing results*

Description

Extract the probability mass function from group testing results for the halving algorithm (objects of class "halving" returned by `halving`).

Usage

```
## S3 method for class 'halving'
pmf(object, ...)
```

Arguments

`object`	An object of class "halving", created by `halving`, from which the PMF is to be extracted.
`...`	currently not used.

Value

Data frame containing the probability mass function extracted from the object `object`.

Author(s)

Brianna D. Hitt

Examples

```
res <- halving(p = rep(0.01, 10), Sp = 1, Se = 1,
               stages = 2, order.p = TRUE)
pmf(res)
```

pmf.Sterrett — *Extract probability mass function (PMF) from group testing results*

Description

Extract the probability mass function from group testing results for the Sterrett algorithm (objects of class "Sterrett" returned by Sterrett).

Usage

```
## S3 method for class 'Sterrett'
pmf(object, ...)
```

Arguments

object	An object of class "Sterrett", created by Sterrett, from which the PMF is to be extracted.
...	currently not used.

Value

Data frame containing the probability mass function extracted from the object `object`.

Author(s)

Brianna D. Hitt

Examples

```
set.seed(1231)
p.vec <- rbeta(n = 8, shape1 = 1, shape2 = 10)
res <- Sterrett(p = p.vec, Sp = 0.90, Se = 0.95)
pmf(res)
```

predict.gtReg — *Predict method for* `gtReg`

Description

Obtains predictions for individual observations and optionally computes the standard errors of those predictions from objects of class "gtReg" returned by gtReg.

Usage

```
## S3 method for class 'gtReg'
predict(
  object,
  newdata,
  type = c("link", "response"),
  se.fit = FALSE,
  conf.level = NULL,
  na.action = na.pass,
  ...
)
```

Arguments

`object`	a fitted object of class `"gtReg"`.
`newdata`	an optional data frame in which to look for variables with which to predict. If omitted, the fitted linear predictors are used.
`type`	the type of prediction required. The `"link"` option is on the scale of the linear predictors. The `"response"` option is on the scale of the response variable. Thus, for the logit model, the `"link"` predictions are of log-odds (probabilities on the logit scale) and `type = "response"` gives the predicted probabilities.
`se.fit`	a logical value indicating whether standard errors are required.
`conf.level`	the confidence level of the interval for the predicted values.
`na.action`	a function determining what should be done with missing values in `newdata`. The default is to predict `NA`.
`...`	currently not used.

Details

If `newdata` is omitted, the predictions are based on the data used for the fit. When `newdata` is present and contains missing values, how the missing values will be dealt with is determined by the `na.action` argument. In this case, if `na.action = na.omit`, omitted cases will not appear, whereas if `na.action = na.exclude`, omitted cases will appear (in predictions and standard errors) with value `NA`.

Value

If `se = FALSE`, a vector or matrix of predictions. If `se = TRUE`, a list containing:

`fit`	predictions.
`se.fit`	estimated standard errors.
`lower`	the lower bound of the confidence interval, if calculated (i.e., `conf.level` is specified).
`upper`	the upper bound of the confidence interval, if calculated (i.e., `conf.level` is specified).

Author(s)

Boan Zhang

Examples

```
data(hivsurv)
fit1 <- gtReg(formula = groupres ~ AGE + EDUC.,
              data = hivsurv, groupn = gnum,
              sens = 0.9, spec = 0.9,
              linkf = "logit", method = "V")
pred.data <- data.frame(AGE = c(15, 25, 30),
                        EDUC. = c(1, 3, 2))
predict(object = fit1, newdata = pred.data,
        type = "link", se.fit = TRUE)
predict(object = fit1, newdata = pred.data,
        type = "response", se.fit = TRUE,
        conf.level = 0.9)
predict(object = fit1, type = "response",
        se.fit = TRUE, conf.level = 0.9)
```

---

print.designEst *Print method for objects of class "designEst"*

Description

Print method for objects of class "designEst" created by `designEst`.

Usage

```
## S3 method for class 'designEst'
print(x, ...)
```

Arguments

`x`	an object of class "designEst" created by `designEst`.
`...`	additional arguments to be passed to `print`. Currently only `digits` to be passed to `signif` for appropriate rounding.

Value

A print out detailing whether the bias restriction was violated, whether the maximum allowed group size was reached, and the minimum MSE and associated group size, expected value, variance, and bias.

Author(s)

Brianna D. Hitt

print.designPower — *Print method for objects of class "designPower"*

Description

Print method for objects of class "designPower" created by designPower.

Usage

```
## S3 method for class 'designPower'
print(x, ...)
```

Arguments

x	an object of class "designPower" created by designPower.
...	additional arguments to be passed to `print`. Currently only `digits` to be passed to `signif` for appropriate rounding.

Value

A print out detailing whether or not power was reached in the range of values (`n` or `s`) provided, the maximal power reached in the range of values, the alternative hypothesis, and the assumed true proportion.

Author(s)

This function was originally written as `print.bgtDesign` by Frank Schaarschmidt for the `binGroup` package. Minor modifications were made for inclusion in the `binGroup2` package.

print.gtReg — *Print method for* `gtReg`

Description

Print method for objects obtained by gtReg.

Usage

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

Arguments

x	An object of class "gtReg" created by gtReg.
digits	digits for rounding.
...	currently not used.

Value

A print out of the function call, coefficients, and the null and residual deviance and degrees of freedom.

Author(s)

This function was originally written by Boan Zhang as the `print.gt` function for the `binGroup` package. Minor modifications were made for inclusion in the `binGroup2` package.

`print.gtTest` *Print method for objects of class "gtTest"*

Description

Print method for objects of class "gtTest" created by the `gtTest` function.

Usage

```
## S3 method for class 'gtTest'
print(x, ...)
```

Arguments

`x`	An object of class "gtTest" (`gtTest`).
`...`	Additional arguments to be passed to `print`. Currently only `digits` to be passed to `signif` for appropriate rounding.

Value

A print out of the p-value and point estimate resulting from `gtTest`.

Author(s)

This function was originally written as `print.bgtTest` by Brad Biggerstaff for the `binGroup` package. Minor modifications were made for inclusion of the function in the `binGroup2` package.

print.halving — *Print method for objects of class "halving"*

Description

Print method for objects of class "halving" created by the halving function.

Usage

```
## S3 method for class 'halving'
print(x, ...)
```

Arguments

x	An object of class "halving" (halving).
...	Additional arguments to be passed to `print`. Currently only `digits` to be passed to `signif` for appropriate rounding.

Value

A print out of the PMF, expected testing expenditure and variance of testing expenditure resulting from halving.

Author(s)

Brianna D. Hitt

print.opChar — *Print method for operating characteristics results*

Description

Print method for objects of class `"opChar"` returned by operatingCharacteristics1 (opChar1) or operatingCharacteristics2 (opChar2).

Usage

```
## S3 method for class 'opChar'
print(x, ...)
```

Arguments

x	an object of class `"opChar"`, providing the calculated operating characteristics for a group testing algorithm.
...	Additional arguments to be passed to `print` (e.g., `digits` to be passed to `round` for appropriate rounding).

Value

A print out of the algorithm, testing configuration, expected number of tests, expected number of tests per individual, and accuracy measures for individuals and for the overall algorithm.

Author(s)

Brianna D. Hitt

`print.OTC` *Print method for optimal testing configuration results*

Description

Print method for objects of class "OTC" returned by OTC1 or OTC2.

Usage

```
## S3 method for class 'OTC'
print(x, ...)
```

Arguments

x	an object of class "OTC", providing the optimal testing configuration results for a group testing algorithm.
...	Additional arguments to be passed to print (e.g., digits to be passed to round for appropriate rounding).

Value

A print out of the algorithm, testing configuration, expected number of tests, expected number of tests per individual, and accuracy measures for individuals and for the overall algorithm.

Author(s)

Brianna D. Hitt

`print.predict.gtReg` *Print method for* `predict.gtReg`

Description

Print method for objects obtained by `predict.gtReg`.

Usage

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

Arguments

`x`	An object of class "predict.gtReg" created by `predict.gtReg`.
`digits`	digits for rounding.
`...`	not currently used.

Value

A matrix of predictions with rows corresponding to each observation in `newdata` (if provided) or each observation in the data set used for the fit. The columns correspond to the predictions (`fit`), the estimated standard errors (`se.fit`), the lower bound of the confidence interval (`lower`), and the upper bound of the confidence interval (`upper`). If `conf.level` is not specified, the `lower` and `upper` columns will not be included. If `se = FALSE`, the `se.fit` column will not be included.

Author(s)

Brianna Hitt

`print.propCI` *Print method for objects of class "propCI"*

Description

Print method for objects of class "propCI" created by the `propCI` function.

Usage

```
## S3 method for class 'propCI'
print(x, ...)
```

Arguments

`x`	An object of class "propCI" (`propCI`).
`...`	Additional arguments to be passed to `print`.

Value

A print out of the point estimate and confidence interval found with `propCI`.

Author(s)

This function is a combination of `print.poolbindiff` and `print.bgt`, written by Brad Biggerstaff for the `binGroup` package. Minor modifications were made for inclusion of the function in the `binGroup2` package.

`print.propDiffCI` *Print method for objects of class "propDiffCI"*

Description

Print method for objects of class "propDiffCI" created by the `propDiffCI` function.

Usage

```
## S3 method for class 'propDiffCI'
print(x, ...)
```

Arguments

`x`	An object of class "propDiffCI" (`propDiffCI`).
`...`	Additional arguments to be passed to `print`.

Value

A print out of the point estimate and confidence interval found with `propDiffCI`.

Author(s)

This function was originally written as `print.poolbindiff` by Brad Biggerstaff for the `binGroup` package. Minor modifications were made for inclusion of the function in the `binGroup2` package.

`print.Sterrett` *Print method for objects of class "Sterrett"*

Description

Print method for objects of class "Sterrett" created by the `Sterrett` function.

Usage

```
## S3 method for class 'Sterrett'
print(x, ...)
```

Arguments

`x`	An object of class "Sterrett" (`Sterrett`).
`...`	Additional arguments to be passed to `print`. Currently only `digits` to be passed to `signif` for appropriate rounding.

Value

A print out of the PMF, expected testing expenditure and variance of testing expenditure resulting from `Sterrett`.

Author(s)

Brianna D. Hitt

`print.summary.gtReg` *Print method for* `summary.gtReg`

Description

Print method for objects obtained by `summary.gtReg`.

Usage

```
## S3 method for class 'summary.gtReg'
print(
  x,
  digits = max(3, getOption("digits") - 3),
  signif.stars = getOption("show.signif.stars"),
  ...
)
```

Arguments

`x`	An object of class "summary.gtReg" created by `summary.gtReg`.
`digits`	digits for rounding.
`signif.stars`	a logical value indicating whether significance stars should be shown.
`...`	Additional arguments to be passed to `printCoefmat`.

Value

A print out of the function call, deviance residuals (for simple pooling and halving only), coefficients, null and residual deviance and degrees of freedom (for simple pooling only), AIC (for simple pooling and halving only), number of Gibbs samples (for array testing only), and the number of iterations.

Author(s)

This function combines code from `print.summary.gt` and `print.summary.gt.mp`, written by Boan Zhang for the `binGroup` package. Minor modifications were made for inclusion in the `binGroup2` package.

`print.TOD` *Print method for* `TOD`

Description

Print method for objects obtained by `TOD`.

Usage

```
## S3 method for class 'TOD'
print(x, ...)
```

Arguments

`x`	An object of class "TOD" created by `TOD`.
`...`	not currently used.

Value

A print out of configuration and operating characteristics found with `TOD`.

Author(s)

Chris Bilder

propCI *Confidence intervals for one proportion in group testing*

Description

Calculates point estimates and confidence intervals for a single proportion with group testing data. Methods are available for groups of equal or different sizes.

Usage

```
propCI(
  x,
  m,
  n,
  pt.method = "mle",
  ci.method,
  conf.level = 0.95,
  alternative = "two.sided",
  maxiter = 100,
  tol = .Machine$double.eps^0.5
)
```

Arguments

`x`	integer specifying the number of positive groups when groups are of equal size, or a vector specifying the number of positive groups among the `n` groups tested when group sizes differ. If the latter, this vector must be of the same length as the `m` and `n` arguments.
`m`	integer specifying the common size of groups when groups are of equal size, or a vector specifying the group sizes when group sizes differ. If the latter, this vector must be of the same length as the `x` and `n` arguments.
`n`	integer specifying the number of groups when these groups are of equal size, or a vector specifying the corresponding number of groups of the sizes `m` when group sizes differ. If the latter, this vector must be of the same length as the `x` and `m` arguments.
`pt.method`	character string specifying the point estimate to compute. Options include `"Firth"` for the bias-preventative, `"Gart"` and `"bc-mle"` for the bias-corrected MLE (where the latter allows for backward compatibility), and `"mle"` for the MLE.
`ci.method`	character string specifying the confidence interval to compute. Options include "AC" for the Agresti-Coull interval, "bc-skew-score" for the bias- and skewness-corrected interval, "Blaker" for the Blaker interval, "CP" for the Clopper-Pearson interval, "exact" for the exact interval as given by Hepworth (1996), "lrt" for the likelihood ratio test interval, "score" for the Wilson score interval, "skew-score" for the skewness-corrected interval, "soc" for the second-order corrected interval, and "Wald" for the Wald interval. Note that the Agresti-Coull, Blaker, Clopper-Pearson, and second-order corrected intervals can only be calculated when `x`, `m`, and `n` are given as integers (equal group size case).

`conf.level`	confidence level of the interval.
`alternative`	character string defining the alternative hypothesis, either `"two.sided"`, `"less"`, or `"greater"`.
`maxiter`	the maximum number of steps in the iteration of confidence limits, for use only with the `"exact"` method when group sizes differ.
`tol`	the accuracy required for iterations in internal functions, for use with asymptotic intervals when group sizes differ only.

Details

Confidence interval methods include the Agresti-Coull (`ci.method = "AC"`), bias- and skewness-corrected (`ci.method = "bc-skew-score"`), Blaker (`ci.method = "Blaker"`), Clopper-Pearson (`ci.method = "CP"`), exact (`ci.method = "exact"`), likelihood ratio test (`ci.method = "lrt"`), Wilson score (`ci.method = "score"`), skewness-corrected (`ci.method = "skew-score"`), second-order corrected (`ci.method = "soc"`), and Wald (`ci.method = "Wald"`) intervals. The Agresti-Coull, Blaker, Clopper-Pearson, and second-order corrected intervals are available only for the equal group size case.

Point estimates available include the MLE (`pt.method = "mle"`), bias-corrected MLE (`pt.method = "Gart"` or `pt.method = "bc-mle"`), and bias-preventative (`pt.method = "Firth"`). Only the MLE method is available when calculating the Clopper-Pearson, Blaker, Agresti-Coull, second-order corrected, or exact intervals.

Equal group sizes: Computation of confidence intervals for group testing with equal group sizes are described in Tebbs & Bilder (2004) and Schaarschmidt (2007).

Unequal group sizes: While the exact method is available when group sizes differ, the algorithm becomes computationally very expensive if the number of different groups, `n`, becomes larger than three. See Hepworth (1996) for additional details on the exact method and other methods for constructing confidence intervals in group testing situations. For computational details and simulation results of the remaining methods, see Biggerstaff (2008). See Hepworth & Biggerstaff (2017) for recommendations on the best point estimator methods.

Value

A list containing:

`conf.int`	a confidence interval for the proportion.
`estimate`	the point estimator of the proportion.
`pt.method`	the method used for point estimation.
`ci.method`	the method used for confidence interval estimation.
`conf.level`	the confidence level of the interval.
`alternative`	the alternative specified by the user.
`x`	the number of positive groups.
`m`	the group sizes.
`n`	the numbers of groups with corresponding group sizes `m`.

Author(s)

This function is a combination of `bgtCI` and `bgtvs` written by Frank Schaarschmidt and `pooledBin` written by Brad Biggerstaff for the `binGroup` package. Minor modifications have been made for inclusion of the functions in the `binGroup2` package.

References

Biggerstaff, B. (2008). "Confidence intervals for the difference of proportions estimated from pooled samples." *Journal of Agricultural, Biological, and Environmental Statistics*, **13**, 478–496.

Hepworth, G. (1996). "Exact confidence intervals for proportions estimated by group testing." *Biometrics*, **52**, 1134–1146.

Hepworth, G., Biggerstaff, B. (2017). "Bias correction in estimating proportions by pooled testing." *Journal of Agricultural, Biological, and Environmental Statistics*, **22**, 602–614.

Schaarschmidt, F. (2007). "Experimental design for one-sided confidence intervals or hypothesis tests in binomial group testing." *Communications in Biometry and Crop Science*, **2**, 32–40. ISSN 1896-0782.

Tebbs, J., Bilder, C. (2004). "Confidence interval procedures for the probability of disease transmission in multiple-vector-transfer designs." *Journal of Agricultural, Biological, and Environmental Statistics*, **9**, 75–90.

See Also

`propDiffCI` for confidence intervals for the difference of proportions in group testing, `gtTest` for hypothesis tests in group testing, `gtPower` for power calculations in group testing, and `binom.test` for an exact confidence interval and test.

Other estimation functions: `designEst()`, `designPower()`, `gtPower()`, `gtTest()`, `gtWidth()`, `propDiffCI()`

Examples

```
# Example from Tebbs and Bilder (2004):
#   3 groups out of 24 test positively;
#   each group has a size of 7.
# Clopper-Pearson interval:
propCI(x = 3, m = 7, n = 24, ci.method = "CP",
       conf.level = 0.95, alternative = "two.sided")

# Clopper-Pearson interval with the bias-corrected
#   MLE (\kbd{pt.method = "Gart"}).
propCI(x = 3, m = 7, n = 24, pt.method = "Gart",
       ci.method = "CP", conf.level = 0.95,
       alternative = "two.sided")

# One-sided Clopper-Pearson interval:
propCI(x = 3, m = 7, n = 24, ci.method = "CP",
       conf.level = 0.95, alternative = "less")

# Blaker interval:
propCI(x = 3, m = 7, n = 24, ci.method = "Blaker",
       conf.level = 0.95, alternative = "two.sided")

# Wilson score interval:
propCI(x = 3, m = 7, n = 24, ci.method = "score",
       conf.level = 0.95, alternative = "two.sided")

# Calculate confidence intervals with a group size of 1.
#   These match those found using the binom.confint()
#   function from the binom package.
propCI(x = 4, m = 1, n = 10, pt.method = "mle",
       ci.method = "AC")
propCI(x = 4, m = 1, n = 10, pt.method = "mle",
       ci.method = "score")
propCI(x = 4, m = 1, n = 10, pt.method = "mle",
       ci.method = "Wald")

# Example from Hepworth (1996, table 5):
#   1 group out of 2 tests positively with
#   groups of size 5; also,
#   2 groups out of 3 test positively with
#   groups of size 2.
propCI(x = c(1,2), m = c(5,2), n = c(2,3), ci.method = "exact")

# Bias-preventative point estimate (\kbd{pt.method = "Firth"})
#   with an exact confidence interval.
propCI(x = c(1,2), m = c(5,2), n = c(2,3),
       pt.method = "Firth", ci.method = "exact")

# Recalculate the example given in
#   Hepworth (1996), table 5:
propCI(x = c(0,0), m = c(5,2), n = c(2,3), ci.method = "exact")
propCI(x = c(0,1), m = c(5,2), n = c(2,3), ci.method = "exact")
propCI(x = c(0,2), m = c(5,2), n = c(2,3), ci.method = "exact")
propCI(x = c(0,3), m = c(5,2), n = c(2,3), ci.method = "exact")
propCI(x = c(1,0), m = c(5,2), n = c(2,3), ci.method = "exact")
propCI(x = c(1,1), m = c(5,2), n = c(2,3), ci.method = "exact")
propCI(x = c(1,2), m = c(5,2), n = c(2,3), ci.method = "exact")
propCI(x = c(1,3), m = c(5,2), n = c(2,3), ci.method = "exact")
propCI(x = c(2,0), m = c(5,2), n = c(2,3), ci.method = "exact")
propCI(x = c(2,1), m = c(5,2), n = c(2,3), ci.method = "exact")
propCI(x = c(2,2), m = c(5,2), n = c(2,3), ci.method = "exact")
propCI(x = c(2,3), m = c(5,2), n = c(2,3), ci.method = "exact")

# Example with multiple groups of various sizes:
#   0 out of 5 groups test positively with
#   groups of size 1 (individual testing);
#   0 out of 5 groups test positively with
#   groups of size 5;
#   1 out of 5 groups test positively with
#   groups of size 10; and
#   2 out of 5 groups test positively with
#   groups of size 50.
x1 <- c(0, 0, 1, 2)
```

```
m1 <- c(1, 5, 10, 50)
n1 <- c(5, 5, 5, 5)
propCI(x = x1, m = m1, n = n1, pt.method = "Gart",
       ci.method = "skew-score")
propCI(x = x1, m = m1, n = n1, pt.method = "Gart",
       ci.method = "score")

# Reproducing estimates from Table 1 in
#   Hepworth & Biggerstaff (2017):
propCI(x = c(1, 2), m = c(20, 5), n = c(8, 8),
       pt.method = "Firth", ci.method = "lrt")
propCI(x = c(7, 8), m = c(20, 5), n = c(8, 8),
       pt.method = "Firth", ci.method = "lrt")
```

---

propDiffCI *Confidence intervals for the difference of proportions in group testing*

Description

Calculates confidence intervals for the difference of two proportions based on group testing data.

Usage

```
propDiffCI(
  x1,
  m1,
  x2,
  m2,
  n1 = rep(1, length(x1)),
  n2 = rep(1, length(x2)),
  pt.method = c("Firth", "Gart", "bc-mle", "mle"),
  ci.method = c("skew-score", "bc-skew-score", "score", "lrt", "Wald"),
  conf.level = 0.95,
  tol = .Machine$double.eps^0.5
)
```

Arguments

`x1`	vector specifying the observed number of positive groups among the number of groups tested (`n1`) in population 1.
`m1`	vector of corresponding group sizes in population 1. Must have the same length as `x1`.
`x2`	vector specifying the observed number of positive groups among the number of groups tested (`n2`) in population 2.
`m2`	vector of corresponding group sizes in population 2. Must have the same length as `x2`.
`n1`	vector of the corresponding number of groups with sizes `m1`.

`n2`	vector of the corresponding number of groups with sizes `m2`.
`pt.method`	character string specifying the point estimator to compute. Options include `"Firth"` for the bias-preventative estimator (Hepworth & Biggerstaff, 2017), the default `"Gart"` for the bias-corrected MLE (Biggerstaff, 2008), `"bc-mle"` (same as `"Gart"` for backward compatibility), and `"mle"` for the MLE.
`ci.method`	character string specifying the confidence interval to compute. Options include `"skew-score"` for the skewness-corrected, `"score"` for the score (the default), `"bc-skew-score"` for the bias- and skewness-corrected, `"lrt"` for the likelihood ratio test, and `"Wald"` for the Wald interval. See Biggerstaff (2008) for additional details.
`conf.level`	confidence level of the interval.
`tol`	the accuracy required for iterations in internal functions.

Details

Confidence interval methods include the Wilson score (`ci.method = "score"`), skewness-corrected score (`ci.method = "skew-score"`), bias- and skewness-corrected score (`ci.method = "bc-skew-score"`), likelihood ratio test (`ci.method = "lrt"`), and Wald (`ci.method = "Wald"`) interval. For computational details, simulation results, and recommendations on confidence interval methods, see Biggerstaff (2008).

Point estimates available include the MLE (`pt.method = "mle"`), bias-corrected MLE (`pt.method = "Gart"` or `pt.method = "bc-mle"`), and bias-preventative (`pt.method = "Firth"`). For additional details and recommendations on point estimation, see Hepworth and Biggerstaff (2017).

Value

A list containing:

`d`	the estimated difference of proportions.
`lcl`	the lower confidence limit.
`ucl`	the upper confidence limit.
`pt.method`	the method used for point estimation.
`ci.method`	the method used for confidence interval estimation.
`conf.level`	the confidence level of the interval.
`x1`	the numbers of positive groups in population 1.
`m1`	the sizes of the groups in population 1.
`n1`	the numbers of groups with corresponding group sizes `m1` in population 1.
`x2`	the numbers of positive groups in population 2.
`m2`	the sizes of the groups in population 2.
`n2`	the numbers of groups with corresponding group sizes `m2` in population 2.

Author(s)

This function was originally written as the `pooledBinDiff` function by Brad Biggerstaff for the `binGroup` package. Minor modifications were made for inclusion of the function in the `binGroup2` package.

References

Biggerstaff, B. (2008). “Confidence intervals for the difference of proportions estimated from pooled samples.” *Journal of Agricultural, Biological, and Environmental Statistics*, **13**, 478–496.

Hepworth, G., Biggerstaff, B. (2017). “Bias correction in estimating proportions by pooled testing.” *Journal of Agricultural, Biological, and Environmental Statistics*, **22**, 602–614.

See Also

`propCI` for confidence intervals for one proportion in group testing, `gtTest` for hypothesis tests in group testing, and `gtPower` for power calculations in group testing.

Other estimation functions: `designEst()`, `designPower()`, `gtPower()`, `gtTest()`, `gtWidth()`, `propCI()`

Examples

```
# Estimate the prevalence in two populations
#   with multiple groups of various sizes:
# Population 1:
#   0 out of 5 groups test positively with
#   groups of size 1 (individual testing);
#   0 out of 5 groups test positively with
#   groups of size 5;
#   1 out of 5 groups test positively with
#   groups of size 10; and
#   2 out of 5 groups test positively with
#   groups of size 50.
# Population 2:
#   0 out of 5 groups test positively with
#   groups of size 1 (individual testing);
#   1 out of 5 groups test positively with
#   groups of size 5;
#   0 out of 5 groups test positively with
#   groups of size 10; and
#   4 out of 5 groups test positively with
#   groups of size 50.
x1 <- c(0, 0, 1, 2)
m <- c(1, 5, 10, 50)
n <- c(5, 5, 5, 5)
x2 <- c(0, 1, 0, 4)
propDiffCI(x1 = x1, m1 = m, x2 = x2, m2 = m, n1 = n, n2 = n,
           pt.method = "Gart", ci.method = "score")

# Compare recommended methods:
propDiffCI(x1 = x1, m1 = m, x2 = x2, m2 = m, n1 = n, n2 = n,
           pt.method = "mle", ci.method = "lrt")

propDiffCI(x1 = x1, m1 = m, x2 = x2, m2 = m, n1 = n, n2 = n,
           pt.method = "mle", ci.method = "score")

propDiffCI(x1 = x1, m1 = m, x2 = x2, m2 = m, n1 = n, n2 = n,
           pt.method = "mle", ci.method = "skew-score")
```

`residuals.gtReg` *Extract model residuals from a fitted group testing model*

Description

Extract model residuals from objects of class "gtReg" returned by `gtReg`.

Usage

```
## S3 method for class 'gtReg'
residuals(object, type = c("deviance", "pearson", "response"), ...)
```

Arguments

`object`	An object of class "gtReg", created by `gtReg`, from which the model residuals are to be extracted.
`type`	The type of residuals which should be returned. Options include "`deviance`" (default), "`pearson`", and "`response`".
`...`	currently not used.

Value

Residuals of group responses extracted from the object `object`.

Author(s)

This function was originally written by Boan Zhang as the `residuals.gt` function for the `binGroup` package.

Examples

```
data(hivsurv)
fit1 <- gtReg(formula = groupres ~ AGE * EDUC.,
              data = hivsurv, groupn = gnum,
              linkf = "probit")
residuals(object = fit1, type = "pearson")
residuals(object = fit1, type = "deviance")
```

`Sterrett`	*Summary measures for Sterrett algorithms*

Description

Summary measures for Sterrett algorithms.

Usage

```
Sterrett(
  p,
  Sp,
  Se,
  plot = FALSE,
  plot.cut.dorf = FALSE,
  cond.prob.plot = FALSE,
  font.name = "sans"
)
```

Arguments

`p`	a vector of individual risk probabilities.
`Sp`	the specificity of the diagnostic test.
`Se`	the sensitivity of the diagnostic test.
`plot`	logical; if TRUE, a plot of the informative Sterrett CDFs will be displayed. Further details are given under 'Details'.
`plot.cut.dorf`	logical; if TRUE, the cut-tree for Dorfman testing will be displayed. Further details are given under 'Details'.
`cond.prob.plot`	logical; if TRUE, a second axis for the conditional probability plot will be displayed on the right side of the plot.
`font.name`	the name of the font to be used in plots.

Details

This function calculates summary measures for informative Sterrett algorithms. Informative algorithms include one-stage informative Sterrett (1SIS), two-stage informative Sterrett (2SIS), full informative Sterrett (FIS), and Dorfman (two-stage hierarchical testing).

The mean and standard deviation of the number of tests, probability mass function (PMF), and cumulative distribution function (CDF) are calculated for all informative Sterrett algorithms and Dorfman testing. Conditional PMFs and conditional moments are calculated for all informative Sterrett algorithms. Subtracting the mean number of tests for two procedures gives the area difference between their CDFs. This area difference is calculated for each pairwise comparison of 1SIS, 2SIS, FIS, and Dorfman testing. CDF plots provide a visualization of how probabilities are distributed over the number of tests. CDFs that increase more rapidly to 1 correspond to more efficient retesting procedures.

Non-informative Sterrett (NIS) decodes positive groups by retesting individuals at random, so there are $I!$ different possible NIS implementations. CDFs are found by permuting the elements in the vector of individual risk probabilities and using the FIS CDF expression without reordering the individual probabilities. That is, the FIS procedure uses the most efficient NIS implementation, which is to retest individuals in order of descending probabilities. When implementing the informative Sterrett algorithms with a large number of individuals, an algorithm is used to compute the PMF for the number of tests under FIS. This is done automatically by `Sterrett` for $I > 12$. The algorithm is described in detail in the Appendix of Bilder et al. (2010).

Value

A list containing:

`mean.sd`	a data frame containing the mean and standard deviation of the expected number of tests for one-stage informative Sterrett (1SIS), two-stage informative Sterrett (2SIS), full informative Sterrett (FIS), and Dorfman testing.
`PMF`	a data frame containing the probability mass function for the number of tests possible for one-stage informative Sterrett (1SIS), two-stage informative Sterrett (2SIS), full informative Sterrett (FIS), and Dorfman testing.
`CDF`	a data frame containing the cumulative distribution function for the number of tests possible for one-stage informative Sterrett (1SIS), two-stage informative Sterrett (2SIS), full informative Sterrett (FIS), and Dorfman testing.
`cond.PMF`	a data frame containing the conditional probability mass function for the number of tests possible for one-stage informative Sterrett (1SIS), two-stage informative Sterrett (2SIS), and full informative Sterrett (FIS) testing.
`cond.moments`	a data frame containing the mean and standard deviation of the conditional moments for one-stage informative Sterrett (1SIS), two-stage informative Sterrett (2SIS), and full informative Sterrett (FIS) testing.
`save.diff.CDF`	a data frame containing the sum of the differences in the cumulative distribution function for each pairwise comparison of one-stage informative Sterrett (1SIS), two-stage informative Sterrett (2SIS), full informative Sterrett (FIS), and Dorfman testing.
`p`	a vector containing the probabilities of positivity for each individual.

Author(s)

This function was originally written as `info.gt` by Christopher Bilder for Bilder et al. (2010). The function was obtained from `http://chrisbilder.com/grouptesting/`. Minor modifications were made for inclusion of the function in the `binGroup2` package.

References

Bilder, C., Tebbs, J., Chen, P. (2010). “Informative retesting.” *Journal of the American Statistical Association*, **105**, 942–955.

See Also

expectOrderBeta for generating a vector of individual risk probabilities for informative group testing and opChar1 for calculating operating characteristics with hierarchical and array-based group testing algorithms.

Other operating characteristic functions: GroupMembershipMatrix(), TOD(), halving(), operatingCharacteristics1() operatingCharacteristics2()

Examples

```
# Example 1: FIS provides the smallest mean
#   number of tests and the smallest standard
#   deviation. 2SIS has slightly larger mean
#   and standard deviation than FIS, but
#   its performance is comparable, indicating
#   2SIS may be preferred because it is
#   easier to implement.
set.seed(1231)
p.vec1 <- rbeta(n = 8, shape1 = 1, shape2 = 10)
save.it1 <- Sterrett(p = p.vec1, Sp = 0.90, Se = 0.95)
save.it1

# Example 2: One individual is "high risk" and
#   the others are "low risk". Since there is
#   only one high-risk individual, the three
#   informative Sterrett procedures perform
#   similarly. All three informative Sterrett
#   procedures offer large improvements over
#   Dorfman testing.
p.vec2 <- c(rep(x = 0.01, times = 9), 0.5)
save.it2 <- Sterrett(p = p.vec2, Sp = 0.99, Se = 0.99)
save.it2

# Example 3: Two individuals are at higher
#   risk than the others. All three informative
#   Sterrett procedures provide large
#   improvements over Dorfman testing.
# Due to the large initial group size, an
#   algorithm (described in the Appendix of
#   Bilder et al. (2010)) is used for FIS.
#   The Sterrett() function does this
#   automatically for I>12.
p.vec3 <- c(rep(x = 0.01, times = 98), 0.1, 0.1)
save.it3 <- Sterrett(p = p.vec3, Sp = 0.99, Se = 0.99)
save.it3
```

---

summary.gtReg *Summary method for* gtReg

Description

Produce a summary list for objects of class "gtReg" returned by gtReg.

Usage

```
## S3 method for class 'gtReg'
summary(object, ...)
```

Arguments

`object`	a fitted object of class "gtReg".
`...`	currently not used.

Details

The `coefficients` component of the results gives a matrix containing the estimated coefficients and their estimated standard errors. The third column is their ratio, labeled `z ratio` using Wald tests. A fourth column gives the two-tailed p-value corresponding to the z-ratio based on a Wald test. Note that it is possible that there are no residual degrees of freedom from which to estimate, in which case the estimate is `NaN`.

Value

`summary.gtReg` returns an object of class "summary.gtReg", a list containing:

`call`	the component from `object`.
`link`	the component from `object`.
`deviance`	the component from `object`, for simple pooling (`type = "sp"` in gtReg) only.
`aic`	the component from `object`, for simple pooling (`type = "sp"` in gtReg) only.
`df.residual`	the component from `object`, for simple pooling (`type = "sp"` in gtReg) only.
`null.deviance`	the component from `object`, for simple pooling (`type = "sp"` in gtReg) only.
`df.null`	the component from `object`, for simple pooling (`type = "sp"` in gtReg) only.
`deviance.resid`	the deviance residuals, for simple pooling (`type = "sp"` in gtReg) only.
`coefficients`	the matrix of coefficients, standard errors, z-values, and p-values. Aliased coefficients are omitted.
`counts`	the component from `object`.
`method`	the component from `object`, for simple pooling (`type = "sp"` in gtReg) only.
`Gibbs.sample.size`	the component from `object`, for array testing (`type = "array"` in gtReg) only.
`cov.mat`	the estimated covariance matrix of the estimated coefficients.

Author(s)

The majority of this function was originally written as `summary.gt` and `summary.gt.mp` by Boan Zhang for the `binGroup` package. Minor modifications were made to the function for inclusion in the `binGroup2` package.

See Also

gtReg for creating an object of class "gtReg".

Examples

```
data(hivsurv)
fit1 <- gtReg(type = "sp",
              formula = groupres ~ AGE + EDUC.,
              data = hivsurv, groupn = gnum,
              sens = 0.9, spec = 0.9,
              method = "Xie")
summary(fit1)

# 5x6 and 4x5 array
set.seed(9128)
sa2a <- gtSim(type = "array", par = c(-7, 0.1),
              size1 = c(5, 4), size2 = c(6, 5),
              sens = 0.95, spec = 0.95)
sa2 <- sa2a$dframe

fit2 <- gtReg(type = "array",
              formula = cbind(col.resp, row.resp) ~ x,
              data = sa2, coln = coln, rown = rown,
              arrayn = arrayn, sens = 0.95, spec = 0.95,
              linkf = "logit", n.gibbs = 1000, tol = 0.005)
summary(fit2)
```

---

summary.opChar — *Summary method for operating characteristics results*

Description

Produce a summary list for objects of class "opChar" returned by operatingCharacteristics1 (opChar1) or operatingCharacteristics2 (opChar2).

Usage

```
## S3 method for class 'opChar'
summary(object, ...)
```

Arguments

object	an object of class "opChar", providing the calculated operating characteristics for a group testing algorithm.
...	currently not used.

Details

This function produces a summary list for objects of class "opChar" returned by operatingCharacteristics1 (opChar1) or operatingCharacteristics2 (opChar2). It formats the testing configuration, expected number of tests, expected number of tests per individual, and accuracy measures.

The Configuration component of the result gives the testing configuration, which may include the group sizes for each stage of a hierarchical testing algorithm or the row/column size and array size for an array testing algorithm. The Tests component of the result gives the expected number of tests and the expected number of tests per individual for the algorithm.

The Accuracy component gives the individual accuracy measures for each individual in object and the overall accuracy measures for the algorithm. Accuracy measures included are the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value. The overall accuracy measures displayed are weighted averages of the corresponding individual accuracy measures for all individuals in the algorithm. Expressions for these averages are provided in the Supplementary Material for Hitt et al. (2019). For more information, see the Details' section for the operatingCharacteristics1 (opChar1) or operatingCharacteristics2 (opChar2) function.

Value

summary.opChar returns an object of class "summary.opChar", a list containing:

Algorithm	character string specifying the name of the group testing algorithm.
Configuration	matrix detailing the configuration from object. For hierarchical testing, this includes the group sizes for each stage of testing. For array testing, this includes the array dimension (row/column size) and the array size (the total number of individuals in the array).
Tests	matrix detailing the expected number of tests and expected number of tests per individual from object

.

Accuracy a list containing:

Individual matrix detailing the accuracy measures for each individual from object (for objects returned by opChar1).

Disease 1 Individual matrix detailing the accuracy measures pertaining to disease 1 for each individual from object (for objects returned by opChar2).

Disease 2 Individual matrix detailing the accuracy measures pertaining to disease 2 for each individual from object (for objects returned by opChar2).

Overall matrix detailing the overall accuracy measures for the algorithm from object.

Author(s)

Brianna D. Hitt

See Also

operatingCharacteristics1 (opChar1) and operatingCharacteristics2 (opChar2) for creating an object of class "opChar".

Examples

```
# Calculate the operating characteristics for
#   non-informative four-stage hierarchical testing.
config.mat <- matrix(data = c(rep(1, 24), rep(1, 16),
                              rep(2, 8), rep(1, 8),
                              rep(2, 8), rep(3, 4),
                              rep(4, 2), rep(5, 2), 1:24),
                     nrow = 4, ncol = 24, byrow = TRUE)
calc1 <- opChar1(algorithm = "D4", p = 0.01,
                 Se = 0.99, Sp = 0.99,
                 hier.config = config.mat,
                 a = c(1, 9, 17, 21, 23))
summary(calc1)

# Calculate the operating characteristics for
#   informative array testing without master pooling.
calc2 <- opChar1(algorithm = "IA2", p = 0.025, alpha = 0.5,
                 Se = 0.95, Sp = 0.99, rowcol.sz = 10)
summary(calc2)

# Calculate the operating characteristics for
#   informative two-stage hierarchical testing
#   with a multiplex assay for two diseases.
config.mat <- matrix(data = c(rep(1, 5), rep(2, 4),
                              1, 1:10),
                     nrow = 2, ncol = 10, byrow = TRUE)
Se <- matrix(data = c(rep(0.95, 2), rep(0.99, 2)),
             nrow = 2, ncol = 2, byrow = FALSE)
Sp <- matrix(data = c(rep(0.96, 2), rep(0.98, 2)),
             nrow = 2, ncol = 2, byrow = FALSE)
calc3 <- opChar2(algorithm = "ID2",
                 alpha = c(18.25, 0.75, 0.75, 0.25),
                 Se = Se, Sp = Sp,
                 hier.config = config.mat)
summary(calc3)

# Calculate the operating characteristics for
#   non-informative array testing with master pooling
#   with a multiplex assay for two diseases.
calc4 <- opChar2(algorithm = "A2M",
                 p.vec = c(0.92, 0.05, 0.02, 0.01),
                 Se = rep(0.95, 2), Sp = rep(0.99, 2),
                 rowcol.sz = 8)
summary(calc4)
```

`summary.OTC` *Summary method for optimal testing configuration results*

Description

Produce a summary list for objects of class "`OTC`" returned by `OTC1` or `OTC2`.

Usage

```
## S3 method for class 'OTC'
summary(object, ...)
```

Arguments

`object`	an object of class "`OTC`", providing the optimal testing configuration and associated operating characteristics for a group testing algorithm.
`...`	currently not used.

Details

This function produces a summary list for objects of class "`OTC`" returned by `OTC1` or `OTC2`. It formats the optimal testing configuration, expected number of tests, expected number of tests per individual, and accuracy measures. A summary of the results from `OTC1` includes results for all objective functions specified by the user.

The `OTC` component of the result gives the optimal testing configuration, which may include the group sizes for each stage of a hierarchical testing algorithm or the row/column size and array size for an array testing algorithm. The `Tests` component of the result gives the expected number of tests and the expected number of tests per individual for the algorithm.

The `Accuracy` component gives the overall accuracy measures for the algorithm. Accuracy measures included are the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value. These values are weighted averages of the corresponding individual accuracy measures for all individuals in the algorithm. Expressions for these averages are provided in the Supplementary Material for Hitt et al. (2019). For more information, see the 'Details' section for the `OTC1` or `OTC2` function.

Value

`summary.OTC` returns an object of class "`summary.OTC`", a list containing:

`Algorithm`	character string specifying the name of the group testing algorithm.
`OTC`	matrix detailing the optimal testing configuration from `object`. For hierarchical testing, this includes the group sizes for each stage of testing. For array testing, this includes the array dimension (row/column size) and the array size (the total number of individuals in the array).
`Tests`	matrix detailing the expected number of tests and expected number of tests per individual from `object`.
`Accuracy`	matrix detailing the overall accuracy measures for the algorithm, including the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for the algorithm from `object`. Further details are found in the 'Details' section.

Author(s)

Brianna D. Hitt

See Also

`OTC1` and `OTC2` for creating an object of class "OTC".

Examples

```
# Find the optimal testing configuration for
#   non-informative two-stage hierarchical testing.
res1 <- OTC1(algorithm = "D2", p = 0.01, Se = 0.99, Sp = 0.99,
             group.sz = 2:100, obj.fn = c("ET", "MAR", "GR1"),
             weights = matrix(data = c(1,1), nrow = 1, ncol = 2))
summary(res1)

# Find the optimal testing configuration for
#   informative three-stage hierarchical testing
res2 <- OTC1(algorithm = "ID3", p = 0.025,
             Se = c(0.95, 0.95, 0.99), Sp = c(0.96, 0.96, 0.98),
             group.sz = 3:10, obj.fn = c("ET", "MAR"), alpha = 2)
summary(res2)

# Find the optimal testing configuration for
#   informative array testing without master pooling.
res3 <- OTC1(algorithm = "IA2", p = 0.05, alpha = 2,
             Se = 0.90, Sp = 0.90, group.sz = 2:15,
             obj.fn = "ET")
summary(res3)

# Find the optimal testing configuraiton for
#   informative two-stage hierarchical testing.
Se <- matrix(data = c(rep(0.95, 2), rep(0.99, 2)),
             nrow = 2, ncol = 2, byrow = FALSE)
Sp <- matrix(data = c(rep(0.96, 2), rep(0.98, 2)),
             nrow = 2, ncol = 2, byrow = FALSE)
res4 <- OTC2(algorithm = "ID2",
             alpha = c(18.25, 0.75, 0.75, 0.25),
             Se = Se, Sp = Sp, group.sz = 8)
summary(res4)

# Find the optimal testing configuration for
#   non-informative three-stage hierarchical testing.
Se <- matrix(data = c(rep(0.95, 6)), nrow = 2, ncol = 3)
Sp <- matrix(data = c(rep(0.99, 6)), nrow = 2, ncol = 3)
res5 <- OTC2(algorithm = "D3",
             p.vec = c(0.95, 0.0275, 0.0175, 0.005),
             Se = Se, Sp = Sp, group.sz = 5:12)
summary(res5)

# Find the optimal testing configuration for
#   non-informative array testing with master pooling.
res6 <- OTC2(algorithm = "A2M", p.vec = c(0.90, 0.04, 0.04, 0.02),
             Se = rep(0.99, 2), Sp = rep(0.99, 2), group.sz = 2:12)
summary(res6)
```

`TOD` *Summary measures for the Thresholded Optimal Dorfman (TOD) algorithm*

Description

Summary measures for the Thresholded Optimal Dorfman (TOD) algorithm.

Usage

```
TOD(p.vec, Se, Sp, max = 15, init.group.sz = NULL, threshold = NULL)
```

Arguments

`p.vec`	a vector of individual risk probabilities.
`Se`	sensitivity of the diagnostic test.
`Sp`	specificity of the diagnostic test.
`max`	the maximum allowable group size. Further details are given under 'Details'.
`init.group.sz`	the initial group size used for TOD, if `threshold` is not specified. Further details are given under 'Details'.
`threshold`	the threshold value for TOD. If a threshold is not specified, one is found algorithmically. Further details are given under 'Details'.

Details

This function finds the characteristics of an informative two-stage hierarchical (Dorfman) decoding process. Characteristics found include the expected expenditure of the decoding process, the variance of the expenditure of the decoding process, and the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for each individual and for the overall algorithm. Calculations of these characteristics are done using equations presented in McMahan et al. (2012).

Thresholded Optimal Dorfman (TOD) is an informative Dorfman algorithm in which all N individuals are partitioned into two classes, low-risk and high-risk individuals. The threshold can be specified using the optional `threshold` argument. Alternatively, the TOD algorithm can identify the optimal threshold value. The low-risk individuals are tested using an optimal common pool size, and the high-risk individuals are tested individually. If desired, the user can add the constraint of a maximum allowable group size (`max`), so that each group will contain no more than the maximum allowable number of individuals.

The displayed overall pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value are weighted averages of the corresponding individual accuracy measures for all individuals within the initial group (or block) for a hierarchical algorithm, or within the entire array for an array-based algorithm. Expressions for these averages are provided in the Supplementary Material for Hitt et al. (2019). These expressions are based on accuracy definitions given by Altman and Bland (1994a, 1994b).

Value

A list containing:

`prob`	the vector of individual risk probabilities, as specified by the user.
`Se`	the sensitivity of the diagnostic test, as specified by the user.
`Sp`	the specificity of the diagnostic test, as specified by the user.
`group.sz`	the initial group size used for TOD, if applicable.
`thresh.val`	the threshold value used for TOD, if applicable.
`OTC`	a list specifying elements of the optimal testing configuration, which may include: **Block.sz** the block size/initial group size for informative Dorfman testing, which is not tested. **pool.szs** group sizes for the first stage of testing for informative Dorfman testing.
`ET`	the expected testing expenditure to decode all individuals in the algorithm.
`Var`	the variance of the testing expenditure to decode all individuals in the algorithm.
`Accuracy`	a list containing: **Individual** a matrix of accuracy measures for each individual. The rows correspond to each unique set of accuracy measures in the algorithm. Individuals with the same set of accuracy measures are displayed together in a single row of the matrix. The columns correspond to the pool index, the individual risk probability, and the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for the individuals in each row of the matrix. **Overall** a matrix of overall accuracy measures for the algorithm. The columns correspond to the pooling sensitivity, pooling specificity, pooling positive predictive value, and pooling negative predictive value for the overall algorithm. Further details are given under 'Details'.

Author(s)

Brianna D. Hitt

References

Altman, D., Bland, J. (1994). “Diagnostic tests 1: Sensitivity and specificity.” *BMJ*, **308**, 1552.

Altman, D., Bland, J. (1994). “Diagnostic tests 2: Predictive values.” *BMJ*, **309**, 102.

Hitt, B., Bilder, C., Tebbs, J., McMahan, C. (2019). “The objective function controversy for group testing: Much ado about nothing?” *Statistics in Medicine*, **38**, 4912–4923.

McMahan, C., Tebbs, J., Bilder, C. (2012a). “Informative Dorfman Screening.” *Biometrics*, **68**, 287–296.

See Also

expectOrderBeta for generating a vector of individual risk probabilities.

Other operating characteristic functions: GroupMembershipMatrix(), Sterrett(), halving(), operatingCharacteristics1(), operatingCharacteristics2()

Examples

```
# Example 1: Find the characteristics of an informative
#   Dorfman algorithm, using the TOD procedure.
set.seed(1002)
p.vec <- expectOrderBeta(p = 0.01, alpha = 2, size = 20)
TOD(p = p.vec, Se = 0.95, Sp = 0.95, max = 5,
    threshold = 0.015)

# Example 2: Find the threshold value for the TOD
#   procedure algorithmically. Then, find
#   characteristics of the algorithm.
TOD(p = p.vec, Se = 0.95, Sp = 0.95, max = 5,
    init.group.sz = 10)
```

Index