Package ‘prevalence’

April 20, 2015

Type Package
Title Tools for Prevalence Assessment Studies
Version 0.4.0
Date 2015-04-20
Author Brecht Devleesschauwer [aut, cre], Paul Torgerson [aut], Johannes Charlier [aut], Bruno Levecke [aut], Nicolas Praet [aut], Sophie Roelandt [aut], Suzanne Smit [aut], Pierre Dorny [aut], Dirk Berkvens [aut], Niko Speybroeck [aut]
Maintainer Brecht Devleesschauwer <brechtdv@gmail.com>

BugReports https://github.com/brechtdv/prevalence/issues

Description The prevalence package provides Frequentist and Bayesian methods for prevalence assessment studies. IMPORTANT: the truePrev functions in the prevalence package call on JAGS (Just Another Gibbs Sampler), which therefore has to be available on the user's system. JAGS can be downloaded from http://mcmc-jags.sourceforge.net/.

Depends R (>= 3.0.0), rjags, coda, methods
SystemRequirements JAGS (>= 3.2.0) (see http://mcmc-jags.sourceforge.net)
License GPL (>= 2)
URL http://prevalence.cbra.be/
LazyLoad yes

R topics documented:

prevalence-package ..................................................... 2
betaExpert ............................................................. 3
betaPERT .............................................................. 5
convert-methods ........................................................ 7
Description


Details

<table>
<thead>
<tr>
<th>Package</th>
<th>prevalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Package</td>
</tr>
<tr>
<td>Version</td>
<td>0.4.0</td>
</tr>
<tr>
<td>Date</td>
<td>2015-04-20</td>
</tr>
<tr>
<td>BugReports</td>
<td><a href="https://github.com/brechtdv/prevalence/issues">https://github.com/brechtdv/prevalence/issues</a></td>
</tr>
<tr>
<td>Depends</td>
<td>R (&gt;= 3.0.0), rjags, coda, methods</td>
</tr>
<tr>
<td>SystemRequirements</td>
<td>JAGS (&gt;= 3.2.0) (see <a href="http://mcmc-jags.sourceforge.net">http://mcmc-jags.sourceforge.net</a>)</td>
</tr>
<tr>
<td>License</td>
<td>GNU &gt;= 2</td>
</tr>
</tbody>
</table>

Available functions in the `prevalence` package:

- **propCI**: Derive confidence intervals for an apparent prevalence estimate
- **truePrev**: Estimate TP from AP obtained by testing individual samples with a single test
- **truePrevMulti**: Estimate TP from AP obtained by testing individual samples with multiple tests, using a conditional probability scheme
- **truePrevMulti2**: Estimate TP from AP obtained by testing individual samples with multiple tests, using a covariance scheme
- **truePrevPools**: Estimate TP from AP obtained by testing pooled samples
- **betaPERT**: Calculate the parameters of a Beta-PERT distribution
- **betaExpert**: Calculate the parameters of a Beta distribution based on expert opinion

**IMPORTANT**: the `truePrev` functions in the `prevalence` package call on JAGS (Just Another Gibbs Sampler), through the `rjags` package. Therefore, JAGS has to be installed on the user's
JAGS can be downloaded from http://sourceforge.net/projects/mcmc-jags/

Author(s)

Creator, Maintainer
Brech Devleesschauwer <brechdv@gmail.com>

Contributors

betaExpert

Calculate the parameters of a Beta distribution based on expert information

Description

The betaExpert function fits a (standard) Beta distribution to expert opinion. The expert provides information on a best-guess estimate (mode or mean), and an uncertainty range:

- The parameter value is with 100*p% certainty greater than lower
- The parameter value is with 100*p% certainty smaller than upper
- The parameter value lies with 100*p% in between lower and upper

Usage

betaExpert(best, lower, upper, p = 0.95, method = "mode")

## S3 method for class 'betaExpert'
print(x, conf.level = .95, ...)
## S3 method for class 'betaExpert'
plot(x, y, ...)

Arguments

best Best-guess estimate; see argument method
lower Lower uncertainty limit
upper Upper uncertainty limit
p Expert’s certainty level
method Does best-guess estimate correspond to the mode or to the mean? Defaults to mode
x Object of class betaExpert
y Currently not implemented
conf.level Confidence level used in printing quantiles of resulting Beta distribution
... Other arguments to pass to function print and plot
The methodology behind the `betaExpert` function is presented by Branscum et al. (2005) and implemented in the `BetaBuster` software, written by Chun-Lung Su.

The parameters of a standard Beta distribution are calculated based on a best-guess estimate and a 100($p$)% uncertainty range, defined by a lower and/or upper limit. The `betaExpert` function uses minimization (`optimize`) to derive $\alpha$ and $\beta$ from this best guess and lower and/or upper limit. The resulting distribution is a standard 2-parameter Beta distribution: Beta($\alpha$, $\beta$).

**Value**

A list of class "betaExpert":

- `alpha`: Parameter $\alpha$ (shape1) of the Beta distribution
- `beta`: Parameter $\beta$ (shape2) of the Beta distribution

The print method for "betaExpert" additionally calculates the mean, median, mode, variance and range of the corresponding Beta distribution.

**Author(s)**

Brecht Devleesschauwer <brechtdv@gmail.com>

**References**


**See Also**

Package `rriskDistributions`, which provides a collection of functions for fitting distributions to given data or by known quantiles.

`betaPERT`, for modelling a generalized Beta distribution based on expert opinion

**Examples**

```r
## Most likely value (mode) is 90%
## Expert states with 95% certainty that true value is larger than 70%
betaExpert(best = 0.90, lower = 0.70, p = 0.95)

## Most likely value (mode) is 0%
## Expert states with 95% certainty that true value is smaller than 40%
betaExpert(best = 0, upper = 0.40, p = 0.95)

## Most likely value (mode) is 80%
## Expert states with 90% certainty that true value lies in between 40% and 90%
betaExpert(best = 0.80, lower = 0.40, upper = 0.90, p = 0.90)
```
## betaPERT

**Calculate the parameters of a Beta-PERT distribution**

### Description

The Beta-PERT methodology allows to parametrize a generalized Beta distribution based on expert opinion regarding a pessimistic estimate (minimum value), a most likely estimate (mode), and an optimistic estimate (maximum value). The `betaPERT` function incorporates two methods of calculating the parameters of a Beta-PERT distribution, designated "classic" and "vose".

### Usage

```r
betaPERT(a, m, b, k = 4, method = c("classic", "vose"))
```

```r
# S3 method for class 'betaPERT'
print(x, conf.level = .95, ...)
```

```r
# S3 method for class 'betaPERT'
plot(x, y, ...)
```

### Arguments

- **a**: Pessimistic estimate (Minimum value)
- **m**: Most likely estimate (Mode)
- **b**: Optimistic estimate (Maximum value)
- **k**: Scale parameter
- **method**: "classic" or "vose"; see details below
- **x**: Object of class `betaExpert`
- **y**: Currently ignored
- **conf.level**: Confidence level used in printing quantiles of resulting Beta-PERT distribution
- **...**: Other arguments to pass to function `print` and `plot`

### Details

The Beta-PERT methodology was developed in the context of Program Evaluation and Review Technique (PERT). Based on a pessimistic estimate (minimum value), a most likely estimate (mode), and an optimistic estimate (maximum value), typically derived through expert elicitation, the parameters of a Beta distribution can be calculated. The Beta-PERT distribution is used in stochastic modeling and risk assessment studies to reflect uncertainty regarding specific parameters.

Different methods exist in literature for defining the parameters of a Beta distribution based on PERT. The two most common methods are included in the `betaPERT` function:
**Classic:** The standard formulas for mean, standard deviation, $\alpha$ and $\beta$, are as follows:

\[
\text{mean} = \frac{a + km + b}{k + 2}
\]

\[
\text{sd} = \frac{b - a}{k + 2}
\]

\[
\alpha = \frac{\text{mean} - a}{b - a} \left\{ \left( \frac{\text{mean} - a}{b - a} \right) \cdot \frac{b - \text{mean}}{\text{sd}^2} - 1 \right\}
\]

\[
\beta = \alpha \cdot \frac{b - \text{mean}}{\text{mean} - a}
\]

The resulting distribution is a 4-parameter Beta distribution: Beta($\alpha$, $\beta$, a, b).

**Vose:** Vose (2000) describes a different formula for $\alpha$:

\[
(mean - a) \cdot \frac{2m - a - b}{(m - mean) \cdot (b - a)}
\]

Mean and $\beta$ are calculated using the standard formulas; as for the classical PERT, the resulting distribution is a 4-parameter Beta distribution: Beta($\alpha$, $\beta$, a, b).

Note: If $m = \text{mean}$, $\alpha$ is calculated as $1 + k/2$, in accordance with the mc2d package (see 'Note').

**Value**

A list of class "betaPERT":

- **alpha** Parameter $\alpha$ (shape1) of the Beta distribution
- **beta** Parameter $\beta$ (shape2) of the Beta distribution
- **a** Pessimistic estimate (Minimum value)
- **m** Most likely estimate (Mode)
- **b** Optimistic estimate (Maximum value)
- **method** Applied method

Available generic functions for class "betaPERT" are print and plot.

**Note**

The mc2d package provides the probability density function, cumulative distribution function, quantile function and random number generation function for the PERT distribution, parametrized by the "vose" method.

**Author(s)**

Brecht Devleesschauwer <brechtdv@gmail.com>
References


PERT distribution in *ModelRisk* (Vose software)

See Also

`betaExpert`, for modelling a standard Beta distribution based on expert opinion

Examples

```r
## The value of a parameter of interest is believed to lie between 0 and 50
## The most likely value is believed to be 10

# Classical PERT
betaPERT(a = 0, m = 10, b = 50, method = "classic")

# Vose parametrization
betaPERT(a = 0, m = 10, b = 50, method = "vose")
```

---

**Description**

Convert objects of class `prev` to matrix

**Usage**

```r
## S4 method for signature 'prev'
as.matrix(x, iters = FALSE, chains = FALSE)
```

**Arguments**

- `x` An object of class `prev`
- `iters` Logical flag, indicating whether a column should be added for iteration number; defaults to `FALSE`
- `chains` Logical flag, indicating whether a column should be added for chain number; defaults to `FALSE`

**Methods**

`signature(x = "prev")` Convert objects of class `prev` to `matrix`

**See Also**

`prev-class`
**Examples**

```r
## Not run:

## Taenia solium cysticercosis 1-test model
cysti <-
  truePrev(x = 142, n = 742,
   SE = dunif(0.60, 1.00), SP = dunif(0.75, 1.00))

head(as.matrix(cysti))

## Campylobacter 2-test model
campy <-
  truePrevMulti(
    x = c(113, 46, 156, 341),
    n = 656,
    prior = {
      theta[1] ~ dunif(0.45, 0.80)
      theta[2] ~ dunif(0.24, 0.50)
      theta[3] <- 1
      theta[4] ~ dbeta(30, 12)
      theta[5] ~ dbeta(1, 1)
      theta[6] <- 1
      theta[7] <- 1
    }
  )

head(as.matrix(campy, iters = TRUE, chains = TRUE))

## End(Not run)
```

---

**Definition of truePrevMulti and truePrevMulti2 model**

**Description**

These utility functions generate definitions for the test results and priors used by `truePrevMulti` and `truePrevMulti2`.

**Usage**

```r
define_x(h)
define_prior(h)
define_prior2(h)
```

**Arguments**

- `h` Number of tests
Details

The vector of apparent tests results, \( x \), must contain the number of samples corresponding to each combination of test results. The models assume that the first value corresponds to the number of samples that tested positive on all tests and that the last value corresponds to the number of samples that tested negative on all tests.

Function \texttt{truePrevmulti} estimates true prevalence from individual samples tested with \( h \) tests, using the approach of Berkvens et al. (2006). The prior in the multinomial model consists of a vector \( \theta \), which holds values for the true prevalence (TP), the sensitivity and specificity of the first test (SE1, SP1), and the conditional dependencies between the results of the subsequent tests and the preceding one(s). \texttt{define_prior} generates the definition of \texttt{prior} for \( h \) tests.

Function \texttt{truePrevmulti2} implements and extends the approach described by Dendukuri and Joseph (2001), which uses a multinomial distribution to model observed test results, and in which conditional dependence between tests is modelled through covariances. Argument \texttt{prior} consists of prior distributions for:

- True Prevalence: TP
- SEnsitivity of each individual test: vector SE
- SPecificity of each individual test: vector SP
- Conditional covariance of all possible test combinations given a truly positive disease status: vector \( a \)
- Conditional covariance of all possible test combinations given a truly negative disease status: vector \( b \)

\texttt{define_prior2} generates the definition of \texttt{prior} for \( h \) tests.

Author(s)

Brecht Devleesschauwer <brechtdv@gmail.com>

References

- Dendukuri N, Joseph L (2001) Bayesian approaches to modeling the conditional dependence between multiple diagnostic tests. Biometrics 57:158-167

See Also

\texttt{truePrevmulti}, \texttt{truePrevmulti2}

Examples

```r
# how is a 2-test model defined?
define_x(2)
# Definition of the apparent test results, 'x', for 2 tests:
# x[1]: T1-, T2-
```
define

# x[2] : T1-, T2+
# x[3] : T1+, T2-
# x[4] : T1+, T2+

define_prior(2)
# Conditional probability scheme
# Definition of the prior, 'theta', for 2 tests:
# theta[1] : Pr(D+) = TP
# theta[2] : Pr(T1+|D+) = SE1
# theta[3] : Pr(T1-|D-) = SP1
# theta[4] : Pr(T2+|D+, T1+)
# theta[5] : Pr(T2+|D+, T1-)
# theta[6] : Pr(T2-|D-, T1-)
# theta[7] : Pr(T2-|D-, T1+)

define_prior2(2)
# Covariance scheme
# Definition of the prior for 2 tests:
# TP : True Prevalence
# SE[1] : Sensitivity T1
# SP[1] : Specificity T1
# SP[2] : Specificity T2
# a[1] : Covariance(T1, T2|D+)
# b[1] : Covariance(T1, T2|D-)

## how is a 3-test model defined?

define_x(3)
# Definition of the apparent test results, 'x', for 3 tests:
# x[1] : T1-, T2-, T3-
# x[2] : T1-, T2-, T3+
# x[3] : T1-, T2+, T3-
# x[4] : T1-, T2+, T3+
# x[5] : T1+, T2-, T3-
# x[6] : T1+, T2-, T3+
# x[7] : T1+, T2+, T3-
# x[8] : T1+, T2+, T3+

define_prior(3)
# Conditional probability scheme
# Definition of the prior, 'theta', for 3 tests:
# theta[1] : Pr(D+) = TP
# theta[2] : Pr(T1+|D+) = SE1
# theta[3] : Pr(T1-|D-) = SP1
# theta[4] : Pr(T2+|D+, T1+)
# theta[5] : Pr(T2+|D+, T1-)
# theta[6] : Pr(T2-|D-, T1-)
# theta[7] : Pr(T2-|D-, T1+)
# theta[8] : Pr(T3+|D+, T1+, T2+)
# theta[9] : Pr(T3+|D+, T1+, T2-)
# theta[10] : Pr(T3+|D+, T1-, T2+)
# theta[11] : Pr(T3+|D+, T1-, T2-)

define_prior2(3)
# Covariance scheme
# Definition of the prior for 3 tests:
# TP : True Prevalence
# SE[1] : Sensitivity T1
# SP[1] : Specificity T1
# SP[2] : Specificity T2
# SP[3] : Specificity T3
# a[1] : Covariance(T1, T2, T3|D+)
# b[2] : Covariance(T1, T2, T3|D-)
# c[2] : Covariance(T2, T3|D+)
# b[3] : Covariance(T1, T2, T3|D-)
# c[3] : Covariance(T2, T3|D-)

## how is a 4-test model defined?
Methods for Function `plot` in Package `prevalence`

**Description**

Plot objects of class `prev`

**Usage**

```r
## S4 method for signature 'prev,ANY'
plot(x, y = NULL, ...)
```

**Arguments**

- `x`  
  An object of class `prev`

- `y`  
  Which parameter to plot? Defaults to `NULL`, in which case TP will be used

- `...`  
  Other arguments to pass to the `plot` function

**Methods**

`signature(x = "prev", y = "ANY")` Show `density`, `trace`, `Brooks-Gelman-Rubin` and `autocorrelation` plots.
plot-methods-coda

Description

Different plotting functions from package coda have been made available as method to class prev

Usage

```r
## S4 method for signature 'prev'
densplot(x, exclude_fixed = TRUE, ...)

## S4 method for signature 'prev'
traceplot(x, exclude_fixed = TRUE, ...)

## S4 method for signature 'prev'
autocorr.plot(x, exclude_fixed = TRUE, chain = 1, ...)
```

Arguments

- `x`: An object of class prev
- `exclude_fixed`: Should fixed parameters be excluded from plotting? defaults to TRUE
- `chain`: Which chain to plot in autocorr.plot; defaults to 1
- `...`: Other arguments to pass to the specific plot function.

Methods

- `signature(x = "prev")`: Show density, trace, Brooks-Gelman-Rubin and autocorrelation plots.

See Also

- `prev-class`
- `densplot-methods, traceplot-methods, gelman.plot-methods, autocorr.plot-methods`
prev-class

prev-class Class "prev"

Description

The "prev" class represents output from Bayesian true prevalence estimation models.

Objects from the Class

Objects of class "prev" are created by `trueprev`, `truePrevMulti`, `truePrevMulti2` and `truePrevPools`.

Slots

Objects of class "prev" contain the following four slots:

- `par`: A list of input parameters
- `model`: The fitted Bayesian model, in BUGS language (S3 class "prevModel")
- `mcmc`: A list, with one element per chain, of the simulated true prevalences, sensitivities and specificities
- `diagnostics`: A list with elements for the Deviance Information Criterion ($\text{DIC}$), the Brooks-Gelman-Rubin statistic ($\text{BGR}$), and in the case of `truePrevMulti` and `truePrevMulti2`, the Bayes-P statistic ($\text{bayesP}$)

Author(s)

Brecht Devleesschauwer <<brechtdv@gmail.com>>

See Also

`truePrev`, `truePrevMulti`, `truePrevMulti2`, `truePrevPools`  
`show-methods`, `print-methods`, `summary-methods`, `convert-methods`, `plot-methods`, `plot-methods-coda`

Examples

```r
## Taenia solium cysticercosis in Nepal
SE <- list(dist = "uniform", min = 0.60, max = 1.00)
SP <- list(dist = "uniform", min = 0.75, max = 1.00)
TP <- truePrev(x = 142, n = 742, SE = SE, SP = SP)

## Summarize estimates per chain
summary(TP)

## Diagnostic plots
par(mfrow = c(2, 2))
plot(TP)

## Generic plots from package coda
par(mfrow = c(1, 1))
```
densplot(TP)
traceplot(TP)
gelman.plot(TP)
autocorr.plot(TP)

## Use 'str()' to see the structure of object TP
str(TP)

## Every slot can be accessed using the '@' operator
TP@par # input parameters
TP@model # fitted model
TP@mcmc # simulated TP, SE, SP
TP@diagnostics # DIC and BGR (and bayesP)

## Each element of TP@mcmc inherits from coda class 'mcmc.list'
## List all available methods for this class
methods(class = "mcmc.list")
## List all available functions in the coda package
library(help = "coda")

## Highest Posterior Density interval, from coda package
HPDinterval(TP@mcmc$TP)

---

**print-methods**

**Methods for Function print in Package prevalence**

---

**Description**

Print objects of class prev

**Usage**

```r
## S4 method for signature 'prev'
print(x, conf.level = 0.95, dig = 3, ...)
```

**Arguments**

- `x` An object of class prev
- `conf.level` Confidence level to be used in credibility interval
- `dig` Number of decimal digits to print
- `...` Other arguments to pass to the print function

**Methods**

`signature(x = "prev")` Print mean, median, mode, standard deviation and credibility interval of estimated true prevalence, sensitivities and specificities. In addition, print multivariate Brooks-Gelman-Rubin statistic (or univariate BGR statistic with corresponding upper confidence limit in case of a single stochastic node). BGR values substantially above 1 indicate lack of convergence. For prev objects created by `truePrevMulti`, the Bayes-P statistic is also printed. Bayes-P should be as close to 0.5 as possible.
propCI

See Also
prevMclass
gelman.diag

propCI  

Calculate confidence intervals for prevalences and other proportions

Description
The propCI function calculates five types of confidence intervals for proportions:

- Wald interval (= Normal approximation interval, asymptotic interval)
- Agresti-Coull interval (= adjusted Wald interval)
- Exact interval (= Clopper-Pearson interval)
- Jeffreys interval (= Bayesian interval)
- Wilson score interval

Usage
propCI(x, n, method = "all", level = 0.95, sortby = "level")

Arguments
- x: Number of successes (positive samples)
- n: Number of trials (sample size)
- method: Confidence interval calculation method; see details
- level: Confidence level for confidence intervals
- sortby: Sort results by "level" or "method"

Details
Five methods are available for calculating confidence intervals. For convenience, synonyms are allowed. Please refer to the PDF version of the manual for proper formatting of the below formulas.

"agresti.coull", "agresti-coull", "ac"

\[
\hat{n} = n + z_{1-\frac{\alpha}{2}}^2
\]

\[
\hat{p} = \frac{1}{\hat{n}}(x + \frac{1}{2}z_{1-\frac{\alpha}{2}}^2)
\]

\[
\hat{p} \pm z_{1-\frac{\alpha}{2}} \sqrt{\frac{\hat{p}(1-\hat{p})}{\hat{n}}}
\]

"exact", "clopper-pearson", "cp"

\[
(Beta(\frac{\alpha}{2}; x, n - x + 1), Beta(1 - \frac{\alpha}{2}; x + 1, n - x))
\]
"jeffreys", "bayes"

\[
\left( \text{Beta} \left( \frac{\alpha}{2} : x + 0.5, n - x + 0.5 \right), \text{Beta} \left( 1 - \frac{\alpha}{2} : x + 0.5, n - x + 0.5 \right) \right)
\]

"wald", "asymptotic", "normal"

\[
p \pm z_{1 - \frac{\alpha}{2}} \sqrt{\frac{p(1 - p)}{n}}
\]

"wilson"

\[
p + \frac{z_{1 - \frac{\alpha}{2}}^2}{2n} \pm z_{1 - \frac{\alpha}{2}} \sqrt{\frac{p(1 - p)}{n} + \frac{z_{1 - \frac{\alpha}{2}}^2}{4n^2} \left( 1 + \frac{z_{1 - \frac{\alpha}{2}}^2}{n} \right)}
\]

Value

Data frame with seven columns:

- **x**: Number of successes (positive samples)
- **n**: Number of trials (sample size)
- **p**: Proportion of successes (prevalence)
- **method**: Confidence interval calculation method
- **level**: Confidence level
- **lower**: Lower confidence limit
- **upper**: Upper confidence limit

Note

In case the observed prevalence equals 0% (ie, \(x = 0\)), an upper one-sided confidence interval is returned. In case the observed prevalence equals 100% (ie, \(x = n\)), a lower one-sided confidence interval is returned. In all other cases, two-sided confidence intervals are returned.

Author(s)

Brecht Devleesschauwer <brechtdv@gmail.com>

Examples

```r
## All methods, 95% confidence intervals
propCI(x = 142, n = 742)

## Wald-type 90%, 95% and 99% confidence intervals
propCI(x = 142, n = 742, method = "wald", level = c(0.90, 0.95, 0.99))
```
show-methods

Methods for Function show in Package prevalence

Description
Show objects of class prev

Usage
```
## S4 method for signature 'prev'
show(object)
```

Arguments
- `object`: An object of class `prev`

Methods
- `signature(object = "prev")` Corresponds to `print(object)`

See Also
- `prev-class`

summary-methods

Methods for Function summary in Package prevalence

Description
Summarize objects of class prev

Usage
```
## S4 method for signature 'prev'
summary(object, conf.level)
```

Arguments
- `object`: An object of class `prev`
- `conf.level`: Confidence level to be used in credibility intervals

Methods
- `signature(object = "prev")` Obtain mean, median, mode, standard deviation, variance, credibility interval and number of samples for each chain separately and for all chains combined.
See Also

prev-class

truePrev

Estimate true prevalence from individuals samples

Description

Bayesian estimation of true prevalence from apparent prevalence obtained by testing individual samples.

Usage

truePrev(x, n, SE = 1, SP = 1, prior = c(1, 1),
  nchains = 2, burnin = 10000, update = 10000,
  verbose = FALSE)

Arguments

x The apparent number of positive samples
n The sample size
SE, SP The prior distribution for sensitivity (SE) and specificity SP); see 'Details' below for specification of these distributions
prior The parameters of the prior Beta distribution for true prevalence; defaults to c(1, 1)
nchains The number of chains used in the estimation process; 'n' must be ≥ 2
burnin The number of discarded model iterations; defaults to 10,000
update The number of withheld model iterations; defaults to 10,000
verbose Logical flag, indicating if JAGS process output should be printed to the R console; defaults to FALSE

Details

truePrev calls on JAGS/rjags to estimate the true prevalence from the apparent prevalence in a Bayesian framework. The default model, in BUGS language, is given below. To see the actual fitted model, see the model slot of the prev-object.

model {
  x ~ dbin(AP, n)
  AP <- SE * TP + (1 - SP) * (1 - TP)
  # SE ~ user-defined (see below)
  # SP ~ user-defined (see below)
  TP ~ dbeta(prior[1], prior[2])
}

truePrev
The test sensitivity (SE) and specificity (SP) can be specified, independently, as one of "fixed", "uniform", "beta", "pert", or "beta-expert", with "fixed" as the default.

Distribution parameters can be specified in a named list() as follows:

- **Fixed**: list(dist = "fixed", par)
- **Uniform**: list(dist = "uniform", min, max)
- **Beta**: list(dist = "beta", alpha, beta)
- **Beta-PERT**: list(dist = "pert", method, a, m, b, k)
  'method' must be "classic" or "vose";
  'a' denotes the pessimistic (minimum) estimate, 'm' the most likely estimate, and 'b' the optimistic (maximum) estimate;
  'k' denotes the scale parameter.
  See betaPERT for more information on Beta-PERT parametrization.
- **Beta-Expert**: list(dist = "beta-expert", mode, mean, lower, upper, p)
  'mode' denotes the most likely estimate, 'mean' the mean estimate;
  'lower' denotes the lower bound, 'upper' the upper bound;
  'p' denotes the confidence level of the expert.
  Only mode or mean should be specified; lower and upper can be specified together or alone.
  See betaExpert for more information on Beta-Expert parametrization.

For Uniform, Beta and Beta-PERT distributions, BUGS-style short-hand notation is also allowed:

- **Uniform**: ~dunif(min, max)
- **Beta**: ~dbeta(alpha, beta)
- **Beta-PERT**: ~dpert(min, mode, max)

**Value**

An object of class `prev`.

**Note**

Markov chain Monte Carlo sampling in truePrev is performed by JAGS (Just Another Gibbs Sampler) through the `rjags` package. JAGS can be downloaded from [http://sourceforge.net/projects/mcmc-jags/](http://sourceforge.net/projects/mcmc-jags/).

**Author(s)**

Brecht Devleesschauwer <<brechtdv@gmail.com>>

**References**

See Also

- **coda** for various functions that can be applied to the `prev@mcmc` object
- **truePrevMulti**: estimate true prevalence from apparent prevalence obtained by testing individual samples with multiple tests, using a conditional probability scheme
- **truePrevMulti2**: estimate true prevalence from apparent prevalence obtained by testing individual samples with multiple tests, using a covariance scheme
- **truePrevPools**: estimate true prevalence from apparent prevalence obtained by testing pooled samples
- **betaPERT**: calculate the parameters of a Beta-PERT distribution
- **betaExpert**: calculate the parameters of a Beta distribution based on expert opinion

Examples

```r
## Taenia solium cysticercosis in Nepal
## 142 positives out of 742 pigs sampled

## Model SE and SP based on literature data
## Sensitivity ranges uniformly between 60% and 100%
## Specificity ranges uniformly between 75% and 100%
## BUGS-style:
truePrev(x = 142, n = 742,
         SE = ~dunif(0.60, 1.00), SP = ~dunif(0.75, 1.00))

## list-style:
SE <- list(dist = "uniform", min = 0.60, max = 1.00)
SP <- list(dist = "uniform", min = 0.75, max = 1.00)
truePrev(x = 142, n = 742, SE = SE, SP = SP)

## Model SE and SP based on expert opinions
## Sensitivity lies in between 60% and 100%; most likely value is 90%
## Specificity is with 95% confidence larger than 75%; most likely value is 90%
## BUGS-style:
truePrev(x = 142, n = 742,
         SE = ~dunif(0.60, m = 0.90, b = 1.00),
         SP = ~dunif(0.75, mode = 0.90, lower = 0.75, p = 0.95))

## list-style:
SE <- list(dist = "beta-expert", mode = 0.90, lower = 0.75, p = 0.95)
SP <- list(dist = "beta-expert", mode = 0.90, lower = 0.75, p = 0.95)
truePrev(x = 142, n = 742, SE = SE, SP = SP)

## Model SE and SP as fixed values (each 90%)
truePrev(x = 142, n = 742, SE = 0.90, SP = 0.90)
```

---

**truePrevMulti**

*Estimate true prevalence from individuals samples using multiple tests – conditional probability scheme*

Description

Bayesian estimation of true prevalence from apparent prevalence obtained by applying multiple tests to individual samples. **truePrevMulti** implements the approach described by Berkvens et al. (2006), which uses a multinomial distribution to model observed test results, and in which conditional dependence between tests is modelled through conditional probabilities.
truePrevMulti

Usage

truePrevMulti(x, n, prior, nchains = 2, burnin = 10000, update = 10000, verbose = FALSE)

Arguments

x  Vector of apparent test results; see 'Details' below
n  The total sample size
prior  The prior distribution for theta; see 'Details' below
nchains  The number of chains used in the estimation process; must be ≥ 2
burnin  The number of discarded model iterations; defaults to 10,000
update  The number of withheld model iterations; defaults to 10,000
verbose  Logical flag, indicating if JAGS process output should be printed to the R console; defaults to FALSE

Details

turePrevMulti calls on JAGS via the rjags package to estimate true prevalence from apparent prevalence in a Bayesian framework. truePrevMulti fits a multinomial model to the apparent test results obtained by testing individual samples with a given number of tests. To see the actual fitted model, see the model slot of the prev-object.

The vector of apparent tests results, x, must contain the number of samples corresponding to each combination of test results. To see how this vector is defined for the number of tests h at hand, use define_x.

The prior in the multinomial model consists of a vector theta, which holds values for the true prevalence (TP), the sensitivity and specificity of the first test (SE1, SP1), and the conditional dependencies between the results of the subsequent tests and the preceding one(s). To see how this vector is defined for the number of tests n at hand, use define_prior.

The values of prior can be specified in two ways, referred to as BUGS-style and list-style, respectively. See also below for some examples.

For BUGS-style specification, the values of theta should be given between curly brackets (i.e., {}), separated by line breaks. theta values can be specified to be deterministic (i.e., fixed), using the ~ operator, or stochastic, using the ~ operator. In the latter case, the following distributions can be used:

- **Uniform:** dunif(min, max)
- **Beta:** dbeta(alpha, beta)
- **Beta-PERT:** dpert(min, mode, max)

Alternatively, theta values can be specified in a named list() as follows:

- **Fixed:** list(dist = "fixed", par)
• **Uniform**: list(dist = "uniform", min, max)
• **Beta**: list(dist = "beta", alpha, beta)
• **Beta-PERT**: list(dist = "pert", method, a, m, b, k)
  - 'method' must be "classic" or "vose";
  - 'a' denotes the pessimistic (minimum) estimate, 'm' the most likely estimate, and 'b' the optimistic (maximum) estimate;
  - 'k' denotes the scale parameter.
  See [betaPERT](#) for more information on Beta-PERT parameterization.
• **Beta-Expert**: list(dist = "beta-expert", mode, mean, lower, upper, p)
  - 'mode' denotes the most likely estimate, 'mean' the mean estimate;
  - 'lower' denotes the lower bound, 'upper' the upper bound;
  - 'p' denotes the confidence level of the expert.
  Only mode or mean should be specified; lower and upper can be specified together or alone.
  See [betaExpert](#) for more information on Beta-Expert parameterization.

**Value**

An object of class `prev`.

**Note**

Markov chain Monte Carlo sampling in `truePrevMulti` is performed by [JAGS (Just Another Gibbs Sampler)](http://sourceforge.net/projects/mcmc-jags/) through the `rjags` package. JAGS can be downloaded from [http://sourceforge.net/projects/mcmc-jags/](http://sourceforge.net/projects/mcmc-jags/).

**Author(s)**

<Brecht.Devleesschauwer@UGent.be>

**References**


**See Also**

`define_x`: how to define the vector of apparent test results `x`
`define_prior`: how to define the vector of theta values in `prior`

[CODA](#) for various functions that can be applied to the `prev@mcmc` object
`truePrevmultiR`: estimate true prevalence from apparent prevalence obtained by testing individual samples with multiple tests, using a covariance scheme
truePrevMulti

**truePrev**: estimate true prevalence from apparent prevalence obtained by testing *individual* samples with a single test

**truePrevPools**: estimate true prevalence from apparent prevalence obtained by testing *pooled* samples

**betaPERT**: calculate the parameters of a Beta-PERT distribution

**betaExpert**: calculate the parameters of a Beta distribution based on expert opinion

**Examples**

```r
## Not run:
## ============================================================= ##
## 2-TEST EXAMPLE: Campylobacter  
## ==============================================================
## Two tests were performed on 656 chicken meat samples  
## -> T1 = enrichment culture  
## -> T2 = direct plating  
## The following assumption were made:  
## -> TP is larger than 45% and smaller than 80%  
## -> SE1 must lie within 24% and 50%  
## -> SP1 and SP2 both equal 100%  
## -> beta(30, 12) describes P(T2+|D+,T1+)  
## The following results were obtained:  
## -> 113 samples T1+,T2+  
## -> 46 samples T1+,T2-  
## -> 156 samples T1-,T2+  
## -> 341 samples T1-,T2-  
## ==============================================================

## how is the 2-test model defined?  
define_x(2)  
define_prior(2)

campy <- truePrevMulti(  
  x = c(113, 46, 156, 341),  
  n = 656,  
  prior = {  
    theta[1] ~ dunif(0.45, 0.80)  
    theta[2] ~ dunif(0.24, 0.50)  
    theta[3] <- 1  
    theta[4] ~ dbeta(30, 12)  
    theta[5] ~ dbeta(1, 1)  
    theta[6] <- 1  
    theta[7] <- 1  
  }
)

## fit same model using 'list-style'  
campy <- truePrevMulti(  
  x = c(113, 46, 156, 341),
```

n = 656,  
_prior =  
_list(  
  theta1 = list(dist = "uniform", min = 0.45, max = 0.80),  
  theta2 = list(dist = "uniform", min = 0.24, max = 0.50),  
  theta3 = 1,  
  theta4 = list(dist = "beta", alpha = 30, beta = 12),  
  theta5 = list(dist = "beta", alpha = 1, beta = 1),  
  theta6 = 1,  
  theta7 = 1  
)  

## show model results  
campy  

## explore model structure  
str(campy)  
str(campy@par)  
str(campy@mcmc)  
campy@model  
campy@diagnostics  

## standard methods  
print(campy)  
summary(campy)  
par(mfrow = c(2, 2))  
plot(campy)  
plot(campy, "SE1")  
plot(campy, "SE2")  

## coda plots of TP, SE1, SE2  
par(mfrow = c(1, 3))  
densplot(campy, col = "red")  
traceplot(campy)  
gelman.plot(campy)  
autocorr.plot(campy)  

## -------------------------------  
## 3-TEST EXAMPLE: Giardia  
## -------------------------------  
## Three tests were performed on stools from 272 dogs  
## -> T1 = immunofluorescence assay  
## -> T2 = direct microscopy  
## -> T3 = SNAP immunochromatography  
## The following assumption were made:  
## -> TP is smaller than 20%  
## -> SE1 must be higher than 80%  
## -> SP1 must be higher than 90%  
## The following results were obtained:  
## -> 6 samples T1+, T2+, T3+  
## -> 4 samples T1+, T2+, T3-  

---
## truePrevMulti2

Estimate true prevalence from individuals samples using multiple tests

- covariance scheme

```r
## how is the 3-test model defined?
define_x(3)
define_prior(3)

## fit giardia 3-test model
giardia <-
truePrevMulti(
  x = c(6, 4, 12, 12, 1, 14, 3, 220),
n = 272,
prior = {
  theta[1] ~ dunif(0.00, 0.20)
  theta[2] ~ dunif(0.90, 1.00)
  theta[3] ~ dunif(0.80, 1.00)
  theta[4] ~ dbeta(1, 1)
  theta[5] ~ dbeta(1, 1)
  theta[6] ~ dbeta(1, 1)
  theta[7] ~ dbeta(1, 1)
  theta[8] ~ dbeta(1, 1)
  theta[9] ~ dbeta(1, 1)
  theta[10] ~ dbeta(1, 1)
  theta[11] ~ dbeta(1, 1)
  theta[12] ~ dbeta(1, 1)
  theta[13] ~ dbeta(1, 1)
  theta[14] ~ dbeta(1, 1)
  theta[15] ~ dbeta(1, 1)
}
)

## show model results
giardia
giardia
densplot(giardia, col = "red")

## End(Not run)
```
Description

Bayesian estimation of true prevalence from apparent prevalence obtained by applying multiple tests to individual samples. truePrevMulti2 implements and extends the approach described by Dendukuri and Joseph (2001), which uses a multinomial distribution to model observed test results, and in which conditional dependence between tests is modelled through covariances.

Usage

ttruePrevMulti2(x, n, prior, nchains = 2, burnin = 10000, update = 10000, verbose = FALSE)

Arguments

- x: Vector of apparent test results; see 'Details' below
- n: The total sample size
- prior: The prior distributions; see 'Details' below
- nchains: The number of chains used in the estimation process; must be \( \geq 2 \)
- burnin: The number of discarded model iterations; defaults to 10,000
- update: The number of withheld model iterations; defaults to 10,000
- verbose: Logical flag, indicating if JAGS process output should be printed to the R console; defaults to FALSE

Details

truePrevMulti2 calls on JAGS via the rjags package to estimate true prevalence from apparent prevalence in a Bayesian framework. truePrevMulti2 fits a multinomial model to the apparent test results obtained by testing individual samples with a given number of tests. To see the actual fitted model, see the model slot of the prev-object.

The vector of apparent tests results, x, must contain the number of samples corresponding to each combination of test results. To see how this vector is defined for the number of tests \( h \) at hand, use define_x.

Argument prior consists of prior distributions for:

- True Prevalence: TP
- SEnsitivity of each individual test: vector SE
- SPecificity of each individual test: vector SP
- Conditional covariance of all possible test combinations given a truly positive disease status: vector a
- Conditional covariance of all possible test combinations given a truly negative disease status: vector b

To see how prior is defined for the number of tests \( h \) at hand, use define_prior2.
The values of prior can be specified in two ways, referred to as BUGS-style and list-style, respectively. See also below for some examples.

For BUGS-style specification, the values of prior should be given between curly brackets (i.e., `{}`), separated by line breaks. Priors can be specified to be deterministic (i.e., fixed), using the `<-` operator, or stochastic, using the `~` operator. In the latter case, the following distributions can be used:

- **Uniform**: `dunif(min, max)`
- **Beta**: `dbeta(alpha, beta)`
- **Beta-PERT**: `dpert(min, mode, max)`

Alternatively, priors can be specified in a named list() as follows:

- **Fixed**: `list(dist = "fixed", par)`
- **Uniform**: `list(dist = "uniform", min, max)`
- **Beta**: `list(dist = "beta", alpha, beta)`
- **Beta-PERT**: `list(dist = "pert", method, a, m, b, k)`
  - 'method' must be "classic" or "vose";
  - 'a' denotes the pessimistic (minimum) estimate, 'm' the most likely estimate, and 'b' the optimistic (maximum) estimate;
  - 'k' denotes the scale parameter.

See `betaPERT` for more information on Beta-PERT parameterization.

- **Beta-Expert**: `list(dist = "beta-expert", mode, mean, lower, upper, p)`
  - 'mode' denotes the most likely estimate, 'mean' the mean estimate;
  - 'lower' denotes the lower bound, 'upper' the upper bound;
  - 'p' denotes the confidence level of the expert.

Only mode or mean should be specified; lower and upper can be specified together or alone. See `betaExpert` for more information on Beta-Expert parameterization.

**Value**

An object of class `prev`.

**Note**

Markov chain Monte Carlo sampling in `truePrevMulti2` is performed by JAGS (Just Another Gibbs Sampler) through the `rjags` package. JAGS can be downloaded from http://sourceforge.net/projects/mcmc-jags/.

**Author(s)**

Brecht Devleesschauwer <<brechtdv@gmail.com>>

**References**

- Dendukuri N, Joseph L (2001) Bayesian approaches to modeling the conditional dependence between multiple diagnostic tests. *Biometrics* 57:158-167
See Also

- `define_x`: how to define the vector of apparent test results \( x \)
- `define_prior2`: how to define prior

**coda** for various functions that can be applied to the \( \text{prev@mcmc} \) object

**truePrevMulti**: estimate true prevalence from apparent prevalence obtained by testing *individual* samples with multiple tests, using a conditional probability scheme

**truePrev**: estimate true prevalence from apparent prevalence obtained by testing *individual* samples with a single test

**truePrevPools**: estimate true prevalence from apparent prevalence obtained by testing *pooled* samples

**betaPERT**: calculate the parameters of a Beta-PERT distribution

**betaExpert**: calculate the parameters of a Beta distribution based on expert opinion

### Examples

```r
## Not run:
## =================================================================== ##
## 2-TEST EXAMPLE: Strongyloides
## =================================================================== ##
## Two tests were performed on 162 humans
## -> T1 = stool examination
## -> T2 = serology test
## Expert opinion generated the following priors:
## -> SE1 ~ dbeta(4.44, 13.31)
## -> SP1 ~ dbeta(71.25, 3.75)
## -> SE2 ~ dbeta(21.96, 5.49)
## -> SP2 ~ dbeta(4.10, 1.76)
## The following results were obtained:
## -> 38 samples T1+, T2+
## -> 2 samples T1+, T2-
## -> 87 samples T1-, T2+
## -> 35 samples T1-, T2-
## =================================================================== ##

## how is the 2-test model defined?
define_x(2)
define_prior2(2)

## fit Strongyloides 2-test model
## a first model assumes conditional independence
## -> set covariance terms to zero
trueyng indep <-
truePrevMulti2(
  x = c(38, 2, 87, 35),
  n = 162,
  prior = {
    TP ~ dbeta(1, 1)
    SE[1] ~ dbeta(4.44, 13.31)
    SP[1] ~ dbeta(71.25, 3.75)
  }
)`
truePrevMulti2

```
SP[2] ~ dbeta( 4.10, 1.76)
a[1] <- 0
b[1] <- 0
```

## show model results

```
strongy_indep
```

## fit same model using 'list-style'

```
strongy_indep <-
truePrevMulti2(
x = c(38, 2, 87, 35),
n = 162,
prior =
   list(
      TP = list(dist = "beta", alpha = 1, beta = 1),
      SE1 = list(dist = "beta", alpha = 4.44, beta = 13.31),
      SP1 = list(dist = "beta", alpha = 71.25, beta = 3.75),
      SE2 = list(dist = "beta", alpha = 21.96, beta = 5.49),
      SP2 = list(dist = "beta", alpha = 4.10, beta = 1.76),
      a1 = 0,
      b1 = 0
   )
)

## show model results

```
strongy_indep
```

## fit Strongyloides 2-test model

```
## a second model allows for conditional dependence
## -> a[1] is the covariance between T1 and T2, given D+
## -> b[1] is the covariance between T1 and T2, given D-
## -> a[1] and b[1] can range between +/- 2^-h, ie, (-.25, .25)
```

```
strongy <-
truePrevMulti2(
x = c(38, 2, 87, 35),
n = 162,
prior = {
   TP ~ dbeta(1, 1)
   SE[1] ~ dbeta( 4.44, 13.31)
   SP[1] ~ dbeta(71.25, 3.75)
   SP[2] ~ dbeta( 4.10, 1.76)
   a[1] ~ dunif(-0.25, 0.25)
   b[1] ~ dunif(-0.25, 0.25)
}
)

## explore model structure

```
str(strongy)     # overall structure
str(strongy@par) # structure of slot 'par'
str(strongy@mcmc) # structure of slot 'mcmc'
strongy$model    # fitted model
strongy@diagnostics # DIC, BGR and Bayes-P values
```
## Standard Methods

```r
# Standard methods
print(strongy)
summary(strongy)
par(mfrow = c(2, 2))
plot(strongy) # shows plots of TP by default
plot(strongy, "SE[1]") # same plots for SE1
plot(strongy, "SE[2]") # same plots for SE2
plot(strongy, "SP[1]") # same plots for SP1
plot(strongy, "SP[2]") # same plots for SP2
plot(strongy, "a[1]") # same plots for a[1]
plot(strongy, "b[1]") # same plots for b[1]

# Coda plots of all parameters
par(mfrow = c(2, 4)); densplot(strongy, col = "red")
par(mfrow = c(2, 4)); traceplot(strongy)
par(mfrow = c(2, 4)); gelman.plot(strongy)
par(mfrow = c(2, 4)); autocorr.plot(strongy)
```

## truePrevPools

### Description

Bayesian estimation of true prevalence from apparent prevalence obtained by testing pooled samples.

### Usage

```r
truePrevPools(x, n, SE = 1, SP = 1, prior = c(1, 1),
               nchains = 2, burnin = 10000, update = 10000,
               verbose = FALSE)
```

### Arguments

- `x` The vector of indicator variables, indicating whether a pool was positive ("1") or negative ("0")
- `n` The vector of pool sizes
- `SE`, `SP` The prior distribution for sensitivity (SE) and specificity (SP); see 'Details' below for specification of these distributions
- `prior` The parameters of the prior Beta distribution for true prevalence; defaults to `c(1, 1)`
- `nchains` The number of chains used in the estimation process; `nchains` must be ≥ 2
- `burnin` The number of discarded model iterations; defaults to 10,000
- `update` The number of withheld model iterations; defaults to 10,000
- `verbose` Logical flag, indicating if JAGS process output should be printed to the R console; defaults to `FALSE`
truePrevPools calls on JAGS/rjags to estimate the true prevalence from the apparent prevalence in a Bayesian framework. The default model, in BUGS language, is given below. To see the actual fitted model, see the model slot of the prev-object.

```r
model {
  for (i in 1:N) {
    x[i] ~ dbern(AP[i])
    AP[i] <- SEpool[i] * (1 - pow(1 - TP, n[i])) + (1 - SPpool[i]) * pow(1 - TP, n[i])
    SEpool[i] <- 1 - (pow(1 - SE, n[i] * TP) * pow(SP, n[i] * (1 - TP)))
    SPpool[i] <- pow(SP, n[i])
  }
  # SE ~ user-defined (see below)
  # SP ~ user-defined (see below)
  TP ~ dbeta(prior[1], prior[2])
}
```

The test sensitivity (SE) and specificity (SP) can be specified by the user, independently, as one of "fixed", "uniform", "beta", "pert", or "beta-expert", with "fixed" as the default. Note that SE and SP must correspond to the test characteristics for testing individual samples; truePrevPools will calculate SEpool and SPpool, the sensitivity and specificity for testing pooled samples, based on Boelaert et al. (2000).

Distribution parameters can be specified in a named `list()` as follows:

- **Fixed**: `list(dist = "fixed", par)
- **Uniform**: `list(dist = "uniform", min, max)
- **Beta**: `list(dist = "beta", alpha, beta)
- **PERT**: `list(dist = "pert", method, a, m, b, k)
  'method' must be "classic" or "vose";
  'a' denotes the pessimistic (minimum) estimate, 'm' the most likely estimate, and 'b' the optimistic (maximum) estimate;
  'k' denotes the scale parameter.
  See `betaPERT` for more information on Beta-PERT parametrization.
- **Beta-Expert**: `list(dist = "beta-expert", mode, mean, lower, upper, p)`
  'mode' denotes the most likely estimate, 'mean' the mean estimate;
  'lower' denotes the lower bound, 'upper' the upper bound;
  'p' denotes the confidence level of the expert.
  Only mode or mean should be specified; lower and upper can be specified together or alone.
  See `betaExpert` for more information on Beta-Expert parameterization.

For Uniform, Beta and Beta-PERT distributions, BUGS-style short-hand notation is also allowed:

- **Uniform**: `~dunif(min, max)
- **Beta**: `~dbeta(alpha, beta)
- **Beta-PERT**: `~dpert(min, mode, max)
Value

An object of class `prev`.

Note

Markov chain Monte Carlo sampling in `truePrevPools` is performed by **JAGS (Just Another Gibbs Sampler)** through the `rjags` package. JAGS can be downloaded from [http://sourceforge.net/projects/mcmc-jags/](http://sourceforge.net/projects/mcmc-jags/).

Author(s)

Brecht Devleesschauwer <<brechtdv@gmail.com>>

References


See Also

- **coda** for various functions that can be applied to the `prev@mcmc` object
- **truePrev**: estimate true prevalence from apparent prevalence obtained by testing *individual* samples with a single test
- **truePrevMulti**: estimate true prevalence from apparent prevalence obtained by testing *individual* samples with multiple tests, using a conditional probability scheme
- **truePrevMulti2**: estimate true prevalence from apparent prevalence obtained by testing *individual* samples with multiple tests, using a covariance scheme
- **betaPERT**: calculate the parameters of a Beta-PERT distribution
- **betaExpert**: calculate the parameters of a Beta distribution based on expert opinion

Examples

```r
## Sandflies in Aurabani, Nepal, 2007
pool_results <- c(0, 0, 0, 0, 0, 0, 0, 1, 0)
pool_sizes <- c(2, 1, 6, 10, 1, 7, 1, 4, 1, 3)

## Sensitivity ranges uniformly between 60% and 95%
## Specificity is considered to be 100%

# BUGS-style:
truePrevPools(x = pool_results, n = pool_sizes,
   SE = dunif(0.60, 0.95), SP = 1)

# list-style:
SE <- list(dist = "uniform", min = 0.60, max = 0.95)
truePrevPools(x = pool_results, n = pool_sizes,
   ...)
```
truePrevPools

\[ SE = SE, \ SP = 1 \]
# Index

<table>
<thead>
<tr>
<th>Topic</th>
<th>Data</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expert</td>
<td>betaExpert, 3</td>
<td>betaPERT, 5</td>
</tr>
<tr>
<td>PERT</td>
<td>betaPERT, 5</td>
<td></td>
</tr>
<tr>
<td>classes</td>
<td>prev-class, 13</td>
<td></td>
</tr>
<tr>
<td>confidence interval</td>
<td>propCI, 15</td>
<td></td>
</tr>
<tr>
<td>methods</td>
<td>convert-methods, 7</td>
<td>plot-methods, 11</td>
</tr>
<tr>
<td></td>
<td>plot-methods-coda, 12</td>
<td>print-methods, 14</td>
</tr>
<tr>
<td></td>
<td>show-methods, 17</td>
<td>summary-methods, 17</td>
</tr>
<tr>
<td>package</td>
<td>prevalence-package, 2</td>
<td></td>
</tr>
<tr>
<td>prevalence</td>
<td>propCI, 15</td>
<td></td>
</tr>
<tr>
<td>as.matrix</td>
<td>prev-method</td>
<td>convert-methods, 7</td>
</tr>
<tr>
<td></td>
<td>(convert-methods), 7</td>
<td></td>
</tr>
<tr>
<td>autocorr.plot</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>autocorr.plot.prev-method</td>
<td>(plot-methods-coda), 12</td>
<td></td>
</tr>
<tr>
<td>autocorr.plot.methods</td>
<td>(plot-methods-coda), 12</td>
<td></td>
</tr>
<tr>
<td>autocorrelation</td>
<td>11, 12</td>
<td></td>
</tr>
<tr>
<td>betaExpert</td>
<td>2, 3, 4, 7, 19, 20, 22, 23, 27, 28, 31, 32</td>
<td></td>
</tr>
<tr>
<td>betaPERT</td>
<td>2, 4, 5, 19, 20, 22, 23, 27, 28, 31, 32</td>
<td></td>
</tr>
<tr>
<td>Brooks-Gelman-Rubin</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Brooks-Gelman-Rubin statistic</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>coda</td>
<td>20, 22, 28, 32</td>
<td></td>
</tr>
<tr>
<td>define</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>define_prior</td>
<td>9, 21, 22</td>
<td>define_prior (define), 8</td>
</tr>
<tr>
<td></td>
<td>define_prior2 (define), 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>define_x, 21, 22, 26, 28</td>
<td>define_x (define), 8</td>
</tr>
<tr>
<td></td>
<td>density, 11, 12</td>
<td></td>
</tr>
<tr>
<td>densplot</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>densplot.prev-method</td>
<td>(plot-methods-coda), 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(plot-methods-coda), 12</td>
<td></td>
</tr>
<tr>
<td>gelman.diag</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>gelman.plot</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>gelman.plot.prev-method</td>
<td>(plot-methods-coda), 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(plot-methods-coda), 12</td>
<td></td>
</tr>
<tr>
<td>matrix</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>optimize</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>plot.prev, ANY-method</td>
<td>(plot-methods), 11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>plot-methods, 11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>plot-methods-coda, 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>plot.betaExpert (betaExpert), 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>plot.betaPERT (betaPERT), 5</td>
<td></td>
</tr>
<tr>
<td>prev</td>
<td>18, 19, 21, 22, 26, 27, 31, 32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>prev-class, 13</td>
<td></td>
</tr>
<tr>
<td>prevalence</td>
<td>prevalence-package, 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>prevalence-package, 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>print.prev-method (print-methods), 14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>print-methods, 14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>print.betaExpert (betaExpert), 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>print.betaPERT (betaPERT), 5</td>
<td></td>
</tr>
<tr>
<td>propCI</td>
<td>2, 15</td>
<td></td>
</tr>
<tr>
<td>rjags</td>
<td>2, 18, 19, 21, 22, 26, 27, 31, 32</td>
<td></td>
</tr>
</tbody>
</table>
INDEX

show.prev-method (show-methods), 17
show-methods, 17
summary.prev-method (summary-methods), 17
summary-methods, 17

trace, 11, 12
traceplot, 12
traceplot.prev-method
  (plot-methods-coda), 12
traceplot-methods (plot-methods-coda), 12
truePrev, 2, 13, 18, 23, 28, 32
truePrevMulti, 2, 8, 9, 13, 14, 20, 20, 21, 28, 32
truePrevMulti2, 2, 8, 9, 13, 20, 22, 25, 26, 32
truePrevPools, 2, 13, 20, 23, 28, 30