Package 'updog'

January 24, 2022

Title Flexible Genotyping for Polyploids

Version 2.1.2

Description Implements empirical Bayes approaches to genotype polyploids from next generation sequencing data while accounting for allele bias, overdispersion, and sequencing error. The main functions are flexdog() and multidog(), which allow the specification of many different genotype distributions. Also provided are functions to simulate genotypes, rgeno(), and read-counts, rflexdog(), as well as functions to calculate oracle genotyping error rates, oracle_mis(), and correlation with the true genotypes, oracle_cor(). These latter two functions are useful for read depth calculations. Run browseVignettes(package = ``updog'') in R for example usage. See Gerard et al. (2018) <doi:10.1534/genetics.118.301468> and Gerard and Ferrao (2020) <doi:10.1093/bioinformatics/btz852> for details on the implemented methods.

Depends R (>= 3.4.0)

License GPL-3

BugReports https://github.com/dcgerard/updog/issues

URL https://github.com/dcgerard/updog/

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp (>= 0.12.16), RcppArmadillo, assertthat, ggplot2, ggthemes, reshape2, foreach, future, doFuture, doRNG, methods, iterators

Suggests covr, testthat, SuppDists, knitr, rmarkdown, VariantAnnotation, GenomicRanges, S4Vectors, IRanges

VignetteBuilder knitr

NeedsCompilation yes

Author David Gerard [aut, cre] (<https://orcid.org/0000-0001-9450-5023>)

Maintainer David Gerard <gerard.1787@gmail.com>

Repository CRAN

Date/Publication 2022-01-24 21:50:02 UTC

R topics documented:

updog-package . . . 3
dbetabinom . . . 4
export_vcf . . . 6
filter_snp . . . 7
flexdog . . . 8
flexdog_full . . . 13
format_multidog . . . 19
get_q_array . . . 19
is.flexdog . . . 20
is.multidog . . . 21
log_sum_exp . . . 21
log_sum_exp_2 . . . 22
multidog . . . 22
oracle_cor . . . 26
oracle_cor_from_joint . . . 27
oracle_joint . . . 29
oracle_mis . . . 30
oracle_mis_from_joint . . . 32
oracle_mis_vec . . . 33
oracle_mis_vec_from_joint . . . 34
oracle_plot . . . 35
plot.flexdog . . . 36
plot.multidog . . . 37
plot_geno . . . 38
rflexdog . . . 40
rgeno . . . 41
snpdat . . . 43
uitdewilligen . . . 44
wem . . . 45

Index **47**

updog-package updog *Flexible Genotyping for Polyploids*

Description

Implements empirical Bayes approaches to genotype polyploids from next generation sequencing data while accounting for allele bias, overdispersion, and sequencing error. The main functions are `flexdog()` and `multidog()`, which allow the specification of many different genotype distributions. Also provided are functions to simulate genotypes, `rgeno()`, and read-counts, `rflexdog()`, as well as functions to calculate oracle genotyping error rates, `oracle_mis()`, and correlation with the true genotypes, `oracle_cor()`. These latter two functions are useful for read depth calculations. Run `browseVignettes(package = "updog")` in R for example usage. See Gerard et al. (2018) <doi: 10.1534/genetics.118.301468> and Gerard and Ferrao (2020) <doi: 10.1093/bioinformatics/btz852> for details on the implemented methods.

Details

The package is named updog for "Using Parental Data for Offspring Genotyping" because we originally developed the method for full-sib populations, but it works now for more general populations.

Our best competitor is probably the `fitPoly` package, which you can check out at `https://cran.r-project.org/package=fitPoly`. Though, we think that updog returns better calibrated measures of uncertainty when you have next-generation sequencing data.

If you find a bug or want an enhancement, please submit an issue at `https://github.com/dcgerard/updog/issues`.

updog Functions

`flexdog()` The main function that fits an empirical Bayes approach to genotype polyploids from next generation sequencing data.

`multidog()` A convenience function for running `flexdog()` over many SNPs. This function provides support for parallel computing.

`format_multidog()` Return arrayicized elements from the output of `multidog()`.

`filter_snp()` Filter SNPs based on the output of `multidog()`

`rgeno()` simulate the genotypes of a sample from one of the models allowed in `flexdog()`.

`rflexdog()` Simulate read-counts from the `flexdog()` model.

`plot.flexdog()` Plotting the output of `flexdog()`.

`plot.multidog()` Plotting the output of `multidog()`.

`oracle_joint()` The joint distribution of the true genotype and an oracle estimator.

`oracle_plot()` Visualize the output of `oracle_joint()`.

`oracle_mis()` The oracle misclassification error rate (Bayes rate).

`oracle_cor()` Correlation between the true genotype and the oracle estimated genotype.

updog **Datasets**

snpdat A small example dataset for using flexdog.

uitdewilligen A small example dataset

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.
- Gerard, David, and Luís Felipe Ventorim Ferrão. "Priors for genotyping polyploids." Bioinformatics 36, no. 6 (2020): 1795-1800. doi: 10.1093/bioinformatics/btz852.

---

dbetabinom *The Beta-Binomial Distribution*

Description

Density, distribution function, quantile function and random generation for the beta-binomial distribution when parameterized by the mean mu and the overdispersion parameter rho rather than the typical shape parameters.

Usage

```
dbetabinom(x, size, mu, rho, log)

pbetabinom(q, size, mu, rho, log_p)

qbetabinom(p, size, mu, rho)

rbetabinom(n, size, mu, rho)
```

Arguments

x, q	A vector of quantiles.
size	A vector of sizes.
mu	Either a scalar of the mean for each observation, or a vector of means of each observation, and thus the same length as x and size. This must be between 0 and 1.
rho	Either a scalar of the overdispersion parameter for each observation, or a vector of overdispersion parameters of each observation, and thus the same length as x and size. This must be between 0 and 1.

log, log_p	A logical vector either of length 1 or the same length as x and size. This determines whether to return the log probabilities for all observations (in the case that its length is 1) or for each observation (in the case that its length is that of x and size).
p	A vector of probabilities.
n	The number of observations.

Details

Let μ and ρ be the mean and overdispersion parameters. Let α and β be the usual shape parameters of a beta distribution. Then we have the relation

$$\mu = \alpha/(\alpha + \beta),$$

and

$$\rho = 1/(1 + \alpha + \beta).$$

This necessarily means that

$$\alpha = \mu(1 - \rho)/\rho,$$

and

$$\beta = (1 - \mu)(1 - \rho)/\rho.$$

Value

Either a random sample (rbetabinom), the density (dbetabinom), the tail probability (pbetabinom), or the quantile (qbetabinom) of the beta-binomial distribution.

Functions

- dbetabinom: Density function.
- pbetabinom: Distribution function.
- qbetabinom: Quantile function.
- rbetabinom: Random generation.

Author(s)

David Gerard

Examples

```
x <- rbetabinom(n = 10, size = 10, mu = 0.1, rho = 0.01)
dbetabinom(x = 1, size = 10, mu = 0.1, rho = 0.01, log = FALSE)
pbetabinom(q = 1, size = 10, mu = 0.1, rho = 0.01, log_p = FALSE)
qbetabinom(p = 0.6, size = 10, mu = 0.1, rho = 0.01)
```

export_vcf *Save the output of* multidog() *to a VCF file.*

Description

This is an experimental function to save the output of multidog() to a VCF file.

Usage

```
export_vcf(obj, filename)
```

Arguments

obj	An object of class multidog().
filename	A string, the path to save the VCF file.

Details

This function uses the Bioconductor packages VariantAnnotation, GenomicRanges, S4Vectors, and IRanges. You can install these using the BiocManager package via:

```
install.packages("BiocManager")
```

```
BiocManager::install(c("VariantAnnotation","GenomicRanges","S4Vectors","IRanges"))
```

To read more about the VCF format, see the official documentation on the Samtools website: https://samtools.github.io/hts-specs/.

Examples

```
## Not run:
library(VariantAnnotation)
data("snpdat")
refmat <- reshape2::acast(data = snpdat,
                          formula = snp ~ id,
                          value.var = "counts")
sizemat <- reshape2::acast(data = snpdat,
                           formula = snp ~ id,
                           value.var = "size")
mout <- multidog(refmat = refmat,
                 sizemat = sizemat,
                 ploidy = 6,
                 model = "s1",
                 p1_id = "Xushu18")
export_vcf(obj = mout, filename = "./sweet_potato.vcf")
spvcf <- readVcf("./sweet_potato.vcf")

## End(Not run)
```

filter_snp *Filter SNPs based on the output of* multidog().

Description

Filter based on provided logical predicates in terms of the variable names in x$snpdf. This function filters both x$snpdf and x$inddf.

Usage

```
filter_snp(x, expr)
```

Arguments

x	The output of multidog.
expr	Logical predicate expression defined in terms of the variables in x$snpdf. Only SNPs where the condition evaluates to TRUE are kept.

Author(s)

David Gerard

See Also

- multidog():For the variables in x$snpdf which you can filter by.

Examples

```
## Not run:
data("uitdewilligen")
mout <- multidog(refmat = t(uitdewilligen$refmat),
                 sizemat = t(uitdewilligen$sizemat),
                 ploidy = uitdewilligen$ploidy,
                 nc = 2)

## The following filters are for educational purposes only and should
## not be taken as a default filter:
mout2 <- filter_snp(mout, bias < 0.8 & od < 0.003)

## End(Not run)
```

flexdog *Flexible genotyping for polyploids from next-generation sequencing data.*

Description

Genotype polyploid individuals from next generation sequencing (NGS) data while assuming the genotype distribution is one of several forms. `flexdog` does this while accounting for allele bias, overdispersion, sequencing error. The method is described in detail in Gerard et. al. (2018) and Gerard and Ferrão (2020). See `multidog()` for running flexdog on multiple SNPs in parallel.

Usage

```
flexdog(
  refvec,
  sizevec,
  ploidy,
  model = c("norm", "hw", "bb", "s1", "s1pp", "f1", "f1pp", "flex", "uniform",
    "custom"),
  p1ref = NULL,
  p1size = NULL,
  p2ref = NULL,
  p2size = NULL,
  snpname = NULL,
  bias_init = exp(c(-1, -0.5, 0, 0.5, 1)),
  verbose = TRUE,
  prior_vec = NULL,
  ...
)
```

Arguments

`refvec`	A vector of counts of reads of the reference allele.
`sizevec`	A vector of total counts.
`ploidy`	The ploidy of the species. Assumed to be the same for each individual.
`model`	What form should the prior (genotype distribution) take? See Details for possible values.
`p1ref`	The reference counts for the first parent if `model = "f1"` or `model = "f1pp"`, or for the only parent if `model = "s1"` or `model = "s1pp"`.
`p1size`	The total counts for the first parent if `model = "f1"` or `model = "f1pp"`, or for the only parent if `model = "s1"` or `model = "s1pp"`.
`p2ref`	The reference counts for the second parent if `model = "f1"` or `model = "f1pp"`.
`p2size`	The total counts for the second parent if `model = "f1"` or `model = "f1pp"`.
`snpname`	A string. The name of the SNP under consideration. This is just returned in the `input` list for your reference.

bias_init	A vector of initial values for the bias parameter over the multiple runs of flexdog_full().
verbose	Should we output more (TRUE) or less (FALSE)?
prior_vec	The pre-specified genotype distribution. Only used if model = "custom" and must otherwise be NULL. If specified, then it should be a vector of length ploidy + 1 with non-negative elements that sum to 1.
...	Additional parameters to pass to flexdog_full().

Details

Possible values of the genotype distribution (values of model) are:

"norm" A distribution whose genotype frequencies are proportional to the density value of a normal with some mean and some standard deviation. Unlike the "bb" and "hw" options, this will allow for distributions both more and less dispersed than a binomial. This seems to be the most robust to violations in modeling assumptions, and so is the default. This prior class was developed in Gerard and Ferrão (2020).

"hw" A binomial distribution that results from assuming that the population is in Hardy-Weinberg equilibrium (HWE). This actually does pretty well even when there are minor to moderate deviations from HWE. Though it does not perform as well as the ‘"norm"‘ option when there are severe deviations from HWE.

"bb" A beta-binomial distribution. This is an overdispersed version of "hw" and can be derived from a special case of the Balding-Nichols model.

"s1" This prior assumes the individuals are all full-siblings resulting from one generation of selfing. I.e. there is only one parent. This model assumes a particular type of meiotic behavior: polysomic inheritance with bivalent, non-preferential pairing.

"f1" This prior assumes the individuals are all full-siblings resulting from one generation of a bi-parental cross. This model assumes a particular type of meiotic behavior: polysomic inheritance with bivalent, non-preferential pairing.

"f1pp" This prior allows for double reduction and preferential pairing in an F1 population of tetraploids.

"s1pp" This prior allows for double reduction and preferential pairing in an S1 population of tetraploids.

"flex" Generically any categorical distribution. Theoretically, this works well if you have a lot of individuals. In practice, it seems to be much less robust to violations in modeling assumptions.

"uniform" A discrete uniform distribution. This should never be used in practice.

"custom" A pre-specified prior distribution. You specify it using the prior_vec argument. You should almost never use this option in practice.

You might think a good default is model = "uniform" because it is somehow an "uninformative prior." But it is very informative and tends to work horribly in practice. The intuition is that it will estimate the allele bias and sequencing error rates so that the estimated genotypes are approximately uniform (since we are assuming that they are approximately uniform). This will usually result in unintuitive genotyping since most populations don't have a uniform genotype distribution. I include it as an option only for completeness. Please don't use it.

The value of prop_mis is a very intuitive measure for the quality of the SNP. prop_mis is the posterior proportion of individuals mis-genotyped. So if you want only SNPS that accurately genotype, say, 95% of the individuals, you could discard all SNPs with a prop_mis over 0.05.

The value of maxpostprob is a very intuitive measure for the quality of the genotype estimate of an individual. This is the posterior probability of correctly genotyping the individual when using geno (the posterior mode) as the genotype estimate. So if you want to correctly genotype, say, 95% of individuals, you could discard all individuals with a maxpostprob of under 0.95. However, if you are just going to impute missing genotypes later, you might consider not discarding any individuals as flexdog's genotype estimates will probably be more accurate than other more naive approaches, such as imputing using the grand mean.

In most datasets I've examined, allelic bias is a major issue. However, you may fit the model assuming no allelic bias by setting update_bias = FALSE and bias_init = 1.

Prior to using flexdog, during the read-mapping step, you could try to get rid of allelic bias by using WASP (doi: 10.1101/011221). If you are successful in removing the allelic bias (because its only source was the read-mapping step), then setting update_bias = FALSE and bias_init = 1 would be reasonable. You can visually inspect SNPs for bias by using plot_geno().

flexdog(), like most methods, is invariant to which allele you label as the "reference" and which you label as the "alternative". That is, if you set refvec with the number of alternative read-counts, then the resulting genotype estimates will be the estimated allele dosage of the alternative allele.

Value

An object of class flexdog, which consists of a list with some or all of the following elements:

bias The estimated bias parameter.

seq The estimated sequencing error rate.

od The estimated overdispersion parameter.

num_iter The number of EM iterations ran. You should be wary if this equals itermax.

llike The maximum marginal log-likelihood.

postmat A matrix of posterior probabilities of each genotype for each individual. The rows index the individuals and the columns index the allele dosage.

genologlike A matrix of genotype *log*-likelihoods of each genotype for each individual. The rows index the individuals and the columns index the allele dosage.

gene_dist The estimated genotype distribution. The ith element is the proportion of individuals with genotype i-1.

par A list of the final estimates of the parameters of the genotype distribution. The elements included in par depends on the value of model:

model = "norm": mu: The normal mean.

sigma: The normal standard deviation (not variance).

model = "hw": alpha: The major allele frequency.

model = "bb": alpha: The major allele frequency.

tau: The overdispersion parameter. See the description of rho in the Details of betabinom().

model = "s1": pgeno: The allele dosage of the parent.

`alpha`: The mixture proportion of the discrete uniform (included and fixed at a small value mostly for numerical stability reasons). See the description of `fs1_alpha` in `flexdog_full()`.

`model = "f1"`: `p1geno`: The allele dosage of the first parent.

`p2geno`: The allele dosage of the second parent.

`alpha`: The mixture proportion of the discrete uniform (included and fixed at a small value mostly for numerical stability reasons). See the description of `fs1_alpha` in `flexdog_full()`.

`model = "s1pp"`: `ell1`: The estimated dosage of the parent.

`tau1`: The estimated double reduction parameter of the parent. Available if `ell1` is 1, 2, or 3. Identified if `ell1` is 1 or 3.

`gamma1`: The estimated preferential pairing parameter. Available if `ell1` is 2. However, it is not returned in an identified form.

`alpha`: The mixture proportion of the discrete uniform (included and fixed at a small value mostly for numerical stability reasons). See the description of `fs1_alpha` in `flexdog_full()`.

`model = "f1pp"`: `ell1`: The estimated dosage of parent 1.

`ell2`: The estimated dosage of parent 2.

`tau1`: The estimated double reduction parameter of parent 1. Available if `ell1` is 1, 2, or 3. Identified if `ell1` is 1 or 3.

`tau2`: The estimated double reduction parameter of parent 2. Available if `ell2` is 1, 2, or 3. Identified if `ell2` is 1 or 3.

`gamma1`: The estimated preferential pairing parameter of parent 1. Available if `ell1` is 2. However, it is not returned in an identified form.

`gamma2`: The estimated preferential pairing parameter of parent 2. Available if `ell2` is 2. However, it is not returned in an identified form.

`alpha`: The mixture proportion of the discrete uniform (included and fixed at a small value mostly for numerical stability reasons). See the description of `fs1_alpha` in `flexdog_full()`.

`model = "flex"`: `par` is an empty list.

`model = "uniform"`: `par` is an empty list.

`model = "custom"`: `par` is an empty list.

`geno` The posterior mode genotype. These are your genotype estimates.

`maxpostprob` The maximum posterior probability. This is equivalent to the posterior probability of correctly genotyping each individual.

`postmean` The posterior mean genotype. In downstream association studies, you might want to consider using these estimates.

`input$refvec` The value of `refvec` provided by the user.

`input$sizevec` The value of `sizevec` provided by the user.

`input$ploidy` The value of `ploidy` provided by the user.

`input$model` The value of `model` provided by the user.

`input$p1ref` The value of `p1ref` provided by the user.

`input$p1size` The value of `p1size` provided by the user.

`input$p2ref` The value of `p2ref` provided by the user.

`input$p2size` The value of `p2size` provided by the user.

`input$snpname` The value of `snpname` provided by the user.

`prop_mis` The posterior proportion of individuals genotyped incorrectly.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.
- Gerard, David, and Luís Felipe Ventorim Ferrão. "Priors for genotyping polyploids." Bioinformatics 36, no. 6 (2020): 1795-1800. doi: 10.1093/bioinformatics/btz852.

See Also

Run `browseVignettes(package = "updog")` in R for example usage. Other useful functions include:

`multidog()` For running `flexdog()` on multiple SNPs in parallel.

`flexdog_full()` For additional parameter options when running `flexdog()`.

`rgeno()` For simulating genotypes under the allowable prior models in `flexdog()`.

`rflexdog()` For simulating read-counts under the assumed likelihood model in `flexdog()`.

`plot.flexdog()` For plotting the output of `flexdog()`.

`oracle_mis()` For calculating the oracle genotyping error rates. This is useful for read-depth calculations *before* collecting data. After you have data, using the value of `prop_mis` is better.

`oracle_cor()` For calculating the correlation between the true genotypes and an oracle estimator (useful for read-depth calculations *before* collecting data).

Examples

```
## An S1 population where the first individual
## is the parent.
data("snpdat")
ploidy  <- 6
refvec  <- snpdat$counts[snpdat$snp == "SNP2"]
sizevec <- snpdat$size[snpdat$snp == "SNP2"]
fout    <- flexdog(refvec   = refvec[-1],
                   sizevec  = sizevec[-1],
                   ploidy   = ploidy,
                   model    = "s1",
                   p1ref    = refvec[1],
                   p1size   = sizevec[1])
plot(fout)
```

```
## A natural population. We will assume a
## normal prior since there are so few
## individuals.
data("uitdewilligen")
ploidy  <- 4
refvec  <- uitdewilligen$refmat[, 1]
sizevec <- uitdewilligen$sizemat[, 1]
fout    <- flexdog(refvec  = refvec,
                   sizevec = sizevec,
                   ploidy  = ploidy,
                   model   = "norm")
plot(fout)
```

`flexdog_full` *Flexible genotyping for polyploids from next-generation sequencing data.*

Description

Genotype polyploid individuals from next generation sequencing (NGS) data while assuming the genotype distribution is one of several forms. `flexdog_full()` does this while accounting for allele bias, overdispersion, and sequencing error. This function has more options than `flexdog` and is only meant for expert users. The method is described in detail in Gerard et. al. (2018) and Gerard and Ferrão (2020).

Usage

```
flexdog_full(
  refvec,
  sizevec,
  ploidy,
  model = c("norm", "hw", "bb", "s1", "s1pp", "f1", "f1pp", "flex", "uniform",
    "custom"),
  verbose = TRUE,
  mean_bias = 0,
  var_bias = 0.7^2,
  mean_seq = -4.7,
  var_seq = 1,
  mean_od = -5.5,
  var_od = 0.5^2,
  seq = 0.005,
  bias = 1,
  od = 0.001,
```

```
  update_bias = TRUE,
  update_seq = TRUE,
  update_od = TRUE,
  itermax = 200,
  tol = 10^-4,
  fs1_alpha = 10^-3,
  p1ref = NULL,
  p1size = NULL,
  p2ref = NULL,
  p2size = NULL,
  snpname = NULL,
  prior_vec = NULL,
  seq_upper = 0.05
)
```

Arguments

refvec	A vector of counts of reads of the reference allele.
sizevec	A vector of total counts.
ploidy	The ploidy of the species. Assumed to be the same for each individual.
model	What form should the prior (genotype distribution) take? See Details for possible values.
verbose	Should we output more (TRUE) or less (FALSE)?
mean_bias	The prior mean of the log-bias.
var_bias	The prior variance of the log-bias.
mean_seq	The prior mean of the logit of the sequencing error rate.
var_seq	The prior variance of the logit of the sequencing error rate.
mean_od	The prior mean of the logit of the overdispersion parameter.
var_od	The prior variance of the logit of the overdispersion parameter.
seq	The starting value of the sequencing error rate.
bias	The starting value of the bias.
od	The starting value of the overdispersion parameter.
update_bias	A logical. Should we update bias (TRUE), or not (FALSE)?
update_seq	A logical. Should we update seq (TRUE), or not (FALSE)?
update_od	A logical. Should we update od (TRUE), or not (FALSE)?
itermax	The total number of EM iterations to run.
tol	The tolerance stopping criterion. The EM algorithm will stop if the difference in the log-likelihoods between two consecutive iterations is less than tol.
fs1_alpha	The value at which to fix the mixing proportion for the uniform component when model = "f1", model = "s1", model = "f1pp", or model = "s1pp". I would recommend some small value such as 10^-3.
p1ref	The reference counts for the first parent if model = "f1" or model = "f1pp", or for the only parent if model = "s1" or model = "s1pp".

p1size	The total counts for the first parent if model = "f1" or model = "f1pp", or for the only parent if model = "s1" or model = "s1pp".
p2ref	The reference counts for the second parent if model = "f1" or model = "f1pp".
p2size	The total counts for the second parent if model = "f1" or model = "f1pp".
snpname	A string. The name of the SNP under consideration. This is just returned in the input list for your reference.
prior_vec	The pre-specified genotype distribution. Only used if model = "custom" and must otherwise be NULL. If specified, then it should be a vector of length ploidy + 1 with non-negative elements that sum to 1.
seq_upper	The upper bound on the possible sequencing error rate. Default is 0.05, but you should adjust this if you have prior knowledge on the sequencing error rate of the sequencing technology.

Details

Possible values of the genotype distribution (values of model) are:

"norm" A distribution whose genotype frequencies are proportional to the density value of a normal with some mean and some standard deviation. Unlike the "bb" and "hw" options, this will allow for distributions both more and less dispersed than a binomial. This seems to be the most robust to violations in modeling assumptions, and so is the default. This prior class was developed in Gerard and Ferrão (2020).

"hw" A binomial distribution that results from assuming that the population is in Hardy-Weinberg equilibrium (HWE). This actually does pretty well even when there are minor to moderate deviations from HWE. Though it does not perform as well as the ‘"norm"‘ option when there are severe deviations from HWE.

"bb" A beta-binomial distribution. This is an overdispersed version of "hw" and can be derived from a special case of the Balding-Nichols model.

"s1" This prior assumes the individuals are all full-siblings resulting from one generation of selfing. I.e. there is only one parent. This model assumes a particular type of meiotic behavior: polysomic inheritance with bivalent, non-preferential pairing.

"f1" This prior assumes the individuals are all full-siblings resulting from one generation of a bi-parental cross. This model assumes a particular type of meiotic behavior: polysomic inheritance with bivalent, non-preferential pairing.

"f1pp" This prior allows for double reduction and preferential pairing in an F1 population of tetraploids.

"s1pp" This prior allows for double reduction and preferential pairing in an S1 population of tetraploids.

"flex" Generically any categorical distribution. Theoretically, this works well if you have a lot of individuals. In practice, it seems to be much less robust to violations in modeling assumptions.

"uniform" A discrete uniform distribution. This should never be used in practice.

"custom" A pre-specified prior distribution. You specify it using the prior_vec argument. You should almost never use this option in practice.

You might think a good default is `model = "uniform"` because it is somehow an "uninformative prior." But it is very informative and tends to work horribly in practice. The intuition is that it will estimate the allele bias and sequencing error rates so that the estimated genotypes are approximately uniform (since we are assuming that they are approximately uniform). This will usually result in unintuitive genotyping since most populations don't have a uniform genotype distribution. I include it as an option only for completeness. Please don't use it.

The value of `prop_mis` is a very intuitive measure for the quality of the SNP. `prop_mis` is the posterior proportion of individuals mis-genotyped. So if you want only SNPS that accurately genotype, say, 95% of the individuals, you could discard all SNPs with a `prop_mis` over `0.05`.

The value of `maxpostprob` is a very intuitive measure for the quality of the genotype estimate of an individual. This is the posterior probability of correctly genotyping the individual when using `geno` (the posterior mode) as the genotype estimate. So if you want to correctly genotype, say, 95% of individuals, you could discard all individuals with a `maxpostprob` of under `0.95`. However, if you are just going to impute missing genotypes later, you might consider not discarding any individuals as `flexdog`'s genotype estimates will probably be more accurate than other more naive approaches, such as imputing using the grand mean.

In most datasets I've examined, allelic bias is a major issue. However, you may fit the model assuming no allelic bias by setting `update_bias = FALSE` and `bias_init = 1`.

Prior to using `flexdog`, during the read-mapping step, you could try to get rid of allelic bias by using WASP (doi: 10.1101/011221). If you are successful in removing the allelic bias (because its only source was the read-mapping step), then setting `update_bias = FALSE` and `bias_init = 1` would be reasonable. You can visually inspect SNPs for bias by using `plot_geno()`.

`flexdog()`, like most methods, is invariant to which allele you label as the "reference" and which you label as the "alternative". That is, if you set `refvec` with the number of alternative read-counts, then the resulting genotype estimates will be the estimated allele dosage of the alternative allele.

Value

An object of class `flexdog`, which consists of a list with some or all of the following elements:

`bias` The estimated bias parameter.

`seq` The estimated sequencing error rate.

`od` The estimated overdispersion parameter.

`num_iter` The number of EM iterations ran. You should be wary if this equals `itermax`.

`llike` The maximum marginal log-likelihood.

`postmat` A matrix of posterior probabilities of each genotype for each individual. The rows index the individuals and the columns index the allele dosage.

`genologlike` A matrix of genotype *log*-likelihoods of each genotype for each individual. The rows index the individuals and the columns index the allele dosage.

`gene_dist` The estimated genotype distribution. The `i`th element is the proportion of individuals with genotype `i-1`.

`par` A list of the final estimates of the parameters of the genotype distribution. The elements included in `par` depends on the value of `model`:

`model = "norm"`: `mu`: The normal mean.

`sigma`: The normal standard deviation (not variance).

model = "hw": alpha: The major allele frequency.

model = "bb": alpha: The major allele frequency.

tau: The overdispersion parameter. See the description of rho in the Details of betabinom().

model = "s1": pgeno: The allele dosage of the parent.

alpha: The mixture proportion of the discrete uniform (included and fixed at a small value mostly for numerical stability reasons). See the description of fs1_alpha in flexdog_full().

model = "f1": p1geno: The allele dosage of the first parent.

p2geno: The allele dosage of the second parent.

alpha: The mixture proportion of the discrete uniform (included and fixed at a small value mostly for numerical stability reasons). See the description of fs1_alpha in flexdog_full().

model = "s1pp": ell1: The estimated dosage of the parent.

tau1: The estimated double reduction parameter of the parent. Available if ell1 is 1, 2, or 3. Identified if ell1 is 1 or 3.

gamma1: The estimated preferential pairing parameter. Available if ell1 is 2. However, it is not returned in an identified form.

alpha: The mixture proportion of the discrete uniform (included and fixed at a small value mostly for numerical stability reasons). See the description of fs1_alpha in flexdog_full().

model = "f1pp": ell1: The estimated dosage of parent 1.

ell2: The estimated dosage of parent 2.

tau1: The estimated double reduction parameter of parent 1. Available if ell1 is 1, 2, or 3. Identified if ell1 is 1 or 3.

tau2: The estimated double reduction parameter of parent 2. Available if ell2 is 1, 2, or 3. Identified if ell2 is 1 or 3.

gamma1: The estimated preferential pairing parameter of parent 1. Available if ell1 is 2. However, it is not returned in an identified form.

gamma2: The estimated preferential pairing parameter of parent 2. Available if ell2 is 2. However, it is not returned in an identified form.

alpha: The mixture proportion of the discrete uniform (included and fixed at a small value mostly for numerical stability reasons). See the description of fs1_alpha in flexdog_full().

model = "flex": par is an empty list.

model = "uniform": par is an empty list.

model = "custom": par is an empty list.

geno The posterior mode genotype. These are your genotype estimates.

maxpostprob The maximum posterior probability. This is equivalent to the posterior probability of correctly genotyping each individual.

postmean The posterior mean genotype. In downstream association studies, you might want to consider using these estimates.

input$refvec The value of refvec provided by the user.

input$sizevec The value of sizevec provided by the user.

input$ploidy The value of ploidy provided by the user.

`input$model` The value of `model` provided by the user.

`input$p1ref` The value of `p1ref` provided by the user.

`input$p1size` The value of `p1size` provided by the user.

`input$p2ref` The value of `p2ref` provided by the user.

`input$p2size` The value of `p2size` provided by the user.

`input$snpname` The value of `snpname` provided by the user.

`prop_mis` The posterior proportion of individuals genotyped incorrectly.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.
- Gerard, David, and Luís Felipe Ventorim Ferrão. "Priors for genotyping polyploids." Bioinformatics 36, no. 6 (2020): 1795-1800. doi: 10.1093/bioinformatics/btz852.

See Also

Run `browseVignettes(package = "updog")` in R for example usage. Other useful functions include:

`multidog` For running `flexdog` on multiple SNPs in parallel.

`flexdog` For a more user-friendly version of `flexdog_full`.

`rgeno` For simulating genotypes under the allowable prior models in `flexdog`.

`rflexdog` For simulating read-counts under the assumed likelihood model in `flexdog`.

`plot.flexdog` For plotting the output of `flexdog`.

`oracle_mis` For calculating the oracle genotyping error rates. This is useful for read-depth calculations *before* collecting data. After you have data, using the value of `prop_mis` is better.

`oracle_cor` For calculating the correlation between the true genotypes and an oracle estimator (useful for read-depth calculations *before* collecting data).

Examples

```
## A natural population. We will assume a
## normal prior since there are so few
## individuals.
data("uitdewilligen")
ploidy  <- 4
refvec  <- uitdewilligen$refmat[, 1]
sizevec <- uitdewilligen$sizemat[, 1]
fout    <- flexdog_full(refvec  = refvec,
                        sizevec = sizevec,
                        ploidy  = ploidy,
                        model   = "norm")
plot(fout)
```

format_multidog *Return arrayicized elements from the output of* multidog.

Description

This function will allow you to have genotype estimates, maximum posterior probability, and other values in the form of a matrix/array. If multiple variable names are provided, the data are formatted as a 3-dimensional array with the dimensions corresponding to (individuals, SNPs, variables).

Usage

```
format_multidog(x, varname = "geno")
```

Arguments

x	The output of multidog.
varname	A character vector of the variable names whose values populate the cells. These should be column names from x$inddf.

Details

Note that the order of the individuals will be reshuffled. The order of the SNPs should be the same as in x$snpdf.

Author(s)

David Gerard

get_q_array *Return the probabilities of an offspring's genotype given its parental genotypes for all possible combinations of parental and offspring genotypes. This is for species with polysomal inheritance and bivalent, non-preferential pairing.*

Description

Return the probabilities of an offspring's genotype given its parental genotypes for all possible combinations of parental and offspring genotypes. This is for species with polysomal inheritance and bivalent, non-preferential pairing.

Usage

```
get_q_array(ploidy)
```

Arguments

ploidy	A positive integer. The ploidy of the species.

Value

An three-way array of proportions. The (i, j, k)th element is the probability of an offspring having k - 1 reference alleles given that parent 1 has i - 1 reference alleles and parent 2 has j - 1 reference alleles. Each dimension of the array is ploidy + 1. In the dimension names, "A" stands for the reference allele and "a" stands for the alternative allele.

Author(s)

David Gerard

Examples

```
qarray <- get_q_array(6)
apply(qarray, c(1, 2), sum) ## should all be 1's.
```

---

is.flexdog *Tests if an argument is a* flexdog *object.*

Description

Tests if an argument is a flexdog object.

Usage

```
is.flexdog(x)
```

Arguments

x	Anything.

Value

A logical. TRUE if x is a flexdog object, and FALSE otherwise.

Author(s)

David Gerard

Examples

```
is.flexdog("anything")
# FALSE
```

is.multidog *Tests if an argument is a* multidog *object.*

Description

Tests if an argument is a multidog object.

Usage

```
is.multidog(x)
```

Arguments

x	Anything.

Value

A logical. TRUE if x is a multidog object, and FALSE otherwise.

Author(s)

David Gerard

Examples

```
is.multidog("anything")
# FALSE
```

log_sum_exp *Log-sum-exponential trick.*

Description

Log-sum-exponential trick.

Usage

```
log_sum_exp(x)
```

Arguments

x	A vector to log-sum-exp.

Value

The log of the sum of the exponential of the elements in x.

Author(s)

David Gerard

`log_sum_exp_2` *Log-sum-exponential trick using just two doubles.*

Description

Log-sum-exponential trick using just two doubles.

Usage

```
log_sum_exp_2(x, y)
```

Arguments

x	A double.
y	Another double.

Value

The log of the sum of the exponential of x and y.

Author(s)

David Gerard

`multidog` *Fit* `flexdog` *to multiple SNPs.*

Description

This is a convenience function that will run `flexdog` over many SNPs. Support is provided for parallel computing through the doParallel package. This function has not been extensively tested. Please report any bugs to https://github.com/dcgerard/updog/issues.

Usage

```
multidog(
  refmat,
  sizemat,
  ploidy,
  model = c("norm", "hw", "bb", "s1", "s1pp", "f1", "f1pp", "flex", "uniform",
    "custom"),
  nc = 1,
  p1_id = NULL,
  p2_id = NULL,
  bias_init = exp(c(-1, -0.5, 0, 0.5, 1)),
  prior_vec = NULL,
  ...
)
```

Arguments

refmat	A matrix of reference read counts. The columns index the individuals and the rows index the markers (SNPs). This matrix must have rownames (for the names of the markers) and column names (for the names of the individuals). These names must match the names in sizemat.
sizemat	A matrix of total read counts. The columns index the individuals and the rows index the markers (SNPs). This matrix must have rownames (for the names of the markers) and column names (for the names of the individuals). These names must match the names in refmat.
ploidy	The ploidy of the species. Assumed to be the same for each individual.
model	What form should the prior (genotype distribution) take? See Details for possible values.
nc	The number of computing cores to use when doing parallelization on your local machine. See the section "Parallel Computation" for how to implement more complicated evaluation strategies using the future package. When you are specifying other evaluation strategies using the future package, you should also set nc = NA. The value of nc should never be more than the number of cores available in your computing environment. You can determine the maximum number of available cores by running future::availableCores() in R.
p1_id	The ID of the first parent. This should be a character of length 1. This should correspond to a single column name in refmat and sizemat.
p2_id	The ID of the second parent. This should be a character of length 1. This should correspond to a single column name in refmat and sizemat.
bias_init	A vector of initial values for the bias parameter over the multiple runs of flexdog_full().
prior_vec	The pre-specified genotype distribution. Only used if model = "custom" and must otherwise be NULL. If specified, then it should be a vector of length ploidy + 1 with non-negative elements that sum to 1.
...	Additional parameters to pass to flexdog_full().

Details

You should format your reference counts and total read counts in two separate matrices. The rows should index the markers (SNPs) and the columns should index the individuals. Row names are how we ID the SNPs and column names are how we ID the individuals, and so they are required attributes.

If your data are in VCF files, I would recommend importing them using the VariantAnnotation package from Bioconductor https://bioconductor.org/packages/VariantAnnotation/. It's a great VCF parser.

See the details of flexdog for the possible values of model.

If model = "f1", model = "s1", model = "f1pp" or model = "s1pp" then the user may provide the individual ID for parent(s) via the p1_id and p2_id arguments.

The output is a list containing two data frames. The first data frame, called snpdf, contains information on each SNP, such as the allele bias and the sequencing error rate. The second data frame, called inddf, contains information on each individual at each SNP, such as the estimated genotype and the posterior probability of being classified correctly.

SNPs that contain 0 reads (or all missing data) are entirely removed.

Value

A list-like object of two data frames.

snpdf A data frame containing properties of the SNPs (markers). The rows index the SNPs. The variables include:

- snp The name of the SNP (marker).
- bias The estimated allele bias of the SNP.
- seq The estimated sequencing error rate of the SNP.
- od The estimated overdispersion parameter of the SNP.
- prop_mis The estimated proportion of individuals misclassified in the SNP.
- num_iter The number of iterations performed during the EM algorithm for that SNP.
- llike The maximum marginal likelihood of the SNP.
- ploidy The provided ploidy of the species.
- model The provided model for the prior genotype distribution.
- p1ref The user-provided reference read counts of parent 1.
- p1size The user-provided total read counts of parent 1.
- p2ref The user-provided reference read counts of parent 2.
- p2size The user-provided total read counts of parent 2.
- Pr_k The estimated frequency of individuals with genotype k, where k can be any integer between 0 and the ploidy level.

Model specific parameter estimates See the return value of par in the help page of flexdog.

inddf A data frame containing the properties of the individuals at each SNP. The variables include:

- snp The name of the SNP (marker).
- ind The name of the individual.
- ref The provided reference counts for that individual at that SNP.

`size` The provided total counts for that individual at that SNP.

`geno` The posterior mode genotype for that individual at that SNP. This is the estimated reference allele dosage for a given individual at a given SNP.

`postmean` The posterior mean genotype for that individual at that SNP. This is a continuous genotype estimate of the reference allele dosage for a given individual at a given SNP.

`maxpostprob` The maximum posterior probability. This is the posterior probability that the individual was genotyped correctly.

`Pr_k` The posterior probability that a given individual at a given SNP has genotype k, where k can vary from 0 to the ploidy level of the species.

`logL_k` The genotype *log*-likelihoods for dosage k for a given individual at a given SNP, where k can vary f rom 0 to the ploidy level of the species.

Parallel Computation

The `multidog()` function supports parallel computing. It does so through the future package.

If you are just running `multidog()` on a local machine, then you can use the `nc` argument to specify the parallelization. Any value of `nc` greater than 1 will result in multiple background R sessions to genotype all of the SNPs. The maximum value of `nc` you should try can be found by running `future::availableCores()`. Running `multidog()` using `nc` is equivalent to setting the future plan with `future::plan(future::multisession,workers = nc)`.

Using the future package means that different evaluation strategies are possible. In particular, if you are using a high performance machine, you can explore using the future.batchtools package to evaluate `multidog()` using schedulers like Slurm or TORQUE/PBS.

To use a different strategy, set `nc = NA` and then run `future::plan()` prior to running `multidog()`. For example, to set up forked R processes on your current machine (instead of using background R sessions), you would run (will not work on Windows): `future::plan(future::multicore)`, followed by running `multidog()` with `nc = NA`. See the examples below.

Author(s)

David Gerard

See Also

- `flexdog()`:For the underlying genotyping function.
- `format_multidog()`:For converting the output of `multidog()` to a matrix.
- `filter_snp()`:For filtering SNPs using the output of `multidog()`.

Examples

```
## Not run:
data("uitdewilligen")

## Run multiple R sessions using the `nc` variable.
mout <- multidog(refmat = t(uitdewilligen$refmat),
                 sizemat = t(uitdewilligen$sizemat),
                 ploidy = uitdewilligen$ploidy,
                 nc = 2)
```

```
mout$inddf
mout$snpdf

## Run multiple external R sessions on the local machine.
## Note that we set `nc = NA`.
cl <- parallel::makeCluster(2, timeout = 60)
future::plan(future::cluster, workers = cl)
mout <- multidog(refmat = t(uitdewilligen$refmat),
                 sizemat = t(uitdewilligen$sizemat),
                 ploidy = uitdewilligen$ploidy,
                 nc = NA)
mout$inddf
mout$snpdf

## Close cluster and reset future to current R process
parallel::stopCluster(cl)
future::plan(future::sequential)

## End(Not run)
```

`oracle_cor` *Calculates the correlation between the true genotype and an oracle estimator.*

Description

Calculates the correlation between the oracle MAP estimator (where we have perfect knowledge about the data generation process) and the true genotype. This is a useful approximation when you have a lot of individuals.

Usage

```
oracle_cor(n, ploidy, seq, bias, od, dist)
```

Arguments

`n`	The read-depth.
`ploidy`	The ploidy of the individual.
`seq`	The sequencing error rate.
`bias`	The allele-bias.
`od`	The overdispersion parameter.
`dist`	The distribution of the alleles.

Details

To come up with `dist`, you need some additional assumptions. For example, if the population is in Hardy-Weinberg equilibrium and the allele frequency is `alpha` then you could calculate `dist` using the R code: `dbinom(x = 0:ploidy, size = ploidy, prob = alpha)`. Alternatively, if you know the genotypes of the individual's two parents are, say, `ref_count1` and `ref_count2`, then you could use the `get_q_array` function from the updog package: `get_q_array(ploidy)[ref_count1 + 1, ref_count2 + 1, ]`.

Value

The Pearson correlation between the true genotype and the oracle estimator.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.

Examples

```
## Hardy-Weinberg population with allele-frequency of 0.75.
## Moderate bias and moderate overdispersion.
## See how correlation decreases as we
## increase the ploidy.
ploidy <- 2
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
oracle_cor(n = 100, ploidy = ploidy, seq = 0.001,
           bias = 0.7, od = 0.01, dist = dist)

ploidy <- 4
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
oracle_cor(n = 100, ploidy = ploidy, seq = 0.001,
           bias = 0.7, od = 0.01, dist = dist)

ploidy <- 6
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
oracle_cor(n = 100, ploidy = ploidy, seq = 0.001,
           bias = 0.7, od = 0.01, dist = dist)
```

`oracle_cor_from_joint` *Calculate the correlation of the oracle estimator with the true genotype from the joint distribution matrix.*

Description

Calculates the correlation between the oracle MAP estimator (where we have perfect knowledge about the data generation process) and the true genotype. This is a useful approximation when you have a lot of individuals.

Usage

```
oracle_cor_from_joint(jd)
```

Arguments

jd	A matrix of numerics. Element (i, j) is the probability of genotype i - 1 and estimated genotype j - 1. This is usually obtained from oracle_joint.

Value

The Pearson correlation between the true genotype and the oracle estimator.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.

See Also

oracle_joint for getting jd. oracle_cor for not having to first calculate jd.

Examples

```
## Hardy-Weinberg population with allele-frequency of 0.75.
## Moderate bias and moderate overdispersion.
ploidy <- 6
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
jd <- oracle_joint(n = 100, ploidy = ploidy, seq = 0.001,
                   bias = 0.7, od = 0.01, dist = dist)
oracle_cor_from_joint(jd = jd)

## Compare to oracle_cor
oracle_cor(n = 100, ploidy = ploidy, seq = 0.001,
           bias = 0.7, od = 0.01, dist = dist)
```

`oracle_joint` *The joint probability of the genotype and the genotype estimate of an oracle estimator.*

Description

This returns the joint distribution of the true genotypes and an oracle estimator given perfect knowledge of the data generating process. This is a useful approximation when you have a lot of individuals.

Usage

```
oracle_joint(n, ploidy, seq, bias, od, dist)
```

Arguments

`n`	The read-depth.
`ploidy`	The ploidy of the individual.
`seq`	The sequencing error rate.
`bias`	The allele-bias.
`od`	The overdispersion parameter.
`dist`	The distribution of the alleles.

Details

To come up with `dist`, you need some additional assumptions. For example, if the population is in Hardy-Weinberg equilibrium and the allele frequency is `alpha` then you could calculate `dist` using the R code: `dbinom(x = 0:ploidy, size = ploidy, prob = alpha)`. Alternatively, if you know the genotypes of the individual's two parents are, say, `ref_count1` and `ref_count2`, then you could use the `get_q_array` function from the updog package: `get_q_array(ploidy)[ref_count1 + 1, ref_count2 + 1, ]`.

See the Examples to see how to reconcile the output of `oracle_joint` with `oracle_mis` and `oracle_mis_vec`.

Value

A matrix. Element (i, j) is the joint probability of estimating the genotype to be i+1 when the true genotype is j+1. That is, the estimated genotype indexes the rows and the true genotype indexes the columns. This is when using an oracle estimator.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.

See Also

`oracle_plot` For visualizing the joint distribution output from `oracle_joint`.

`oracle_mis_from_joint` For obtaining the same results as `oracle_mis` directly from the output of `oracle_joint`.

`oracle_mis_vec_from_joint` For obtaining the same results as `oracle_mis_vec` directly from the output of `oracle_joint`.

`oracle_cor_from_joint` For obtaining the same results as `oracle_cor` directly from the output of `oracle_joint`.

Examples

```
## Hardy-Weinberg population with allele-frequency of 0.75.
## Moderate bias and moderate overdispersion.
ploidy <- 4
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
jd <- oracle_joint(n = 100, ploidy = ploidy, seq = 0.001,
                   bias = 0.7, od = 0.01, dist = dist)
jd

## Get same output as oracle_mis this way:
1 - sum(diag(jd))
oracle_mis(n = 100, ploidy = ploidy, seq = 0.001,
           bias = 0.7, od = 0.01, dist = dist)

## Get same output as oracle_mis_vec this way:
1 - diag(sweep(x = jd, MARGIN = 2, STATS = colSums(jd), FUN = "/"))
oracle_mis_vec(n = 100, ploidy = ploidy, seq = 0.001,
               bias = 0.7, od = 0.01, dist = dist)
```

`oracle_mis` *Calculate oracle misclassification error rate.*

Description

Given perfect knowledge of the data generating parameters, `oracle_mis` calculates the misclassification error rate, where the error rate is taken over both the data generation process and the allele-distribution. This is an ideal level of the misclassification error rate and any real method will have a larger rate than this. This is a useful approximation when you have a lot of individuals.

Usage

```
oracle_mis(n, ploidy, seq, bias, od, dist)
```

Arguments

n	The read-depth.
ploidy	The ploidy of the individual.
seq	The sequencing error rate.
bias	The allele-bias.
od	The overdispersion parameter.
dist	The distribution of the alleles.

Details

To come up with dist, you need some additional assumptions. For example, if the population is in Hardy-Weinberg equilibrium and the allele frequency is alpha then you could calculate dist using the R code: dbinom(x = 0:ploidy,size = ploidy,prob = alpha). Alternatively, if you know the genotypes of the individual's two parents are, say, ref_count1 and ref_count2, then you could use the get_q_array function from the updog package: get_q_array(ploidy)[ref_count1 + 1,ref_count2 + 1,].

Value

A double. The oracle misclassification error rate.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.

Examples

```
## Hardy-Weinberg population with allele-frequency of 0.75.
## Moderate bias and moderate overdispersion.
## See how oracle misclassification error rates change as we
## increase the ploidy.
ploidy <- 2
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
oracle_mis(n = 100, ploidy = ploidy, seq = 0.001,
           bias = 0.7, od = 0.01, dist = dist)

ploidy <- 4
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
oracle_mis(n = 100, ploidy = ploidy, seq = 0.001,
           bias = 0.7, od = 0.01, dist = dist)

ploidy <- 6
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
oracle_mis(n = 100, ploidy = ploidy, seq = 0.001,
```

```
            bias = 0.7, od = 0.01, dist = dist)
```

`oracle_mis_from_joint` *Get the oracle misclassification error rate directly from the joint distribution of the genotype and the oracle estimator.*

Description

Get the oracle misclassification error rate directly from the joint distribution of the genotype and the oracle estimator.

Usage

```
oracle_mis_from_joint(jd)
```

Arguments

`jd`	A matrix of numerics. Element (i, j) is the probability of genotype i - 1 and estimated genotype j - 1. This is usually obtained from `oracle_joint`.

Value

A double. The oracle misclassification error rate.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.

See Also

`oracle_joint` for getting `jd`. `oracle_mis` for not having to first calculate `jd`.

Examples

```
## Hardy-Weinberg population with allele-frequency of 0.75.
## Moderate bias and moderate overdispersion.
ploidy <- 6
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
jd <- oracle_joint(n = 100, ploidy = ploidy, seq = 0.001,
                   bias = 0.7, od = 0.01, dist = dist)
oracle_mis_from_joint(jd = jd)

## Compare to oracle_cor
```

```
oracle_mis(n = 100, ploidy = ploidy, seq = 0.001,
           bias = 0.7, od = 0.01, dist = dist)
```

`oracle_mis_vec` *Returns the oracle misclassification rates for each genotype.*

Description

Given perfect knowledge of the data generating parameters, `oracle_mis_vec` calculates the misclassification error rate at each genotype. This differs from `oracle_mis` in that this will *not* average over the genotype distribution to get an overall misclassification error rate. That is, `oracle_mis_vec` returns a vector of misclassification error rates *conditional* on each genotype.

Usage

```
oracle_mis_vec(n, ploidy, seq, bias, od, dist)
```

Arguments

`n`	The read-depth.
`ploidy`	The ploidy of the individual.
`seq`	The sequencing error rate.
`bias`	The allele-bias.
`od`	The overdispersion parameter.
`dist`	The distribution of the alleles.

Details

This is an ideal level of the misclassification error rate and any real method will have a larger rate than this. This is a useful approximation when you have a lot of individuals.

To come up with `dist`, you need some additional assumptions. For example, if the population is in Hardy-Weinberg equilibrium and the allele frequency is `alpha` then you could calculate `dist` using the R code: `dbinom(x = 0:ploidy,size = ploidy,prob = alpha)`. Alternatively, if you know the genotypes of the individual's two parents are, say, `ref_count1` and `ref_count2`, then you could use the `get_q_array` function from the updog package: `get_q_array(ploidy)[ref_count1 + 1,ref_count2 + 1,]`.

Value

A vector of numerics. Element i is the oracle misclassification error rate when genotyping an individual with actual genotype i + 1.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.

Examples

```
## Hardy-Weinberg population with allele-frequency of 0.75.
## Moderate bias and moderate overdispersion.
ploidy <- 4
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
om <- oracle_mis_vec(n = 100, ploidy = ploidy, seq = 0.001,
                     bias = 0.7, od = 0.01, dist = dist)
om

## Get same output as oracle_mis this way:
sum(dist * om)
oracle_mis(n = 100, ploidy = ploidy, seq = 0.001,
           bias = 0.7, od = 0.01, dist = dist)
```

`oracle_mis_vec_from_joint`

Get the oracle misclassification error rates (conditional on true genotype) directly from the joint distribution of the genotype and the oracle estimator.

Description

Get the oracle misclassification error rates (conditional on true genotype) directly from the joint distribution of the genotype and the oracle estimator.

Usage

```
oracle_mis_vec_from_joint(jd)
```

Arguments

`jd`	A matrix of numerics. Element (i, j) is the probability of genotype i - 1 and estimated genotype j - 1. This is usually obtained from `oracle_joint`.

Value

A vector of numerics. Element i is the oracle misclassification error rate when genotyping an individual with actual genotype i + 1.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.

See Also

`oracle_joint` for getting `jd`. `oracle_mis_vec` for not having to first calculate `jd`.

Examples

```
## Hardy-Weinberg population with allele-frequency of 0.75.
## Moderate bias and moderate overdispersion.
ploidy <- 6
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
jd <- oracle_joint(n = 100, ploidy = ploidy, seq = 0.001,
                   bias = 0.7, od = 0.01, dist = dist)
oracle_mis_vec_from_joint(jd = jd)

## Compare to oracle_cor
oracle_mis_vec(n = 100, ploidy = ploidy, seq = 0.001,
               bias = 0.7, od = 0.01, dist = dist)
```

---

oracle_plot *Construct an oracle plot from the output of* `oracle_joint`.

Description

After obtaining the joint distribution of the true genotype with the estimated genotype from the oracle estimator using `oracle_joint`, you can use `oracle_plot` to visualize this joint distribution.

Usage

```
oracle_plot(jd)
```

Arguments

`jd`	A matrix containing the joint distribution of the true genotype and the oracle estimator. Usually, this is obtained by a call from `oracle_joint`.

Value

A `ggplot` object containing the oracle plot. The x-axis indexes the possible values of the estimated genotype. The y-axis indexes the possible values of the true genotype. The number in cell (i, j) is the probability that an individual will have true genotype i but is estimated to have genotype j. This is when using an oracle estimator. The cells are also color-coded by the size of the probability in each cell. At the top are listed the oracle misclassification error rate and the correlation of the true genotype with the estimated genotype. Both of these quantities may be derived from the joint distribution.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.

See Also

`oracle_joint` for obtaining `jd`.

Examples

```
ploidy <- 6
dist <- stats::dbinom(0:ploidy, ploidy, 0.75)
jd <- oracle_joint(n = 100, ploidy = ploidy, seq = 0.001,
                   bias = 0.7, od = 0.01, dist = dist)
pl <- oracle_plot(jd = jd)
print(pl)
```

---

plot.flexdog *Draw a genotype plot from the output of* `flexdog`.

Description

A wrapper for `plot_geno`. This will create a genotype plot for a single SNP.

Usage

```
## S3 method for class 'flexdog'
plot(x, use_colorblind = TRUE, ...)
```

Arguments

`x`	A `flexdog` object.
`use_colorblind`	Should we use a colorblind-safe palette (`TRUE`) or not (`FALSE`)? `TRUE` is only allowed if the ploidy is less than or equal to 6.
`...`	Not used.

Details

On a genotype plot, the x-axis contains the counts of the non-reference allele and the y-axis contains the counts of the reference allele. The dashed lines are the expected counts (both reference and alternative) given the sequencing error rate and the allele-bias. The plots are color-coded by the maximum-a-posterior genotypes. Transparency is proportional to the maximum posterior probability for an individual's genotype. Thus, we are less certain of the genotype of more transparent individuals. These types of plots are used in Gerard et. al. (2018) and Gerard and Ferrão (2020).

Value

A ggplot object for the genotype plot.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.
- Gerard, David, and Luís Felipe Ventorim Ferrão. "Priors for genotyping polyploids." Bioinformatics 36, no. 6 (2020): 1795-1800. doi: 10.1093/bioinformatics/btz852.

See Also

plot_geno The underlying plotting function.

flexdog Creates a flexdog object.

---

plot.multidog *Plot the output of multidog.*

Description

Produce genotype plots from the output of multidog. You may select which SNPs to plot.

Usage

```
## S3 method for class 'multidog'
plot(x, indices = seq(1, min(5, nrow(x$snpdf))), ...)
```

Arguments

x	The output of multidog.
indices	A vector of integers. The indices of the SNPs to plot.
...	not used.

Details

On a genotype plot, the x-axis contains the counts of the non-reference allele and the y-axis contains the counts of the reference allele. The dashed lines are the expected counts (both reference and alternative) given the sequencing error rate and the allele-bias. The plots are color-coded by the maximum-a-posterior genotypes. Transparency is proportional to the maximum posterior probability for an individual's genotype. Thus, we are less certain of the genotype of more transparent individuals. These types of plots are used in Gerard et. al. (2018) and Gerard and Ferrão (2020).

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.
- Gerard, David, and Luís Felipe Ventorim Ferrão. "Priors for genotyping polyploids." Bioinformatics 36, no. 6 (2020): 1795-1800. doi: 10.1093/bioinformatics/btz852.

See Also

plot_geno.

---

plot_geno *Make a genotype plot.*

Description

The x-axis is the counts of the non-reference allele, and the y-axis is the counts of the reference allele. Transparency is controlled by the maxpostprob vector. These types of plots are used in Gerard et. al. (2018) and Gerard and Ferrão (2020).

Usage

```
plot_geno(
  refvec,
  sizevec,
  ploidy,
  p1ref = NULL,
  p1size = NULL,
  p2ref = NULL,
  p2size = NULL,
  geno = NULL,
  seq = 0,
  bias = 1,
  maxpostprob = NULL,
  p1geno = NULL,
  p2geno = NULL,
  use_colorblind = TRUE
)
```

Arguments

refvec	A vector of non-negative integers. The number of reference reads observed in the individuals
sizevec	A vector of positive integers. The total number of reads in the individuals.
ploidy	A non-negative integer. The ploidy of the species.
p1ref	A vector of non-negative integers. The number of reference reads observed in parent 1 (if the individuals are all siblings).
p1size	A vector of positive integers. The total number of reads in parent 1 (if the individuals are all siblings).
p2ref	A vector of non-negative integers. The number of reference reads observed in parent 2 (if the individuals are all siblings).
p2size	A vector of positive integers. The total number of reads in parent 2 (if the individuals are all siblings).
geno	The individual genotypes.
seq	The sequencing error rate.
bias	The bias parameter.
maxpostprob	A vector of the posterior probabilities of being at the modal genotype.
p1geno	Parent 1's genotype.
p2geno	Parent 2's genotype.
use_colorblind	A logical. Should we use a colorblind safe palette (TRUE), or not (FALSE)? Only allowed if ploidy <= 6.

Details

If parental genotypes are provided (p1geno and p2geno) then they will be colored the same as the offspring. Since they are often hard to see, a small black dot will also indicate their position.

Value

A ggplot object for the genotype plot.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.
- Gerard, David, and Luís Felipe Ventorim Ferrão. "Priors for genotyping polyploids." Bioinformatics 36, no. 6 (2020): 1795-1800. doi: 10.1093/bioinformatics/btz852.

Examples

```
data("snpdat")
refvec  <- snpdat$counts[snpdat$snp == "SNP1"]
sizevec <- snpdat$size[snpdat$snp == "SNP1"]
ploidy  <- 6
plot_geno(refvec = refvec, sizevec = sizevec, ploidy = ploidy)
```

`rflexdog` *Simulate GBS data from the* `flexdog` *likelihood.*

Description

This will take a vector of genotypes and a vector of total read-counts, then generate a vector of reference counts. To get the genotypes, you could use `rgeno`. The likelihood used to generate read-counts is described in detail in Gerard et. al. (2018).

Usage

```
rflexdog(sizevec, geno, ploidy, seq = 0.005, bias = 1, od = 0.001)
```

Arguments

`sizevec`	A vector of total read-counts for the individuals.
`geno`	A vector of genotypes for the individuals. I.e. the number of reference alleles each individual has.
`ploidy`	The ploidy of the species.
`seq`	The sequencing error rate.
`bias`	The bias parameter. Pr(a read after selected) / Pr(A read after selected).
`od`	The overdispersion parameter. See the Details of the `rho` variable in `betabinom`.

Value

A vector the same length as `sizevec`. The ith element is the number of reference counts for individual i.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.
- Gerard, David, and Luís Felipe Ventorim Ferrão. "Priors for genotyping polyploids." Bioinformatics 36, no. 6 (2020): 1795-1800. doi: 10.1093/bioinformatics/btz852.

See Also

rgeno for a way to generate genotypes of individuals. rbetabinom for how we generate the read-counts.

Examples

```
set.seed(1)
n       <- 100
ploidy  <- 6

## Generate the genotypes of individuals from an F1 population,
## where the first parent has 1 copy of the reference allele
## and the second parent has two copies of the reference
## allele.
genovec <- rgeno(n = n, ploidy = ploidy, model = "f1",
                 p1geno = 1, p2geno = 2)

## Get the total number of read-counts for each individual.
## Ideally, you would take this from real data as the total
## read-counts are definitely not Poisson.
sizevec <- stats::rpois(n = n, lambda = 200)

## Generate the counts of reads with the reference allele
## when there is a strong bias for the reference allele
## and there is no overdispersion.
refvec  <- rflexdog(sizevec = sizevec, geno = genovec,
                    ploidy = ploidy, seq = 0.001,
                    bias = 0.5, od = 0)

## Plot the simulated data using plot_geno.
plot_geno(refvec = refvec, sizevec = sizevec,
          ploidy = ploidy, seq = 0.001, bias = 0.5)
```

---

rgeno *Simulate individual genotypes from one of the supported* flexdog *models.*

Description

This will simulate genotypes of a sample of individuals drawn from one of the populations supported by flexdog. See the details of flexdog for the models allowed. These genotype distributions are described in detail in Gerard and Ferrão (2020).

Usage

```
rgeno(
  n,
  ploidy,
```

```
  model = c("hw", "bb", "norm", "f1", "s1", "flex", "uniform"),
  allele_freq = NULL,
  od = NULL,
  p1geno = NULL,
  p2geno = NULL,
  pivec = NULL,
  mu = NULL,
  sigma = NULL
)
```

Arguments

n	The number of observations.
ploidy	The ploidy of the species.
model	What form should the prior take? See Details in flexdog.
allele_freq	If model = "hw", then this is the allele frequency of the population. For any other model, this should be NULL.
od	If model = "bb", then this is the overdispersion parameter of the beta-binomial distribution. See betabinom for details. For any other model, this should be NULL.
p1geno	Either the first parent's genotype if model = "f1", or the only parent's genotype if model = "s1". For any other model, this should be NULL.
p2geno	The second parent's genotype if model = "f1". For any other model, this should be NULL.
pivec	A vector of probabilities. If model = "ash", then this represents the mixing proportions of the discrete uniforms. If model = "flex", then element i is the probability of genotype i -1. For any other model, this should be NULL.
mu	If model = "norm", this is the mean of the normal. For any other model, this should be NULL.
sigma	If model = "norm", this is the standard deviation of the normal. For any other model, this should be NULL.

Details

List of non-NULL arguments:

model = "flex": pivec

model = "hw": allele_freq

model = "f1": p1geno and p2geno

model = "s1": p1geno

model = "uniform": no non-NULL arguments

model = "bb": allele_freq and od

model == "norm": mu and sigma

Value

A vector of length n with the genotypes of the sampled individuals.

Author(s)

David Gerard

References

- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.
- Gerard, David, and Luís Felipe Ventorim Ferrão. "Priors for genotyping polyploids." Bioinformatics 36, no. 6 (2020): 1795-1800. doi: 10.1093/bioinformatics/btz852.

Examples

```
## F1 Population where parent 1 has 1 copy of the referenc allele
## and parent 2 has 4 copies of the reference allele.
ploidy <- 6
rgeno(n = 10, ploidy = ploidy, model = "f1", p1geno = 1, p2geno = 4)

## A population in Hardy-Weinberge equilibrium with an
## allele frequency of 0.75
rgeno(n = 10, ploidy = ploidy, model = "hw", allele_freq = 0.75)
```

---

snpdat *GBS data from Shirasawa et al (2017)*

Description

Contains counts of reference alleles and total read counts from the GBS data of Shirasawa et al (2017) for the three SNPs used as examples in Gerard et. al. (2018).

Usage

```
snpdat
```

Format

A tibble with 419 rows and 4 columns:

id The identification label of the individuals.

snp The SNP label.

counts The number of read-counts that support the reference allele.

size The total number of read-counts at a given SNP.

Value

A `tibble`. See the Format Section.

Source

doi: 10.1038/srep44207

References

- Shirasawa, Kenta, Masaru Tanaka, Yasuhiro Takahata, Daifu Ma, Qinghe Cao, Qingchang Liu, Hong Zhai, Sang-Soo Kwak, Jae Cheol Jeong, Ung-Han Yoon, Hyeong-Un Lee, Hideki Hirakawa, and Sahiko Isobe "A high-density SNP genetic map consisting of a complete set of homologous groups in autohexaploid sweetpotato (Ipomoea batatas)." *Scientific Reports 7* (2017). doi: 10.1038/srep44207
- Gerard, D., Ferrão, L. F. V., Garcia, A. A. F., & Stephens, M. (2018). Genotyping Polyploids from Messy Sequencing Data. *Genetics*, 210(3), 789-807. doi: 10.1534/genetics.118.301468.

`uitdewilligen` *Subset of individuals and SNPs from Uitdewilligen et al (2013).*

Description

A list containing a matrix of reference counts, a matrix of total counts, and the ploidy level (4) of the species. This is a subset of the data from Uitdewilligen et al (2013).

Usage

```
uitdewilligen
```

Format

A list containing three objects. Two matrices and a numeric scalar:

refmat A matrix of read counts containing the reference allele. The rows index the individuals and the columns index the SNPs.

sizemat A matrix of the total number of read counts. The rows index the individuals and the columns index the SNPs.

ploidy The ploidy level of the species (just 4).

Value

A list. See the Format Section.

Source

doi: 10.1371/journal.pone.0062355

References

- Uitdewilligen, J. G., Wolters, A. M. A., Bjorn, B., Borm, T. J., Visser, R. G., & van Eck, H. J. (2013). A next-generation sequencing method for genotyping-by-sequencing of highly heterozygous autotetraploid potato. *PLoS One*, 8(5), e62355. doi: 10.1371/journal.pone.0062355

`wem` *EM algorithm to fit weighted ash objective.*

Description

Solves the following optimization problem

$$\max_{\pi} \sum_{k} w_k \log(\sum_{j} \pi_j \ell_j k).$$

It does this using a weighted EM algorithm.

Usage

```
wem(weight_vec, lmat, pi_init, lambda, itermax, obj_tol)
```

Arguments

`weight_vec`	A vector of weights. Each element of `weight_vec` corresponds to a column of `lmat`.
`lmat`	A matrix of inner weights. The columns are the "individuals" and the rows are the "classes."
`pi_init`	The initial values of `pivec`. Each element of `pi_init` corresponds to a row of `lmat`.
`lambda`	The penalty on the pi's. Should be greater than 0 and really really small.
`itermax`	The maximum number of EM iterations to take.
`obj_tol`	The objective stopping criterion.

Value

A vector of numerics.

Author(s)

David Gerard

Examples

```
set.seed(2)
n <- 3
p <- 5
lmat <- matrix(stats::runif(n * p), nrow = n)
weight_vec <- seq_len(p)
pi_init <- stats::runif(n)
pi_init <- pi_init / sum(pi_init)
wem(weight_vec = weight_vec,
    lmat       = lmat,
    pi_init    = pi_init,
    lambda     = 0,
    itermax    = 100,
    obj_tol    = 10^-6)
```

Index

∗ **datasets**
    snpdat, 43
    uitdewilligen, 44

betabinom, *10*, *17*, *40*, *42*
betabinom *(*dbetabinom*)*, 4

dbetabinom, 4

export_vcf, 6

filter_snp, *3*, 7, *25*
flexdog, *3, 4*, 8, *13*, *18*, *22*, *24, 25*, *36, 37*, *40–42*
flexdog_full, *9*, *11–13*, 13, *17*, *23*
format_multidog, *3*, 19, *25*

get_q_array, 19, *27*, *29*, *31*, *33*
ggplot, *35*, *37*, *39*

is.flexdog, 20
is.multidog, 21

log_sum_exp, 21
log_sum_exp_2, 22

multidog, *3*, *6–8*, *12*, *18, 19*, 22, *37*

oracle_cor, *3*, *12*, *18*, 26, *28*, *30*
oracle_cor_from_joint, 27, *30*
oracle_joint, *3*, *28*, 29, *32*, *34–36*
oracle_mis, *3*, *12*, *18*, *29, 30*, 30, *32, 33*
oracle_mis_from_joint, *30*, 32
oracle_mis_vec, *29, 30*, 33, *35*
oracle_mis_vec_from_joint, *30*, 34
oracle_plot, *3*, *30*, 35

pbetabinom *(*dbetabinom*)*, 4
plot.flexdog, *3*, *12*, *18*, 36
plot.multidog, *3*, 37
plot_geno, *10*, *16*, *36–38*, 38

qbetabinom *(*dbetabinom*)*, 4

rbetabinom, *41*
rbetabinom *(*dbetabinom*)*, 4
rflexdog, *3*, *12*, *18*, 40
rgeno, *3*, *12*, *18*, *40, 41*, 41

snpdat, *4*, 43

uitdewilligen, *4*, 44
updog *(*updog-package*)*, 3
updog-package, 3

wem, 45