Package 'stockR'

April 26, 2023

Title Identifying Stocks in Genetic Data

Version 1.0.76

Author Scott D. Foster [aut, cre]

Description Provides a mixture model for clustering individuals (or sampling groups) into stocks based on their genetic profile. Here, sampling groups are individuals that are sure to come from the same stock (e.g. breeding adults or larvae). The mixture (log-)likelihood is maximised using the EM-algorithm after finding good starting values via a K-means clustering of the genetic data. Details can be found in: Foster, S. D.; Feutry, P.; Grewe, P. M.; Berry, O.; Hui, F. K. C. & Davies (2020) <doi:10.1111/1755-0998.12920>.

License GPL (>= 2)

LinkingTo Rcpp

Imports Rcpp, stats, gtools, parallel, RColorBrewer, methods

Suggests knitr

VignetteBuilder knitr

NeedsCompilation yes

Maintainer Scott D. Foster <scott.foster@data61.csiro.au>

Depends R (>= 2.10)

Repository CRAN

Date/Publication 2023-04-26 08:00:02 UTC

R topics documented:

`calcFst` *Calculate Fst statistics for frequency data*

Description

Calculates Weir and Cockerham's Fst for frequency data

Usage

```
calcFst(data, grps)
```

Arguments

	You can put normal text in **Arguments**, too, like this. Remember to indent all arguments, as below.
	a nMarker by nAnimal matrix of allele frequencies. That is for animal i and codominant marker j data[j,i] is the number of copies of the SNP allele (0, 1, or 2).
`grps`	a numeric vector giving the populations of the nAnimals

Details

This function is really just a wrapper for the `Fstat()` function, taken from the (now unsupported) Geneland package.

Value

A numeric matrix containing the pairwise Fst values between all populations.

Authors

Arnaud Estoup for original code in Turbo Pascal. Translation in Fortran and interface with R by Gilles Guillot (for Geneland package). Scott Foster for latest wrapper for stockR package.

Author(s)

Scott D. Foster

Examples

```
set.seed(717)
data <- sim.stock.data( nAnimals=50, nSNPs=500, nSampleGrps=50, K=5, ninform=50)
calcFst( data, attributes( data)$grps)
```

`plot.stockBOOT.object` *Plots results of a stockR analysis (a barplot-like representation)*

Description

For a given fitted stockBOOT object, visualise the membership of each individual to each stock.

Arguments

`x`	A stockBOOT.object, typically obtained from a call to `stockBOOT`. Note that the accuracy of the results will depend on the number of bootstrap resamples used to create the stockBOOT.object.
`locations`	A data.frame with two columns and the number of rows defined by the number of individuals in the stockBOOT.object. Important to note that the ordering of individuals must be the same in the locations data.frame as in the stockBOOT.oject (otherwise garbage will be given). The first column in the locations data.frame is the 'region' from which an individual was sampled. The second column gives information about how to plot the 'regions' – in particular their plotting order. Assign a low number to regions that you want plotted on the left side of the page and a high number for those on the right. Default argument value (for locations) is NULL, in which case all individuals are assumed to come from a single region (plot is for a single block).
`plotTitle`	The main title (text) to give the plot.
`CI.width`	The width of the confidence intervals to take transparency from. See details.
`region.lwd`	The line width of the box around the regions (groups of individuals).
`...`	Other parameters to be passed through to plotting functions.

Details

These plots give the probability of an individual to be assigned to each group (stock) identified by the mixture model, through stockSTRUCTURE and stockBOOT. This is an assignment probability, not a admixture proportion that is obtained from the STRUCTURE program (for example).

The intensity of the colour is given by the amount of uncertainty in the estimate of the probabilities – more solid colours are less uncertain (or more certain if you like to avoid double negatives). Very faint colours have a 100*CI.wdith% confidence interval width that is essentially 1 (so nothing is known about the probabilities).

Value

NULL

method

plot.stockBOOT.object(x, locations=NULL, plotTitle=NULL, CI.width=0.95, region.lwd=3.5, ...)

Author(s)

Scott D. Foster

References

Foster, S.D., Feutry, P., Grewe, P.M., Berry, O, Hui, F.K.C. and Davies, C.R. (in press) Reliably Discriminating Stock Structure with Genetic Markers: Mixture Models with Robust and Fast Computation. Molecular Ecology Resources

See Also

stockSTRUCTURE, stockBOOT

Examples

```
  ##This example will take a little while to run.
 #This should be challenging as there are actually 2 stocks and we fit a model with 3.
tmpDat1 <- sim.stock.data( nAnimals=100, nSNP=5000, nSampleGrps=100, K=2, ninform=5000,
                           sds=c(alpha=1.6,beta.inform=0.75,beta.noise=0.0005))
  #EM estimation from Kmeans starting values
tmp <- stockSTRUCTURE( tmpDat1, sample.grps=attr(tmpDat1,"sampleGrps"), K=3, start.grps=NULL)
  #in general, you'll want to use as many cores as possible, or close to.
  #mc.cores=1 is used here to please the CRAN submission checks
  tmpBOOT <- stockBOOT( tmp, B=100, mc.cores=1)
print( round( apply( tmpBOOT$postProbs, FUN=quantile, MARGIN=1:2, probs=c(0.025,0.975))), 5)
#Note that, in this case, the posterior probabilities are not very informative; they could
 #be anywhere between 0 and 1. There are likely to be a few individuals, of course, where
#they have a very low chance of belonging to a particular stock (and this is chance). There
  #may even some individuals that get assigned to a group with almost certainty.
  #Let's visualise it.
  plot( tmpBOOT, locations=NULL, plotTitle="Data contains 2 groups, model fits 3")
  #You can try it with 2 groups.

  #Let's now pretend that there are sampling regions
  plot( tmpBOOT, locations=data.frame( locations=rep(1:4, each=25),
order=rep( c(4,3,1,2), each=25)), plotTitle="Plot with grouping")
```

`sim.stock.data` *Simulates SNP data according to stockSTRUCTURE model*

Description

For given sample characteristics (number of sampling groups, number of animals, number of SNPs and number/size of stocks) simulates some SNP data.

Usage

```
sim.stock.data(nAnimals, nSNPs, nSampleGrps, K, ninform = nSNPs,
  means=c(alpha=-1.9,beta=0), sds=c(alpha=1.6,beta.inform=0.85,
  beta.noise=0.0005), minStockSize=1)
```

Arguments

nAnimals	integer giving the number of animals to simulate. No default, must be specified.
nSNPs	integer giving the number of SNPs to generate. No default, must be specified.
nSampleGrps	integer giving the number of sampling groups. Must be less than nAnimals. No default, must be specified.
K	integer giving the number of stocks
ninform	integer giving the number of informative SNPs. That is the first ninform SNPs will discriminate the stocks and the remainder will not.
means	a named two-element numeric vector consisting of the mean of the intercepts (named "alpha") and the mean of the deviations (named "beta"). The latter should always be zero (the betas will get standardised in any case).
sds	a names three-element numeric vector consisting of the standard deviations of the intercepts (named "alpha"), the sds of the betas that are differentiating between stocks (named "beta.inform"), and the sds of the SNP effects that are not differentiating between stocks (named "beta.noise").
minStockSize	the size of the smallest stock group allowed. That is the size within the sample, not the actual population size.

Details

The proportions of each stock in the sample is taken as a single draw from a flat Dirichlet distribution (which is flat over the simplex). The expected frequencies, in each stock, are generated on the logit scale. The mean frequency, for each SNP, is drawn from a random normal with mean means["alpha"] and sd sds["alpha"]. Deviations from the mean, for the first ninform SNPs, are generated from a normal with mean means["beta"] and sd=sds["beta.inform"]. For other SNPs these deviations are drawn from a normal with mean means["alpha"] and sd sds["beta.noise"] (typically really small). These expectations are then inverse logit trasnsformed and SNP data generated using binomial sampling with 2 trials. There was a good reason for choosing these means and standard deviations for generating SNP expectations, but I can't remember what it was...

Value

A numeric matrix containing SNP allele frequencies (in rows) for each animal (in columns). The matrix has attributes, "grps" for the stock groups and "sampleGrps" for the sampling groups.

Author(s)

Scott D. Foster

See Also

stockSTRUCTURE

Examples

```
#a data set with 100 individuals, from 100 different sampling groups (so no individual
#can be assumed a priori to belong to the same stock as any other individual), with
#5000 SNP markers. There are 3 stocks and only 200 of the 500 SNP markers are
#differentiated between stocks.
myData <- sim.stock.data( nAnimal=100, nSNP=5000, nSampleGrps=100, K=3, ninform=200)
print( dim( myData))  #should be 5000 x 100
```

`stockBOOT` *Calculates bootstrap estimates of stockSTRUCTURE model*

Description

For a given fitted stockSTRUCTURE model determine uncertainty in parameter values (and outputs) using Bayesian Bootstrap.

Usage

```
stockBOOT( fm, B=100, mc.cores=NULL)
```

Arguments

`fm`	A stockSTRUCTURE.object, typically obtained from a call to `stockSTRUCTURE`. This object MUST NOT be run with the small object option. So, in the initial model fit make sure that small.object=FALSE in the control list (this is the default).
`B`	The number of (Bayesian) bootstrap resamples to perform. Default is 100 but for serious applications more will be needed.
`mc.cores`	The number of parallel processes to perform at once. Default is NULL, in which case the function will determine the number of cores available and use them all.

Details

Bootstrapping is done here using the Bayesian Bootstrap of Rubin (1981) as described in Foster et al (2018). Briefly, the sampling distribution of the model's parameters is obtained by taking weighted re-samples of the original data and then re-estimating the model. The variability between the re-estimations is the same as the sampling variability.

Value

An object of class stockBOOT.object. This is a list with the following components

`condMeans`	a three dimensional array with each slice corresponds to the mean frequencies for each allele in each stock for each bootstrap resample.
`pis`	a three dimensional array where each slice corresponds to the proportion, in the data, of each of the stocks for each bootstrap resample. Small numbers imply smaller stocks. Note that sum(pi)=1 within each resample.

postProbs	a three dimensional array where each slice is the soft-classification of sample.grps to stocks for each bootstrap resample. Be aware that the order will be according to levels(sample.grps) and not the natural ordering that you may have imagined. Blame this on R's internal representation of factors. Consider using mixedsort() from the gtools package.
margLogl	a vector giving the marginal log-likelihood obtained at the parameter estimates for each bootstrap sample.

Author(s)

Scott D. Foster

References

Foster, S.D., Feutry, P., Grewe, P.M., Berry, O, Hui, F.K.C. and Davies, C.R. (in press) Reliably Discriminating Stock Structure with Genetic Markers: Mixture Models with Robust and Fast Computation. Molecular Ecology Resources

Rubin, D.B. (1981) The Bayesian Bootstrap. The Annals of Statistics *9*:130–134.

See Also

stockSTRUCTURE

Examples

```
##This example will take a little while to run.
#This should be very challenging as stock differentiation is non-existant (K=1).
tmpDat1 <- sim.stock.data( nAnimals=100, nSNP=5000, nSampleGrps=100, K=1, ninform=5000,
                                  sds=c(alpha=1.6,beta.inform=0.75,beta.noise=0.0005))
#EM estimation from Kmeans starting values
tmp <- stockSTRUCTURE( tmpDat1, sample.grps=attr(tmpDat1,"sampleGrps"), K=3, start.grps=NULL)
#in general, you'll want to use as many cores as possible, or close to.
#mc.cores=1 is used here to please the CRAN submission checks
tmpBOOT <- stockBOOT( tmp, B=100, mc.cores=1)
print( round( apply( tmpBOOT$postProbs, FUN=quantile, MARGIN=1:2, probs=c(0.025,0.975))), 5)
#Note that, in this case, the posterior probabilities are not very informative; they could
#be anywhere between 0 and 1. There are likely to be a few individuals, of course, where
#they have a very low chance of belonging to a particular stock (and this is chance). There
#may even some individuals that get assigned to a group with almost certainty.
```

---

stockSTRUCTURE *Finds stock structure for sampling groups of animals*

Description

For a given set of markers, scored for each animal (in sampling groups), determine likely stock structure using mixture models.

Usage

```
stockSTRUCTURE(SNPdata = NULL, sample.grps = as.factor(1:ncol(SNPdata)),  K = 3,
      weights = rep(1, nlevels( sample.grps)), start.grps = NULL,  control = NULL)
```

Arguments

SNPdata	a numeric matrix of dimension (number of SNPs X number of individuals). As the dimension suggests, each row corresponds to an SNP marker and each column an individual. The entries are the number of copies of a marker present in the animal. So, 0 for no copies (aa), 1 for one copy (Aa), and 2 for two copies (AA). Columns that are constant for all animals are removed.
sample.grps	a factor (or something that can be converted into a factor) of length ncol(SNPdata). This gives the sampling the allocation of animals (columns of SNPdata) to sampling groups. The animals must be ordered the same in sample.grps and in SNPdata. The default sample.grps is to have each animal in its own group (so that no knowledge of breeding groups is assumed). Note that if sample.grps is specified, then the function's output will have (internal) ordering which is based on the alphabetical listing of sample.grps.
K	an integer giving the number of stocks to partition the data into
weights	a numeric vector of length ncol(SNPdata). Gives the weighting for each animal to the analysis. Default is to weight all animals equally. You should have a really good reason not to use equal weights.
start.grps	a numeric vector of length ncol(SNPdata) or NULL. If NULL the (EM-)optimisation algorithm will be started from groups found using the K-means algorithm. If not NULL then these groups will be used to start the EM optimiser. For most purposes this argument should be set as NULL. If specified, then the model's likelihood is less likely to be maximised and the major signals in the data may not be found. This is not a prior, not in the sense of a Bayesian analysis in any case. Please do not supply the groups you think are in the data as starting values – you are likely to just be confirming your own beliefs.
control	a list of control parameters for initialisation and optimisation. See below for details.

Details

Control arguments, and especially their defaults, will vary depending on the type of optimisation used (only a handful are common). The arguments are

Common to all methods

small.object a boolean indicating whether the return object should be made small (ie no returning of data etc). Default is FALSE, so that the return object is normal size.

quiet a boolean indicating whether reporting should be performed throughout the estimation process. Default is FALSE (not quiet) so that reporting is performed. Users could also use `suppressMessages` if they prefer.

method a character string. One of "Kmeans.EM" (default), "SA.EM", "EM", or "DA.EM". These specify how the optimisation is performed. "SA.EM" implements the initialisation-then-optimisation strategy in Foster et al (in prep). "EM" performs the EM algorithm from random

starts (unless user specifies the start location), this is similar to Chen et al (2006). "DA.EM" implements the deterministic annealing algorithm proposed in Zhou and Lange (2010) from random starts, or user specified start.

For "Kmeans.EM" (Default and recommended) method

nKmeanStart the number of K-menas groupings to perform to obtain starting values. Default (NULL) corresponds to 25 starts.

nKPCA the SNP data is rotated, prior to initial K-means clustering, using PCA. This argument defines the number of PCAs to use. The default is nKPCA=min(nFish, nSNP, 100). The argument is always checked and the value of min(nFish, nSNP, nKPCA) is used.

EM.eps the absolute convergence tolerance for the EM-algorithm. Default is 1e-5, that is successive differences in parameters have to be very small before converged is reached.

EM.maxit the maximum number of iterations for the EM-algorithm. Default is 100.

EM.minit the minimum number of iteraction for the EM-algorithm. Default is 1. EM.minit is really only important for when EM.tau.step < 1.

EM.tau.step the step-size of the initial update for the posterior probabilities. Default is 1. If less than 1 (say 0.5), then the step size is less (half say) than the size that the EM-algorithm suggests. EM.tau.step increases linearly to 1 after EM.minit steps.

tau.init.max After the initial hard-clustering, the groupings are made soft by specifying posterior probabilities that are less than 1 and greater than 0. The default is given in Foster et al (in prep), but makes sure that the hard clustering recieves twice the probability mass than other stocks.

For "SA.EM" method

SANN.temp the initial temperature for simulated annealing. Default is half the number of sampling groups. This means that half of the sampling groups can be swapped to new groups in the first iteration.

SANN.maxit the maximum number of iterations for the simulated annealing. Default is 5000, which is probably overkill for many problems.

SANN.tmin the minimum number of swaps per iteration (ie once the annealing has run for a while). The default is max(nSampGrps %/% 20, 1) where nSampGrps is the number of sampling groups.

SANN.nreport the number of iterations to do before printing out report (if printing at all). Default is 100.

SANN.trace a boolean indicating if any trace information should be given. See optim.

EM.eps the absolute convergence tolerance for the EM-algorithm. Default is 1e-5, that is successive differences in parameters have to be very small before converged is reached.

EM.maxit the maximum number of iterations for the EM-algorithm. Default is 100.

EM.minit the minimum number of iteraction for the EM-algorithm. Default is 1. EM.minit is really only important for when EM.tau.step < 1.

EM.tau.step the step-size of the initial update for the posterior probabilities. Default is 1. If less than 1 (say 0.5), then the step size is less (half say) than the size that the EM-algorithm suggests. EM.tau.step increases linearly to 1 after EM.minit steps.

tau.init.max After the initial hard-clustering, the groupings are made soft by specifying posterior probabilities that are less than 1 and greater than 0. The default is given in Foster et al (in prep), but makes sure that the hard clustering recieves twice the probability mass than other stocks.

For "EM" method

As per last five entries for "SA.EM" (an having a random start rather than an initial clustering).

For "DA.EM" method

EM.eps the absolute convergence tolerance for the EM-algorithm. Default is 1e-5, that is successive differences in parameters have to be very small before converged is reached.

EM.maxit the maximum number of iterations for the EM-algorithm. Default is 100.

EM.minit the minimum number of iteraction for the EM-algorithm. Default is 25. EM.minit is the number of steps required to reach the non-cooled log-likelihood surface.

EM.tau.step the step-size of the initial update for the posterior probabilities. Default is 1. If less than 1 (say 0.5), then the step size is less (half say) than the size that the EM-algorithm suggests. EM.tau.step increases linearly to 1 after EM.minit steps.

tau.init.max After the initial hard-clustering, the groupings are made soft by specifying posterior probabilities that are less than 1 and greater than 0. The default is to make the hard clusterings ever-so-slightly more likely than others.

DA.nu.init the initial cooling parameter. Default is 1e-4. Algorithm seems particularly sensitive to this choice. The cooling parameter nu is increased gradually until it reaches 1 (max) at EM.tau.step iterations. See Zhou and Lange (2010) for details.

Value

An object of class stockSTRUCTURE.object. This is a list with the following components

`condMeans`	the mean frequencies for each allele in each stock
`pis`	the proportion, in the data, of each of the stocks. Small numbers imply smaller stocks. Note that sum(pi)=1.
`margLogl`	the marginal log-likelihood obtained at the parameter estimates.
`postProbs`	the soft-classification of sample.grps to stocks. Be aware that the order will be according to levels(sample.grps) and not the `natural` ordering that you may have imagined. Blame this on R's internal representation of factors. Consider using mixedsort() from the gtools package.
`K`	the number of stocks (specified by user).
`data`	a list containing all the bits and pieces of data used in the analysis.
`init.grps`	the initial stock structure for starting the final optimisation (via the EM algorithm).
`constantSNPs`	a boolean vector giving the locations of the SNP markers with no variation.
`margLoglTrace`	a trace of the marginal log-likelihood throughout the final optimisation (EM algorithm).
`control`	the control parameters used in estimation. See below for details.

Note that only condMeans, pis, margLogl, postProbs, and K are returned if control$small.object=TRUE is specified. See below for details. Note that if sample.grps is specified, then the value of the function's output is orientated for the alphabetical ordering of sample.grps. This is done to help keep track of the groupped results when the number of sample groups is less than the number of individuals.

Author(s)

Scott D. Foster

References

Foster, S.D., Feutry, P., Grewe, P.M., Berry, O, Hui, F.K.C. and Davies, C.R. (in press) Reliably Discriminating Stock Structure with Genetic Markers: Mixture Models with Robust and Fast Computation. Molecular Ecology Resources

See Also

sim.stock.data, stockBOOT

Examples

```
set.seed(727)
tmpDat1 <- sim.stock.data( nAnimals=100, nSNP=5000, nSampleGrps=100, K=3, ninform=5000,
                                    sds=c(alpha=1.6,beta.inform=0.75,beta.noise=0.0005))
#This should not be too challenging as stock differentiation is quite large.
print( calcFst( tmpDat1, attributes( tmpDat1)$grps))
#EM estimation from Kmeans starting values
tmp <- stockSTRUCTURE( tmpDat1, sample.grps=attr(tmpDat1,"sampleGrps"), K=3, start.grps=NULL)
#an easy gold-standard for simulations (only know starting values as this is simulated data)
tmp1 <- stockSTRUCTURE( tmpDat1, sample.grps=attr(tmpDat1,"sampleGrps"), K=3,
                      start.grps=attr( tmpDat1,"grps"), control=list( method="EM"))
#an easily misled estimation method
tmp2 <- stockSTRUCTURE( tmpDat1, sample.grps=attr(tmpDat1,"sampleGrps"), K=3,
                                         start.grps=NULL, control=list( method="EM"))
#combine into a single object to investigate results
grpings <- cbind( attributes( tmpDat1)$grps, apply( tmp$postProbs, 1, which.max),
              apply( tmp2$postProbs, 1, which.max), apply( tmp1$postProbs, 1, which.max))
colnames( grpings) <- c("True","Kmeans.EM Estimated","EM Estimated","Estimated From TRUE Start")
print( "How did we go with the stock identification?")
print( grpings)
#up to label switching this looks good, except for the EM from random start method (a few
#confused individuals for the second and thrid stocks)
```

Index