Package 'happy'

August 22, 2006

Version 2.0.4

Date 2006-07-06

Title Quantitative Trait Locus genetic analysis in Heterogeneous Stocks

Author Richard Mott <Richard.Mott@well.ox.ac.uk>

Maintainer Richard Mott <Richard.Mott@well.ox.ac.uk>

Depends R

Description happy is an R interface into the HAPPY C package for fine-mapping Quantitative Trait Loci (QTL) in Heterogenous Stocks (HS). An HS is an advanced intercross between (usually eight) founder inbred strains of mice. HS are suitable for fine-mapping QTL. The happy package is an extension of the original C program happy; it uses the C code to compute the probability of descent from each of the founders, at each locus position, but the happy packager allows a much richer range of models to be fit to the data.

License GPL version 2 or newer

URL http://www.r-project.org, http://www.well.ox.ac.uk/happy

R topics documented:

`Happy` *Quantitative Trait Locus analysis in Heterogeneous Stocks*

Description

happy is an R interface into the HAPPY C package for fine-mapping Quantitative Trait Loci (QTL) in Heterogenous Stocks (HS). HAPPY uses a multipoint analysis which offers significant improvements in statistical power to detect QTLs over that achieved by single-marker association. An HS is an advanced intercross between (usually eight) founder inbred strains of mice. HS are suitable for fine-mapping QTL. The happy package is an extension of the original C program happy; it uses the C code to compute the probability of descent from each of the founders, at each locus position, but the happy packager allows a much richer range of models to be fit to the data.

happy() is used to initialise input files and perform dynamic programming in C. Model fitting is then performed by subsequent calls to hfit() etc. Input file foramt is described at `http://www.well.ox.ac.uk/happy`

Usage

```
happy( datafile, allelesfile, generations=200, standardise=FALSE,
        phase="unknown", file.format="happy", missing.code="NA",
do.dp=TRUE, min.dist=1.0e=5, mapfile=NULL )
happy.matrices( h )
happy.save( h, file )
```

Arguments

`datafile`	name of the text file containing the genotype and phenotype data in HAPPY format
`allelesfile`	
`generations`	the number of breeding generations in the HS
`standardise`	if TRUE then the phenotype is transformed to have mean 0 and variance 1. This does not affect the identification of QTL but can make the interpretation of effects easier.
`phase`	If phase=="unknown" then the phase of the genotypes is unknown and no attempt is made to infer it. If phase="estimate" then it is estimated using parental genotype data when available. If phase="known" then it is assumed the phase of the input genotypes is correct i.e. the first and second alleles in each genotype for an individual are on the respectively the first and second chromosomes. Where phase is known this setting should increase power, but it will cause erroneous output if it is set when the data are unphased. If phase="estimate" then file.format="ped" is assumed automatically, because the input data file must be in ped-file format in order to specify parental information.
`file.format`	The format of the genotype file. Either "happy" (the default) or "ped". "happy" files do not contain any pedigree information. They are structured so that one record corresponds to an individual. The first two fields are the subject id (unique) and the phenotype value. The remaining fields are the N genotypes for the subject (where N is the number of markers specified in the alleles file), arranged in 2N fields all separated by spaces.

The "ped" file format is similar except that in place of the two columns "id" and "phenotype" in the original "happy" file format there should be six columns "family", "id", "mother", "father", "gender", "phenotype". Note that the "family" field can be constant so long as the id's are all unique. The resultant name of each subject is constructed as "family.id", which must be unique. The genotype data then follow as in "happy" format. Note that if phase="estimate" then file.format="ped" automatically; it is only necessary to use this option when the input file format is "ped" but it is desired not to make any use of pedigree information.

`missing.code` The code for a missing allele in the input file. Defaults to "NA". Note that old HAPPY files use "ND"

`do.dp` A switch that turns off the dynamic programming part of happy. By default dynamic programmig is performed. The only reason to turn this off is when only the genotypes are required.

`min.dist` The minimum genetic distance (in centiMorgans) allowed between adjacent merkers. Markers positioned closer than min.dist in the input file are treated as being min.dist cM apart. This prevents problems with markers at the same position, and which HAPPY cannot process.

`mapfile` Optional name of text file containing the physical base-pair coordinates of the markers (the alleles file only contains the genetic map in centiMorgans). The file format has three columns named "marker", "chromosome" and "bp". This file is not required unless genome cache objects are to be made (see save.genome()).

`h` An object of class "happy"

`file` Name of file in which to save data

Details

Biological Background

Most phenotypes of medical importance can be measured quantitatively, and in many cases the genetic contribution is substantial, accounting for 40% or more of the phenotypic variance. Considerable efforts have been made to isolate the genes responsible for quantitative genetic variation in human populations, but with little success, mostly because genetic loci contributing to quantitative traits (quantitative trait loci, QTL) have only a small effect on the phenotype. Association studies have been proposed as the most appropriate method for finding the genes that influence complex traits. However, family-based studies may not provide the resolution needed for positional cloning, unless they are very large, while environmental or genetic differences between cases and controls may confound population-based association studies.

These difficulties have led to the study of animal models of human traits. Studies using experimental crosses between inbred animal strains have been successful in mapping QTLs with effects on a number of different phenotypes, including behaviour, but attempts to fine-map QTLs in animals have often foundered on the discovery that a single QTL of large effect was in fact due to multiple loci of small effect positioned within the same chromosomal region. A further potential difficulty with detecting QTLs between inbred crosses is the significant reduction in genetic heterogeneity compared to the total genetic variation present in animal populations: a QTL segregating in the wild need not be present in the experimental cross.

In an attempt to circumvent the difficulties encountered with inbred crosses, we have been using a genetically heterogeneous stock (HS) of mice for which the ancestry is known. The heterogeneous stock was established from an 8 way cross of C57BL, BALB/c, RIII, AKR, DBA/2, I, A and C3H/2 inbred strains. Since its foundation 30 years ago, the stock has been maintained by breeding from 40 pairs and, at the time of this experiment, was in its 60th generation. Thus each chromosome

from an HS animal is a fine-grained genetic mosaic of the founder strains, with an average distance between recombinants of 1/60 or 1.7 cM.

Theoretically, the HS offers at least a 30 fold increase in resolution for QTL mapping compared to an F2 intercross. The high level of recombination means that fine-mapping is possible using a relatively small number of animals; for QTLs of small to moderate effect, mapping to under 0.5 cM is possible with fewer than 2,000 animals. The large number of founders increases the genetic heterogeneity, and in theory one can map all QTLs that account for progenitor strain genetic differences. Potentially, the use of the HS offers a substantial improvement over current methods for QTL mapping.

Problem Statement and Requirements

1. HAPPY is designed to map QTL in Heterogeneous Stocks (HS), ie populations founded from known inbred lines, which have interbred over many generations. No pedigree information is required.
2. Obviously, phenotypic values for the trait must be known for all individuals. It is preferable that these are normally distributed because HAPPY uses Analysis of Variance F statistics to test for linkage (however, a permutation test can be used instead).
3. For each genotyped marker, it is necessary to know the ancestral alleles in the inbred founders (which by definition must be homozygous), and the genotypes from the individuals in the final generation.
4. The chromosomal position in centiMorgans of each marker must be known.
5. Missing data are accomodated provided these are due to random failures in the genotyping and not selective genotyping based on the trait values (however, it is permissible to selectively genotype all the markers provided the same individuals are genotyped at each locus).

What HAPPY does

HAPPY's analyis is essentially two stage; ancestral haplotype reconstruction using dynamic programming, followed by QTL testing by linear regression:

- Assume that at a QTL, a pair of chromosomes originating from the progenitor strains, labelled s, t contribute an unknown amount T_{st} to the phenotype. In the special case where the contribution from each chromosome is additive at the locus then $T_{st} = T_s + T_t$,say.
- a test for a QTL is equivalent to testing for differences between the T's.
- A dynamic-programming algorithm is used to compute the probability F_{iLst} that a given individual i has the ancestral alleles s, t at locus labelled L, conditional upon all the genotype data for the individual. Then the expected phenotype is

$$y = \sum_{st} T_{st} F_{iLst}$$

  , and the T's are estimated by a linear regression of the observed phenotypes on these expected values across all individuals, followed by an analysis of variance to test whether the progenitor estimates differ significantly.
- The method's power depends on the ability to distinguish ancestral haplotypes across the interval; clearly the power will be lower if all markers in a region have the same type of non-informative allele distribution, but the markers can share information where there is a mixture.
- All inference is based on regression of the phenotypes on the probabilities of descent from the founder loci, F_{nst}.

Although the models are presented here in the linear model framework (ie least-squares estimation, with ANOVA F-tests), it is of course straighforward to extend them to R's generalised linear model framework. Multivariate analysis is also possible.

It is straighforward to fit models involving the effects of multiple loci and of covariates. It is easiest to see this by rewriting the problem in standard linear modelling notation. Consider first the case of fitting a QTL at a locus, L. Let $\mathbf{y}$ be the vector of trait values. Let $\mathbf{X_L}$ be the design matrix for fitting a QTL at the locus L. Let $\mathbf{t_L}$ be the vector of parameters to be estimated at the locus. For an additive QTL, the paramters are the strain effect sizes; for a full interaction model there is a paramter for every possible strain combination. Then the one-QTL model is

$$\mathbf{E}(\mathbf{y}) = \mathbf{X_L}\mathbf{t_L}$$

There are $S(S-1)/2$ parameters to be estimated in a full model allowing for interactions between the alleles within the locus, and $S-1$ parameters in an additive model. For the full model, the i,j'th element of the design matrix $\boldsymbol{X}$ is related to the strain probabilities thus:

$$\boldsymbol{X_{Lij}} = \boldsymbol{F_{iLst}}$$

, where

$$j(s,t) = \min(s + S(t-1), t + S(s-1)$$

and for the additive model

$$\boldsymbol{X_{Lij}} = \sum_{\boldsymbol{s}} \boldsymbol{F_{iLsj}}$$

More complex models

To add covariates to the model (for instance sex or age) we add additional columns $\boldsymbol{C}$ to the design matrix:

$$\mathbf{E}(\mathbf{y}) = [\mathbf{X_L}\|\mathbf{C}]\,(\mathbf{t_L}\|\mathbf{c})$$

where $\mathbf{C}$ is a design matrix representing the covariates of interest, and $\mathbf{c}$ are the parameters to be estimated.$(\mathbf{t_L}\|\mathbf{c})$ represents the vector formed by adjoining the vectors t_L and c. Note that at present $\boldsymbol{C}$ must be a numeric matrix: factors must be explicitly converted into columns of dummy variables.

Similarly to fit an additional locus K we adjoin the design matrix $\boldsymbol{X_K}$, for example:

$$\mathbf{E}(\mathbf{y}) = [\mathbf{X_L}\|\mathbf{X_K}\|\mathbf{C}]\,(\mathbf{t_L}\|\mathbf{t_K}\|\mathbf{c})$$

(this is essentially **composite interval mapping**). The happy package allows the inclusion of arbitrary covariate matrices, which can include other loci; new loci are then tested to see if they significantly improve the fit conditional upon the presence of the covariates. In this way we can analyse any number of linear combinations of loci and covariates.

Epistasis, or the interaction between loci, is supported as well. At present the package can test for interactions between unlinked loci, but not linked loci. The test compares the fit between the sum of the additive contributions from each locus and the interaction. This is accomplished as follows: Let X_L, X_K be the design matrices for the loci L, K. Let m_L be the number of columns in X_L Then form a matrix X_{LK} whose $m_L m_K$ columns are formed by multiplying the elements in each pair of columns in the original matrices.

Merging Strains

An important feature of the happy package is the suite of functions to merge strains together. The models described above (particularly the full interaction models) have the disadvantage that the fits sometimes involve a larger number of parameters, with many degrees of freedom. This is particularly true for full non-additive models and for epistasis. For example in an 8-strain HS, 28 df are required to fir a full model for a single locus. Large numbers of degrees of freedom have two problems: firstly the models may become overspecified, and secondly even if there are plenty of degrees of freedom for the residual error, the power to detect an effect is diluted.

A partial solution is to note that since most polymorphisms are diallelic (eg SNPs), it makes sense to group the strains according to their alleles at some polymorphic locus. This corresponds to operating with design matrices in which certain columns are combined by adding their corresponding elements together. A diallelic merge reduces the number of degrees of freedom dramatically: only 3 df (instead of 28df) are required to fit a full model at a locus (and only 1 df instead of 7df for the addtive model), and an epistatic interaction between two merged loci will involve only 3df (additive) or 8df (full).

Value

happy() returns an object of type happy, which should be passed onto model-fitting functions such as hfit(). A happy object 'h' is a list with a number of useful members:

`strains`	a character vector containing the names of the founder strains
`markers`	a character vector containing the names of the markers, in map order
`map`	a numeric vector containing the map coordinates in centiMorgans of the markers
`subjects`	a character vector containing the subject names
`phenotype`	a numeric vector containing the subject phenotypes
`handle`	a numeric index used internally by the C-code. Do not change.
`matrices`	a list of matrices used in model fitting (only created after a call to happy.matrices())
use.pedigrees	boolean variable indicating whether pedigree information was used to help determine the phase of the genotypes
phase.known	boolean variable indicating whether or not the phase of the genotypes is assumed to be known

happy.save() will save a happy object to a file so that it can be re-used in a later session with the load() command.

happy.matrices() is not normally called directly - its function is to copy all the dynamic-programming matrices created by a call to happy() from the underlying C memory space into R objects. The object returned is still a happy object, but with an additional component 'matrices'. It can be used in exactly the same way as a normal happy object except that the underlying C memory is no longer used. When a happy object is saved using happy.save() is is first converted by a call to happy.matrices(), and when a happy object is reloaded using load() it uses matrices stored in R memory. Thus these functions are a useful way to save computing time - the dynamic programming step need only be performed once, the data can be persisted to disk and then re-used (e.g. to analyse multiple phenotypes) at a later date.

Author(s)

Richard Mott

References

Mott R, Talbot CJ, Turri MG, Collins AC, Flint J. A method for fine mapping quantitative trait loci in outbred animal stocks. Proc Natl Acad Sci U S A. 2000 Nov 7;97(23):12649-54.

See Also

hfit(), mergefit(), happyplot()

Examples

```
## Not run: h <- happy('HS.data', 'HS.alleles', generations=200)
## Not run:  happy.save(h,'h.Rdata')
## Not run:  load('h.Rdata')
```

---

epistasis *Analysis of Epistasis between Markers*

Description

epistasis() will test for a statistical interaction between two sets of markers within the happy framework. The markers should be sufficiently far apart that they are unlinked (in practice 10cM for a 30 generation HS is sufficient). A partial F-test is performed to test if a model allowing for interactions fits better than a model in which each marker's contribution is additive between loci. Note that the effect of each marker within a locus can be either additive or full. Merging of strain is permitted.

epistasispair() is the same as epistasis() except that only one pair of markers is tested.

Usage

```
epistasis( h, markers1, markers2, merge1=NULL, merge2=NULL,
model='additive', verbose=FALSE, family='gaussian' )
epistasispair( h, marker1, marker2, merge1=NULL, merge2=NULL,
model='additive', verbose=FALSE, d1=NULL, d2=NULL, main1=NULL, main2=NULL ) )
```

Arguments

`h`	an object returned by a previous call to happy()
`markers1`	an array of marker names or indices
`markers2`	an array of marker names or indices
`marker1`	a single marker name or index
`marker2`	a single marker name or index
`merge1`	an optional merge object (returned by mergematrices()) determining how the strains should be merged together for the markers listed in marker1
`merge2`	an optional merge object (returned by mergematrices()) determining how the strains should be merged together for the markers listed in marker2
`model`	the type of model fitted at each locus. Either 'additive' or 'full'
`verbose`	switch controlling output to screen
`d1`	optional design matrix for the main effect of the first marker (saves computation time)
`main1`	optional log-P-value for the main effect of the first marker. NOTE: If d1 is not NULL then main1 *must* be set
`d2`	optional design matrix for the main effect of the second marker (saves computation time).

`main2` optional log-P-value for the main effect of the second marker. NOTE: If d2 is not NULL then main2 *must* be set

`family` The distribution of errors in the data. The default is 'gaussian'. This variable controls the type of model fitting. In the gauusian case a standard linear model is fitted using lm(). Otherwise the data are fitted as a generalised linear model using glm(), when the value of family must be one of the distributions hangled by glm(), such as 'binomial', 'gamma'. See family() for the full range of models.

Value

epistasis() returns a matrix with columns named 'marker1', 'marker2', 'main1', 'main2', 'main1+main2', 'main1*main2', 'main1.main2'. marker1 and marker2 are the names of the markers being compared in a given row, the remaining values are the ANOVA log-P-values of the main effects (main1 and main2), the combined additive effect (main1+main2), the addtive plus interaction (main1*main2) and the partial F of the interaction (main1.main2) after allowing for main1+main2. epistasispair() returns a list with the same fields.

Author(s)

Richard Mott

---

gfit *Fit a Gaussian Mixture Model to an object returned by happy()*

Description

gfit() fits a QTL model to a happy() object. The model is a mixture of Gaussians, each with a different mean, and corresponds loosely to the "full" model in hfit(). The difference is that hfit() fits the observed phenotype values to the expected phenotypes under a full model, whereas gfit() uses maximum likelihood to fit the observed phenotype values to a mixture of Gaussians, each with a different mean but common variance. The other functions in this suite are not usually called directly.

The statistical model fitted is as follows. Consider first the case of fitting a QTL at a locus, L. Let $\mathbf{y}$ be the vector of trait values. Let $\mathbf{X_L}$ be the design matrix for fitting a QTL at the locus L. Let $\mathbf{t_L}$ be the vector of parameters to be estimated at the locus. For an additive QTL, the paramters are the strain effect sizes; for a full interaction model there is a paramter for every possible strain combination. Then the one-QTL model is

$$\mathbf{E}(\mathbf{y}) = \mathbf{X_L t_L}$$

There are $S(S-1)/2$ parameters to be estimated in a full model allowing for interactions between the alleles within the locus, so the i,j'th element of the design matrix $\boldsymbol{X}$ is related to the strain probabilities thus:

$$\boldsymbol{X_{Lij} = F_{iLst}}$$

, where

$$j(s,t) = \min(s + S(t-1), t + S(s-1))$$

In the function hfit(), the observed phenotypes are regressed directly on the expected trait values. This is not an optimal procedure becuase the data are really a mixture:

$$y_i \tilde{\sum}_{st} F_{iLst} f((y_i - \beta_{Lst})/2\sigma_L^2)$$

where $f(x)$ is a standard Gaussian density. The $\boldsymbol{\beta_L}$ is a vector of mean trait values for the strain combinations. The parameters $\boldsymbol{\beta_L}, \sigma_L$ are estimated by maximum likelihood, and the test for the presence of a QTL at locus L is equivalent to the test that all the $\beta_{st} = \mu$, when the model collapses to a single Gaussian distribution.

The model-fitting is implemented in the function gfit() by an iterative process, rather like a simplified version of EM. Is is slower than hfit(), and generally gives similar results as far as overall QTL detection is concered,m but gives more accurate parameter estimates. The log-likelihood for the data is

$$L = \sum_i \log(\sum_j p_{ij} \frac{\exp(-\frac{(y_i-\beta_j)^2}{2\sigma^2})}{\sqrt{2\pi\sigma^2}})$$

$$= \sum_i \log(\sum_j p_{ij} \exp(-\frac{(y_i-\beta_j)^2}{2\sigma^2})) - \frac{N\log(2\pi\sigma^2)}{2}$$

Differentiating wrt to the parameters gives

$$\frac{\partial L}{\partial \sigma^2} = \sum_i \frac{\sum_j p_{ij}(y_i-\beta_j)^2 \exp(-\frac{(y_i-\beta_j)^2}{2\sigma^2})}{2\sigma^4 \sum_j p_{ij} \exp(-\frac{(y_i-\beta_j)^2}{2\sigma^2})} - \frac{N}{2\sigma^2}$$

$$\frac{\partial L}{\partial \beta_j} = -\sum_i \frac{p_{ij} \frac{(y_i-\beta_j)}{\sigma^2} \exp(-\frac{(y_i-\beta_j)^2}{2\sigma^2})}{\sum_j e_{ij}}$$

$$= \frac{1}{\sigma^2}\left(-\sum_i \frac{y_i e_{ij}}{\sum_j e_{ij}} + \beta_j \frac{\sum_i e_{ij}}{\sum_j e_{ij}}\right)$$

write

$$w_{ij} = \frac{p_{ij} \exp(-\frac{(y_i-\beta_j)^2}{2\sigma^2})}{\sum_j \frac{p_{ij} \exp(-\frac{(y_i-\beta_j)^2}{2\sigma^2})}}$$

then the mle satisfies

$$\hat{\sigma^2} = \frac{1}{N}\sum_i \frac{\sum_j p_{ij}(y_i-\beta_j)^2 \exp(-\frac{(y_i-\beta_j)^2}{2\sigma^2})}{\sum_j p_{ij} \exp(-\frac{(y_i-\beta_j)^2}{2\sigma^2})}$$

$$\hat{\sigma^2} = \frac{1}{N}\sum_i \sum_j \hat{w}_{ij}(y_i - \hat{\beta}_j)^2$$

$$\hat{\beta}_j = \frac{\sum_i \hat{e}_{ij} y_i}{\sum_i \hat{e}_{ij}}$$

and the log-likelihood is

$$\hat{L} = \sum_i (\log \sum_j \hat{e}_{ij}) - \frac{N\log(2\pi\hat{\sigma}^2)}{2}$$

Usage

```
gfit( h,eps=1.0e-4, shuffle=FALSE, method="optim" )
gaussian.loop( d, maxit=100, eps=1.0e-3, df=NULL )
gaussian.null( n, sigma2 )
gaussian.init( d )
gaussian.iterate( d, params )
gaussian.fn( p, d=NULL )
gaussian.gr( p, d=NULL )
```

Arguments

`h`	an object returned by a previous call to happy()
`shuffle`	boolean indicating whether the shuffle the phenotypes to perform a permutation test
`method`	The optimisation algorithm. Default is to use R's "optim" function, which uses derivative information. All other values of this argument will use an EM type iteration.
`d`	a list comprising two elements d, probs
`maxit`	the maximum number of iterations in the ML fitting
`eps`	the terminatation accuracy in the model fitting : the log likelihood must change by less than eps in successive iterations
`df`	the degress of freedom to use. If NULL then this is computed as the rank of the data
`n`	the number of observations with non-missing phenotypes
`sigma2`	
`params`	a list with two components, beta = the group means and sigma = the standard deviation
`p`	vector of paramters. For internal use only

Value

gfit() returns a matrix with columns "marker", "LogL", "Null", "Chi", "df", "Pval", "LogPval". Each row of the column describes the fit of the model for thecorresponding marker interval.

gaussian.loop() fits the model to a single marker and returns a list with the same elements as in hfit()

gaussian.iterate() performs a single iteration of the fitting process and returns a list with the updated LogL, beta, sigma, dbeta and dsigma

gaussian.init() intialises the parameters under the Null model, ie to the case where the means are all identical and the variance is the overal variance.

gaussian.null() returns the log-likelihood under the Null model

gaussian.fn() and gaussian.gr() are the function and gradient required by the optim function.

Author(s)

Richard Mott

See Also

happy, hprob

Examples

```
## An example session:
# initialise happy
## Not run: h <- happy('HS.data','HS.alleles')
# fit all the markers
## Not run: f <- gfit(h)
```

---

happy-internal *Internal Happy Functions*

Description

Internal functions for happy. These are not normally called by the user

Usage

```
matrixSquared( matrix1, matrix2 )
twofit( happy, marker1, marker2, merge1=NULL, merge2=NULL, model =
'additive', verbose=TRUE, family='gaussian' )
mfit( happy, markers, model='additive', mergematrix=NULL,
covariatematrix=NULL, verbose=TRUE , family='gaussian', variants=NULL )
condfit( happy, markers, condmarker, merge=NULL, condmerge=NULL,
model='additive',condmodel='additive', epistasis=FALSE, verbose=TRUE, family='ga
strain.effects( h, fit, family='gaussian' )
glmfit( formula=NA, family='gaussian' )
sdp( strains, alleles )
```

Author(s)

Richard Mott

---

happyplot *Plotting functions for happy model fits*

Description

happyplot() will plot along the genome the log P-value that a QTL is not found in a series of marker intervals. It accepts as input the results of hfit(), mfit() and mergefit(). mergeplot() is a convenience function for calling happlyplot() after a call to mergefit(), with several parameters set.

Usage

```
happyplot( fit, mode='logP', labels=NULL, xlab='cM', ylab=NULL, main=NULL, t='s'
mergeplot( fit, mergedata, mode='logP', xlab='bp', ylab=NULL, main=NULL,  t='p',
```

Arguments

fit	an object returned by a previous call to hfit(), mfit(), or mergefit()
mode	the mode of the plot - either 'logP', when the negative base-10 logarithm of the ANOVA P-value of plotted, or 'SS', when the fitting sums-of-squares is plotted
labels	optional matrix detailing marker labels to be drawn on the plot. The labels are written vertically above the plot, with vertical lines extending down into the plot area. labels is a matrix with two named columns 'marker', containing the marker names, and 'POSITION', containing the x-axis positions of the markers.
mergedata	(mergeplot() only). an object returned by a previous call to mergeprepare(). This is used to construct labels for plotting
xlab	the x-axis label
ylab	the y-axis label
main	the titke of the plot
t	the type of plot - either 'p', 'l', 's' or 'S', with the same meanings as in plot()
pch	the plotting character code, with the same meaning as in plot()

Value

A plot to the current graphics device is produced. For happyplot, if fit$permdataisnotNULL(i.e.hfit()wasrunusingth$ $pvaluerelativethewholeregion(ieadjustedforthenumberofmarkers)whilepoint.logpshowstheempiricallog-$ $pvalueforeachinterval.Iffit$permdata is NULL then the plots give the ANOVA logP values. If the model used in hfit() is 'additive' then the logP for the additive mocdel vs the null model is plotted; if the model is 'full' then the curves for the full, additive and partial F-test logP values are plotted.

Author(s)

Richard Mott

See Also

hfit(), mfit(), mergefit()

Examples

```
## Not run:  h <- happy( 'HS.data', 'HS.alleles' )
## Not run:  fit <- hfit( h, h$markers, model='full' )
## Not run:  happyplot( fit )
```

---

cache *Save HAPPY design matrices and genotypes to disk for rapid reloading*

Description

`save.genome()` will persist the happy design matrices or genotypes from a series of happy objects to disk as a collection of R delayed data packages (as implemented in the package `g.data`). `load.genome()` "reloads" the data, although the matrices are not actually loaded into memory until used. `load.markers()` loads in a specific set of design matrices or genotypes, as defined by their marker names. These functions are very usefiul when access to a random selection of loci across the genome is required, and when it would be impossible for reasons of space to load many entire HAPPY objects into memory. `save.happy()` saves a single happy object as a delayed data package. `the.chromosomes()` is a conveniemce funtion that generates a character vector of chromosome names.

Usage

```
save.genome( gdir, sdir , prefix, chrs=NULL, file.format="ped",
mapfile=NULL, generations=50)
genome <- load.genome( sdir, use.X=TRUE, chr=the.chromosomes(use.X=use.X) )
marker.list <- load.markers( genome, markers )
save.happy( h, pkg, dir, model="additive" )
```

Arguments

`gdir`	Path to the directory containing the genotype (.alleles and either .data or .ped) input files required to instantiate happy objects. This directory wil1 typically contain a pair of files for each chromosome of the genome of interest
`sdir`	Path to the directory where the data will be saved by `save.genome`, and read back by `load.genome()`.
`prefix`	Text fragment used to define the file names sought by `save.genome()`. An attempt is made to find files in `gdir` named like `chrN.prefix.*` where N is the chromosome number (1...20, X, Y), as defined in `chrs`.
`chrs`	List of chromosome numbers to be processed.
`use.X`	logical to determine whether to use X-chromsome data, in load.genome().
`file.format`	Defines the input genotype file format, either "ped" (Ped file format) or "happy" (HAPPY .data file format).
`mapfile`	Name of a text file containing the physicla (base pair) map for the genome. It contains three columns named "marker", "chromosome" and "bp". Every marker in the .alleles files should be listed in the file.
`generations`	The number of generations since the HS was founded (see happy()).
`genome`	An object returned by `load.genome()`.
`markers`	A vector of marker names. These names will be searched for in the `genome` object, and if found, their corresponding data retrieved.
`h`	A HAPPY object
`pkg`	The name of the R delayed data package to be created
`dir`	Name of directory to create a delayed data package for a single happy object
`model`	One of "additive", "full", "genotype"

.

Value

`save.genome()` returns NULL. `load.genome()` returns a list object which contains information about the delayed datapackages loaded, and how the markers are distributed between the packages. The list comprises two components, named "genome" and "subjects". The former is a datatable with columns "marker", "chromosome", "map", "ddp" which acts as a genome-wide lookup-table for each marker. The latter lists the subject names corresponding to the rows in the design matrices or genotypes. NOTE: The software assumes that all the chromosome-specific files used in `save.genome()` are consistent. i.e. the same subjects in the same order occur in each chromosome, and that a marker is only present once across the genome. `load.markers()` returns a list of data (either matrices or genotype vectors), each datum being named accoring to the relevant marker `the.chromosomes()` returns a character vector of chromosome names, like `c( "chr1", "chr2" ..., "chrX", "chrY" )`.

Author(s)

Richard Mott

See Also

happy(). Note that the function happy.save() differs from save.happy(), in that it saves a single happy object for reloading with `load()`; it does not use delayed data loading.

Examples

---

hdesign — *Extract design matrix or genotypes for a specific marker interval from a happy object*

Description

hdesign() will call C to extract the design matrix to fit a QTL to a marker interval. hprob() will call C to extract a raw probability matrix. hgenotype() will return the raw genotype data for a marker

Usage

```
hdesign( h, marker, model='additive', mergematrix=NULL )
hprob( h, marker=NULL )
hgenotype ( h, marker, collapse=FALSE, sep="" )
```

Arguments

`h`	an object returned by a previous call to happy()
`marker`	either a character string giving the name of the marker or the index of the marker in the array h$markers
`model`	either 'additive' (default) or 'full'. The additive design matrix returns an array with S columns, where S is the number of founder strains in the HS. The full design matrix returns a matrix with S(S-1)/2 columns, one for each combination of strains

`mergematrix`	an object returned by mergematrices, used to define sets of strains that are to be merged together. This is accomplished by adding the corresponding columns in the original design matrix.
`collapse`	a boolean variable indicating whether to collapse the alleles into a single genotype.
`sep`	the text to be used to separate the alleles if collapsed.

Value

hdesign() returns a design matrix d_{ij}, in which the ith row corresponds to the subject i, and the jth column to the corresponding strain or combination of strains or merged strains. () returns a matrix i th row corresponds to the subject i, and the $x = s * S + t$ th column contains the probability that the ancestral strains are s, t where S is the total number of strains. () returns a $Nx2$ matrix i th row corresponds to the subject i, and column 1 contains the first allele and column 2 the second allele at the marker specified, or (if `collapse=TRUE`) a vector of genotypes with the alleles pasted together.

Author(s)

Richard Mott

See Also

happy(), hfit()

Examples

```
## Not run: h <- happy( 'HS.data', 'HS.alleles', generations=200 )
## Not run: d <- hdesign( h, 1 ) ## the first marker interval
## Not run: d <- hdesign( h, 'D1MIT264') ## the marker interval with left-hand marker D1M
## Not run: d <- hdesign( h, 'D1MIT264', model='full' ) ## ditto with full design matrix
```

`hfit` *Fit a model to an object returned by happy()*

Description

hfit() fits a QTL model to a happy() object, for a set of markers specified. The model can additive or full (ie allowing for dominance effects). The test is a partial F-test. In the case of the full model two tests are performed: the full against the null, and the full against the additive.

hfit.sequential() performs an automated search for multiple QTL, fitting marker intervals in a sequential manner, and testing for a QTL conditional upon the presence of previously identified QTL. This is essentially forward selection of variables in multiple regression, and very similar to composite interval mapping.

pfit() is a conveneince function to fit several univariate phenotypes to the same genotype data.

normalise() is a convenience function to convert vector of phenotype data into a set of standard Gaussian deviates: the values are first ranked and then the ranks replaced by the corresponding percentiles in a standard Normal distribution. This may be used to help map traits that are strongly non-normal (or use the permute argument in hfit()).

Usage

```
hfit( h, markers=NULL, model='additive', mergematrix=NULL,
covariatematrix=NULL, verbose=FALSE, phenotype=NULL, family='gaussian', permute=

hfit.sequential( h, threshold=2,  markers=NULL, model='additive',
mergematrix=NULL, covariatematrix=NULL, verbose=FALSE, family='gaussian')

pfit( h, phen, markers=NULL, model='additive', mergematrix=NULL,
covariatematrix=NULL, verbose=FALSE, family='gaussian' )

normalise(values)
```

Arguments

`h`	an object returned by a previous call to happy()
`markers`	
`model`	specify the type of model to be fit. Either 'additive', where the contrinutions of each allele at the locus are assumed to act additively, or 'full', in which a term for every possible combination of alleles is included. The default 'additive' mimics the behaviour of the original C HAPPY software.
`mergematrix`	specify a mergematrix object (returned by mergematrices()) which describes which founder strains are to be merged. This is used to test whether merging strains reduces statistical significance (see mergematrices())
`covariatematrix`	Optional additional matrix of covariates to include in the mode. These may be additional markers (terurned by hdesign) or covariates such as sex, age etc.
`verbose`	control whether to print the results of the fits to the screen, or work silently (the default)
`threshold`	
`family`	
`permute`	
`phen`	
`phenotype`	
`values`	a numeric vector of phenotype values to transform into normal deviates (normalise() only)

Value

hfit() returns a list. The following components of the list are of interest:

`table`	a table with the log-P values of the F statistics. The table contains rows, one per marker interval. The columns are the negative base-10 logarithms of the F-test P-values that there is no QTL in the marker interval. In the case of model='full', the partial F-test that the full model is no better than the additive is also given. In the special case of model='additive' and verbose=TRUE the effects of all estimable strains are compared with a T-test, taking into account the correlations between these estimates. However, it should be noted that estimates of individual strain effects may be hard to interpret when some combinations of strains are indistinguishable, and it is possible for the overall F-statistic to be very significant whilst none of the strains appear to be significant, based on their T-statistics. The F-statistic is a better indicator of the true overall fit of the model.

`permdata` a list containing the results of the permutation analysis, or NULL if permute=0. The list contains the following elements:

N The number of permutations

permutation.dist A vector containing sorted ANOVA logP values from the N permutations. These values can be used to estimate the shape of the null distribution, and plotted e.g. using hist().

permutation.pval A data table containing the permutation p-values for each marker interval. The columns in the datatable give the position in cM, the marker name (left-hand marker in the interval), the original ANOVA logp, the permutation pval for this logp, and the log permutation P-value. Bothe global (ie region-wide) and point-wise pvalues are given. The Global pvalue for a marker interval is the fraction of times that the logP for the interval (either additive or full, depending on the model specified) is exceeded by the maximum logP in all intervals for permuted data. The pointwise pvalue is the fraction of permutation logP at the marker interval that exceed the logP for that interval.

The object returned by hfit() is suitable for plotting with happyplot()

pfit() returns a list of hfit() objects, the n'th being the fit for the n'th column (phenotype) in phen.

Author(s)

Richard Mott

See Also

happy

Examples

```
## An example session:
# initialise happy
## Not run: h <- happy('Hs.data','HS.alleles')
# fit all the markers with an additive model
## Not run: f <- hfit(h)
# plot the results
## Not run: happyplot(f)
# fit a non-additive model
## Not run: ff <- hfit(h, model='full')
# view the results
## Not run: write.table(ff,quote=F)
# plot the results
## Not run: happyplot(ff)
# use noramlised trait values
## Not run: ff <- hfit(h,phenotype=normalise(h$phenotypes))
# permutation test with 1000 permutations
## Not run: ff <- hfit(h, model='full', permute=1000)
```

mergelist *Create an object descrbing how to merge strains together*

Description

mergelist() is a convenience function which creates a list object suitable for use with mergematrices()

Usage

```
mergelist( strains, alleles )
```

Arguments

strains	a character vector of strain names
alleles	a character matrix with one row of strain/allele combinations. There must be a named column in the matrix corresponding to every strain name in strains. The value of the element is the allele for that strain

Value

a list of lists of strains describing how the strains are grouped together. For instance `mergelist <- list( A=list('AJ', 'BALB', 'AKR'), T=list('RIII','I', 'DBA', 'C57', 'C3H') )` divides the strains into two groups corresponding to the alleles A, T (the allele names are not important). It is essential that the all strain names match all the values in strains.

The object should be used as an input parameter to mergematrices()

Author(s)

Richard Mott

See Also

mergematrices()

mergematrices *Construct matrices used to merge together founder strains*

Description

mergematrices() creates a list containing two matrices suitable for pre-multiplying with an additive or full happy marker design matrix, in order to produce matrices with certain columns combined. These reduced matrices are used to test whether the specified merge reduces the significance of the fit. This function is not usually called directly but is used by mergfit() and hfit() megedpositionmatrix() will return either the merged design matrix or the mergematrices object corresponding to an object returned by mergeprepare()

Usage

```
 mergematrices( strains, mergelist=NULL, verbose=FALSE )
 mergedpositionmatrix( h, position, prepmerge, model='additive',
verbose=FALSE, design=TRUE )
```

Arguments

strains	character array of strain names
mergelist	a list of lists of strains describing how the strains are grouped together. For instance `mergelist <- list( A=list('AJ', 'BALB', 'AKR'), T=list('RIII','I', 'DBA', 'C57', 'C3H') )` divides the strains into two groups corresponding to the alleles A, T (the allele names are not important). It is essential that the all strain names match all the values in strains.
verbose	switch to determine whether to tell what is happening.
h	an object returned by a previous call to happy()
position	the coordinate of the polymorphism to be tested, ie an entry in prepmerge$testmarkerdata$POSITION
prepmerge	an object returned by mergeprepare()
model	the type of model to be fitted - 'additive' or 'full'
design	switch to make mergepositionmatrix return the mergematrix object rather than the merged design matrix

Value

mergematrices() and mergepositionmatrix() return an object comprising a list with two elements:

amat	the matrix to apply to an additive-model design matrix
imat	the matrix to apply to a full-model (interaction) design matrix

normal-bracket30bracket-normal

Author(s)

Richard Mott

See Also

happy(), mergefit(), hfit(), mergelist()

`mergeprepare` *Perform tests to determine whether individual polymorphisms could have given rise to a QTL*

Description

mergeprepare() reads in datafiles descrbing the locations and strain distribution patterns of polymorphisms (SNPs or otherwise) which have not necessarily been genotyped. The following tasks are performed:

1. the polymorphism data are read in from testmarkerfile.For each polymorphism the corresponding sketon marker interval is determined, based on their coordinates. Only those polymorphisms lying inside a skeleton marker interval are retained.
2. the coordinates (typically in bp rather than cM) of the genotyped markers are read in from markerposfile. Note that these coordinates are distinct from those in the cM map in h$map used in happy(). Only those markers listed in markerposfile that are also in h$markers are retained - the rest are discarded. The retained markers are referred to as 'skeleton' markers as they define a framework of genotype data that can us used to test the significance of other polymorphisms.

mergefit() tests each of the polymorphisms to see if it could be a QTL. It performs the following operations on each polymorphism:

1. The founder strains are merged together based on the strin distribution pattern for that polymorphism.
2. The merged data are used to fit a QTL in the corresponding skeleton marker interval
3. The unmerged data are used to fir a QTL in the corresponding skeleton marker interval.
4. The fits of the merged and unmerged data are compared with a partial F-test. If the unmerged data are significant but the merged data are not then there is evidence to reject the polymorphism as being associated with the trait.

fastmergefit() is a convenience function which perfroms a complete analysis without making a prior call to happy().

condmergefit() performs a conditional analysis in which each variant is fitted conditional upon every other variant being included in turn. This is VERY SLOW.

Usage

```
mergeprepare( h, markerposfile, testmarkerfile, verbose=FALSE )
mergefit( h, mergedata,  model='additive', covariatematrix=NULL,
verbose=FALSE )
fastmergefit( datafile, allelesfile, markerposfile,
testmarkerfile, generations=200, model='additive', verbose=FALSE )
condmergefit( h, mergedata, model='additive', covariatematrix=NULL,
verbose=FALSE )
```

Arguments

`h` an object returned by a previous call to happy()

`markerposfile`
the name of a text file containing the names and locations of the genotyped markers. Contains two names columns 'marker' and 'POSITION'

`testmarkerfile`
the name of a text file containign the names, positions and strain/allele distribution patterns for each polymorphism to be tested. Contains two columns 'marker' and 'POSITION' plus an additional named column for each of the strains listed in h$strains - *the column names and strain names must match exactly*.

`verbose` switch to control the level of ouput sent to the screen

`mergedata` an object created by a previous call to mergeprepare()

`model` determine the type of model to be fitted - either 'additive' or 'full'.

For the additive model it is assumed that the contribution to the phenotype from each chromosome is additive, ie if the founder strains at the locus being tested are s, t then the expected phenotype will be of the form $T_s + T_t$.

For the full model the expected phenotype will be of the form T_{st}.

Analysis of variance is used to test for differences between the estimated effects T_s, T_{st}.

The additive model is a submodel of the full, so for model='full' in addition a partial F-test is performed to test if the full model explains more variance than the additive.

`covariatematrix`
an optional design matrix which can be used to include additional terms in the model, such as other markers (using the matrix returned by hdesign()) and/or other covariates such as sex, age etc

`datafile` the name of a genotype datafile to be passed to happy()

`allelesfile` the name of the corresponding alleles datafile to be passed to happy()

`generations` the number of generations to be passed to happy()

Value

mergeprepare() returns a list with the following named elements:

`markerpos` the positions of the markers

`interval` an array. interval[m] contains the index of the genotyped marker interval in which the polymorhism p is located, or NULL if it is outside all genotyped intervals.

`markers`
`testmarkerdata`
details about the polymorphisms to be tested

normal-bracket50bracket-normal

mergefit() and fastmergefit() return an object, called say 'fit', suitable for plotting using mergeplot(). It contains a named element 'table' containing the log-P values as in hfit(), which can be printed using `write.table(fit$table)`.

condmergefit() returns a table with columns "position", "interval", "sdp", "logPself", "logPmax", "logPmaxPosition" .

Author(s)

Richard Mott

See Also

happy(), mergeplot()

Examples

```
## An example session:
# initialise happy
## Not run: h <- happy('Hs.data','HS.alleles')
# prepare the merge files
## Not run: prep <- mergeprepare('markers.positions','testmarkers.txt')
# run the merge fit
## Not run: fit <- mergefit( h, prep )
# alternative, and equivalent, use of fastmergefit():
## Not run:
fit <- fastmergefit( 'Hs.data','HS.alleles',
'markers.positions','testmarkers.txt' )
## End(Not run)
# plot the results
## Not run: mergeplot( fit, prep )
```

Index