

Package ‘ElstonStewart’

October 12, 2022

Type Package
Title Elston-Stewart Algorithm
Depends kinship2
Imports methods, parallel, digest
Version 1.2
Date 2022-06-06
Author Herve Perdry
Maintainer Herve Perdry <herve.perdry@universite-paris-saclay.fr>
Description Flexible implementation of Elston-Stewart algorithm.
License GPL (>= 2)
Encoding UTF-8
LazyLoad yes
LazyData yes
NeedsCompilation no
Repository CRAN
Date/Publication 2022-06-20 13:40:02 UTC

R topics documented:

ElstonStewart-package	2
conrad2	2
Elston	3
es.pedigree	4
es.stopCluster	5
fams	6
Likelihood	6
plot.es.pedigree	8

Index	10
--------------	-----------

ElstonStewart-package *Elston-Stewart package*

Description

A flexible implementation of the Elston-Stewart algorithm

Details

The Elston-Stewart algorithm allows to compute probability functions in pedigrees.

The function `es.pedigree` allows to create S3 objects for pedigrees. They can be plotted with a S3 method `plot.es.pedigree` which is simply a wrapper for the method provided in `kinship2`.

The algorithm itself is run by two functions, `Elston` and `Likelihood`. `Elston` computes a probability for a single pedigree, given a probability model (similar to the one given in `modele.di`). It relies on memoization (or dynamic programming) and allows vectorization (computing a vector of probabilities at once). `Likelihood` is similar, but runs on pedigrees lists: it computes (the sum of) the logarithms of the probabilities for each pedigree. It allows parallelization, running the computation for the various pedigrees on different nodes of a cluster. The cluster is left opened for being re-used (with memoization) on the same set of pedigrees. It can be closed with `es.stopCluster`.

Two small data sets are provided for illustration, `conrad2` and `fams`. The Elston-Stewart vignette gives commented examples.

Author(s)

Herve Perdry <herve.perdry@u-psud.fr>

conrad2

The pedigree of Conrad II, Holy Roman Emperor

Description

A pedigree with inbreeding used in examples

Usage

```
data(conrad2)
```

Format

A data frame with 22 lines with the following variables.

`id` subject id (22 is Conrad II, 2 is Louis the Stammerer)

`father` id of the subject's father

`mother` id of the subject's mother

`sex` 1=male, 2=female

Details

See vignette("Elston-Stewart") for the name of some other individuals in the pedigree.

Examples

```
data(conrad2)
```

Elston

Compute a probability function on a pedigree

Description

Compute the probability of a pedigree, for a parameter theta and a given modele

Usage

```
Elston(ped, modele, theta, mem = new.env())
```

Arguments

ped	a pedigree, created by <code>es.pedigree</code>
modele	a modele
theta	a parameter for the modele
mem	an environment used for memoization

Details

This function runs Elston-Stewart algorithm. If mem is provided, some intermediate results of previous runs stored in it can be re-used.

Value

A list with two components, `result`: the probability value, and `mem`: an object storing intermediate results for future runs

See Also

[Likelihood](#), [es.pedigree](#)

Examples

```
## cf Elston-Stewart vignette for more coments on this example
data(conrad2)
# creating an es.pedigree object
genotypes <- c( rep(list(0:2), 21), 2 )

X <- es.pedigree( id = conrad2$id, father = conrad2$father, mother = conrad2$mother,
                 sex = conrad2$sex, pheno = rep(0, 22), geno = genotypes )

# running Elston-Stewart
r <- Elston(X, modele.di, list(p = 0.98))
r$result

# using the memoization...
r <- Elston(X, modele.di, list(p = 0.99), r$mem)
r$result
```

es.pedigree

Creates a pedigree

Description

Creates a pedigree from vectors giving individuals' id, fathers' id, mothers' is, sex, phenotype and genotype

Usage

```
es.pedigree(id, father, mother, sex, pheno, geno, famid)
```

Arguments

id	a numeric vector of unique individuals id. Ids must be different from 0
father	a vector giving the id of the father of each individual (0 for founder)
mother	a vector giving the id of the mother of each individual (0 for founder)
sex	a vector giving the sex of each individual (1 = male, 2 = female)
pheno	a list giving the phenotypes of each individual
geno	a list giving the genotypes of each individual
famid	(facultative) a family id, used only to issue error messages

Details

All vectors must have the same length, including the lists pheno and geno. The list geno is a list of vectors containing the possible genotypes of each individual. The format of pheno depends on the modele which will be used to compute likelihoods with Elston or Likelihood.

Value

An object of class S3 es.pedigree

See Also

[plot.es.pedigree](#)

Examples

```
## cf Elston-Stewart vignette for more coments on this example
data(conrad2)
# creating an es.pedigree object
genotypes <- c( rep(list(0:2), 21), 2 )

X <- es.pedigree( id = conrad2$id, father = conrad2$father, mother = conrad2$mother,
                 sex = conrad2$sex, pheno = rep(0, 22), geno = genotypes )
X # displays a short information on X
```

es.stopCluster

Closing clusters

Description

This function closes clusters opened by Likelihood

Usage

```
es.stopCluster(verbose = TRUE)
```

Arguments

verbose if TRUE, the function will display a short information message when closing clusters

See Also

[Likelihood](#)

fams

The pedigree of Conrad II, Holy Roman Emperor

Description

A pedigree with imbreeding used in examples

Usage

```
data(conrad2)
```

Format

A data frame with 802 lines with the following variables.

fam family id

id subject id

father id of the subject's father

mother id of the subject's mother

sex 1=male, 2=female

genotype coded additively. Many genotypes are unknown (NA)

Examples

```
data(fams)
```

Likelihood

Compute the log-likelihood of a parameter

Description

Compute the log-likelihood of a parameter theta, given a list of pedigrees and a model, using multiple cores and memoization

Usage

```
Likelihood(ped.set, modele, theta,  
           n.cores = getOption("mc.cores", 2L),  
           optim.alloc = TRUE, sum.likelihoods = TRUE,  
           PSOCK = Sys.info()["sysname"] != "Linux")
```

Arguments

<code>ped.set</code>	a list of pedigrees, each created with <code>es.pedigree</code>
<code>modele</code>	a <code>modele</code>
<code>theta</code>	a parameter for the <code>modele</code>
<code>n.cores</code>	number of cores on which to run the computation
<code>optim.alloc</code>	cf <code>Details</code>
<code>sum.likelihoods</code>	cf <code>Value</code>
<code>PSOCK</code>	Use <code>PSOCK</code> cluster instead of fork cluster (defaults to <code>TRUE</code> on non-Linux)

Details

The parameter `theta` will be given to the functions in `modele` to compute the likelihoods.

If `n.cores > 1` a cluster is created and left open for futur use with the same parameters `ped.set` and `modele`. Open clusters can be closed with `es.stopCluster()`.

If `optim.alloc = TRUE`, the function tries to optimize the distribution of the computation (if `n.cores > 1`) between the cluster nodes. This has no effect on the first run for given parameters `ped.set` and `modele`, but will reduce running time if the algorithm is ran several times with different values of `theta` and same parameters `ped.set` and `modele`. This is typically usefull for likelihood maximization.

Value

If `sum.likelihoods = TRUE`, the function returns a single value, the sum of the log-likelihoods computed for each pedigree of `ped.set`. Else, the function returns a vector containing these log-likelihoods.

See Also

[Elston](#), [es.pedigree](#), [es.stopCluster](#)

Examples

```
data(fams)

# this data frame contains various families
# getting their famid
fam.ids <- unique(fams$fam);

# creating a list of genotypes corresponding to all individuals in fams
# NA -> 0, 1 or 2
genotypes <- lapply( fams$genotype, function(x) if(is.na(x)) 0:2 else x )

# creating a list of es.pedigree objects
X <- vector("list", length(fam.ids))
for(i in seq_along(fam.ids))
{
  w <- which(fams$fam == fam.ids[i])
```

```

X[[i]] <- es.pedigree( id = fams$id[w], father = fams$father[w],
  mother = fams$mother[w], sex = fams$sex[w], pheno = rep(0, length(w)),
  geno = genotypes[w], famid = fam.ids[i] )
}

## Not run: # computing the likelihood for a single value p
Likelihood(X, modele.di, theta = list( p=0.5), n.cores=1 )

# computing the likelihood for a vector p (Elston-Stewart is ran only once!)
p <- seq(0,1,length=501)
L <- Likelihood(X, modele.di, theta = list( p=p ), n.cores=1 )
plot( p, exp(L), type="l" )

# running an optimization algorithm
# Elston-Stewart is ran several times
# here we run the algorithm with 2 cores
L <- function(p) -Likelihood(X, modele.di, theta = list( p=p ), n.cores=2 )
optimize(L , c(0.35,0.45) )
## End(Not run)

```

`plot.es.pedigree` *Plot pedigrees*

Description

Plot objects created with `es.pedigree`

Usage

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

Arguments

<code>x</code>	an object created with <code>es.pedigree</code>
<code>...</code>	additionnal arguments including possibly a vector of phenotypes <code>pheno</code> , and other arguments to be given to <code>plot.pedigree</code>

Details

This function relies on the function `plot.pedigree` provided by the package `kinship2`. The phenotype is build assuming that `x$pheno` is a list of vectors, the first element of which gives the status. Alternatively, a vector `pheno` of appropriate length can be given as an additionnal argument.

See Also

[es.pedigree](#), [plot.pedigree](#)

Examples

```
## cf Elston-Stewart vignette for more coments on this example
data(conrad2)
# creating an es.pedigree object
genotypes <- c( rep(list(0:2), 21), 2 )

X <- es.pedigree( id = conrad2$id, father = conrad2$father, mother = conrad2$mother,
                 sex = conrad2$sex, pheno = rep(0, 22), geno = genotypes )

plot(X)
```

Index

* datasets

conrad2, [2](#)

fams, [6](#)

conrad2, [2](#), [2](#)

Elston, [2](#), [3](#), [7](#)

ElstonStewart (ElstonStewart-package), [2](#)

ElstonStewart-package, [2](#)

es.pedigree, [2](#), [3](#), [4](#), [7](#), [8](#)

es.stopCluster, [2](#), [5](#), [7](#)

fams, [2](#), [6](#)

Likelihood, [2](#), [3](#), [5](#), [6](#)

modele.di (ElstonStewart-package), [2](#)

plot.es.pedigree, [2](#), [5](#), [8](#)

plot.pedigree, [8](#)

print.es.pedigree (es.pedigree), [4](#)