Package 'epinetr'

July 22, 2025

Title Epistatic Network Modelling with Forward-Time Simulation

Version 0.96

Description Allows for forward-in-time simulation of epistatic networks with associated phenotypic output.

URL https://github.com/diondetterer/epinetr

BugReports https://github.com/diondetterer/epinetr/issues

Depends R (>= 2.10)

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports GA (>= 3.2), ggplot2 (>= 3.1.1), igraph (>= 1.2.4), methods (>= 2.10), Rcpp (>= 0.12.18), RcppAlgos (>= 2.4.1), reshape2 (>= 1.4.3), vcfR (>= 1.8.0)

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

LinkingTo Rcpp

NeedsCompilation yes

Author Dion Detterer [aut, cre],
Paul Kwan [aut],
Cedric Gondro [aut]

Maintainer Dion Detterer <ddettere@myune.edu.au>

Repository CRAN

Date/Publication 2022-03-10 10:30:02 UTC

Contents

`addEffects` *Attach additive effects to population.*

Description

Attach additive effects to a `Population` object.

Usage

```
addEffects(pop, effects = NULL, distrib = rnorm)
```

Arguments

`pop`	an object of class `Population`.
`effects`	an optional vector of additive effect coefficients.
`distrib`	an optional random number generator function for a distribution, defaulting to `rnorm`.

Details

`addEffects()` is a function for attaching additive effects to a given population, ensuring that the initial additive variance is as given in the population parameters.

If additive effect coefficients are directly supplied via the `effects` vector, these may be scaled in order to comply with the initial additive variance.

Value

A copy of the supplied `Population` object is returned, with additive effects attached.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

`Population`, `attachEpiNet`

Examples

```
# Create population
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.9, narrowh2 = 0.6, traitVar = 40
)

# Attach additive effects using a normal distribution
pop <- addEffects(pop)

# Attach additive effects using a uniform distribution
pop2 <- addEffects(pop, distrib = runif)

# Attach additive effects using a vector of coefficients
effects <- c(
  1.2, 1.5, -0.3, -1.4, 0.8,
  2.4, 0.2, -0.8, -0.4, 0.8,
  -0.2, -1.4, 1.4, 0.2, -0.9,
  0.4, -0.8, 0.0, -1.1, -1.3
)
pop3 <- addEffects(pop, effects = effects)

# Print first population
pop

# Print second population
pop2

# Print third population
pop3
```

`attachEpiNet` *Attach epistatic network to population.*

Description

Constructs and attaches a new epistatic network between QTLs to a given `Population` object.

Usage

```
attachEpiNet(pop, scaleFree = FALSE, k = 2, m = 1, additive = 0, incmat = NULL)
```

Arguments

`pop`	the `Population` object to which the epistatic network will be attached
`scaleFree`	an optional logical value indicating whether to construct a network using the Barabasi-Albert model for generating scale-free networks
`k`	an optional vector listing orders of interaction to include
`m`	an optional integer indicating the minimum number of interactions for each QTL; see details below
`additive`	an optional integer indicating the number of QTLs which will remain purely additive
`incmat`	an optional incidence matrix specifying a user-defined network

Details

`attachEpiNet()` can be used to construct a new epistatic network based either on stochastic processes or by adapting a user-supplied incidence matrix.

If `scaleFree` is `FALSE`, a network is constructed at random; if it is `TRUE`, however, a scale-free network is constructed between QTLs using the Barabasi-Albert model. (The random network is constructed using the same algorithm but without preferential attachment.)

A minimal initial network of randomly selected QTLs is first constructed, before then growing the network using randomly selected QTLs, preferentially attaching each QTL to at least `m` other QTLs (in the scale-free case). For increasing orders of interaction, degrees from lower orders of interaction are used when determining preferential attachment.

If a user-supplied incidence matrix is given via the `incmat` argument, the `scaleFree`, `k` and `m` arguments are ignored, and the network structure is instead derived from the incidence matrix. The matrix should be such that the rows represent QTLs and the columns represent interactions between QTLs: a `1` at both `incmat[i1,j]` and `incmat[i2,j]` means there is an interaction between the QTLs `i1` and `i2`.

The contributions that interactions make towards the phenotypic value are generated randomly when this function is called, based on a normal distribution. By default, each order of interaction has the same variance; however, using the `varfun` argument, a function can be supplied to alter the variance per order of interaction.

Internally, each QTL is assigned a randomly generated value per interaction per genotype (i.e. the heterozygous genotype and the two homozygous genotypes), and these values are summed according to the value of the genotype.

Value

A copy of the supplied `Population` is returned, with the new epistatic network attached.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

References

Barabasi AL, Albert R, 'Emergence of scaling in random networks,' *Science* 286(5439): 509-12, 15 October 1999.

See Also

`Population`, `getEpiNet`, `plot.EpiNet`, `addEffects`

Examples

```
# Generate a population and attach additive effects
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.9, narrowh2 = 0.6, traitVar = 40
)
pop <- addEffects(pop)

# Attach a random epistatic network with two- to four-way
# interactions between QTLs
popRnd <- attachEpiNet(pop, k = 2:4)

# Plot random network
plot(getEpiNet(popRnd))

# Attach a scale-free epistatic network with two-way interactions
# between QTLs and a minimum of three interactions per QTL
popSF <- attachEpiNet(pop, scaleFree = TRUE, m = 3)

# Plot scale-free network
plot(getEpiNet(popSF))

# Attach user-defined epistatic network
popUser <- attachEpiNet(pop, incmat = rincmat100snp)

# Plot user-defined network
plot(getEpiNet(popUser))

# Attach a random epistatic network with two- to ten-way
# interactions between QTLs and decaying variance
popDecay <- attachEpiNet(pop, k = 2:10)
```

epinetr *epinetr*

Description

epinetr is a package intended to aid in the investigation of the contribution of epistatic networks to complex traits,

Details

This package provides a range of functions for running forward-time simulation using epistatic networks, including visualisation tools.

For a complete list of functions, use library(help = "epinetr").

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

geno100snp *Example genotype matrix.*

Description

An example genotype matrix for 100 SNPs across 500 individuals.

Usage

```
geno100snp
```

Format

geno100snp is a matrix with 500 rows and 200 columns, used in examples when constructing a population from genotypes using 500 individuals and 100 SNPs.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

Population, map100snp

getAddCoefs *Get additive coefficients.*

Description

Retrieve additive coefficients from population.

Usage

```
getAddCoefs(pop)
```

Arguments

pop	a Population object with additive effects attached

Details

getAddCoefs retrieves the additive coefficients currently in use in a Population object, assuming additive effects have been attached.

Value

getAddCoefs returns the additive coefficients currently in use by the population.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

addEffects

Examples

```
# Construct a population with additive effects
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.6, narrowh2 = 0.6, traitVar = 40
)
pop <- addEffects(pop)

# Get additive coefficients
additive <- getAddCoefs(pop)
```

`getAddOffset` *Retrieve additive offset.*

Description

Retrieve offset used for calculating additive component.

Usage

```
getAddOffset(pop)
```

Arguments

pop	A valid `Population` object with additive effects attached

Details

In order for the initial population to have an additive component with a mean of 0 for its phenotype, an offset is added, and it remains fixed across generations. This function retrieves that offset.

Value

The additive offset is returned.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

`getAddCoefs`, `addEffects`

Examples

```
# Construct a new population with additive effects
pop <- Population(
  popSize = 20, map = map100snp, QTL = 20,
  broadH2 = 0.4, narrowh2 = 0.4, traitVar = 40,
  alleleFrequencies = runif(100, 0.05, 0.5)
)
pop <- addEffects(pop)

# Find the additive contribution to the individuals' phenotypes
hap <- getHaplo(pop)
hap <- (hap[[1]] + hap[[2]])[, getQTL(pop)$Index]
(hap %*% getAddCoefs(pop))[, 1] + getAddOffset(pop)

# Compare with additive component from getComponents()
getComponents(pop)$Additive
```

`getAlleleFreqRun` *Get allele frequencies.*

Description

Get allele frequencies across a simulation run.

Usage

```
getAlleleFreqRun(pop)
```

Arguments

`pop`	a `Population` object as returned by a previous simulation run.

Details

Retrieves the allele frequencies in each generation across a simulation run.

Value

Returns a matrix of allele frequencies, one generation per row.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

`runSim`

Examples

```
# Construct a population with additive and epistatic effects
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.9, narrowh2 = 0.6, traitVar = 40
)
pop <- addEffects(pop)
pop <- attachEpiNet(pop)

# Run the simulator
pop2 <- runSim(pop, generations = 150)

af <- getAlleleFreqRun(pop2)
```

getComponents *Get phenotypic components.*

Description

Get pedigree and phenotypic data from current population.

Usage

```
getComponents(pop)
```

Arguments

pop	a Population object

Details

Retrieves the pedigree and phenotypic data components from all individuals in the current population.

Value

Returns a data.frame giving the individual's ID, the ID of the sire, the ID of the dam, the additive, epistatic and environmental components of the phenotype and the overall phenotypic value.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

Population, addEffects, attachEpiNet

Examples

```
# Construct a population with additive and epistatic effects
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.9, narrowh2 = 0.6, traitVar = 40
)
pop <- addEffects(pop)
pop <- attachEpiNet(pop)

# Retrieve phenotypic components from population
components <- getComponents(pop)
```

getEpiNet *Epistatic network retrieval.*

Description

Get an epistatic network from a Population object.

Usage

```
getEpiNet(pop)
```

Arguments

pop	An object of class 'Population' which has an EpiNet object attached.

Details

getEpiNet() is merely a function for retrieving an epistatic network object. The common purpose is to plot the network.

Value

An object of class 'EpiNet' is returned.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

Population, plot.EpiNet, attachEpiNet

Examples

```
# Create population and attach an epistatic network
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.9, narrowh2 = 0, traitVar = 40
)
pop <- attachEpiNet(pop)

# Plot epistatic network
epiNet <- getEpiNet(pop)
plot(epiNet)
```

`getEpiOffset` *Retrieve epistatic offset.*

Description

Retrieve offset used for calculating epistatic component.

Usage

```
getEpiOffset(pop)
```

Arguments

pop	A valid `Population` object with epistatic effects attached

Details

In order for the initial population to have an epistatic component with a mean of 0 for its phenotype, an offset is added, and it remains fixed across generations. This function retrieves that offset.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

`getEpistasis`

Examples

```
# Construct a new population with epistatic effects
pop <- Population(
  popSize = 20, map = map100snp, QTL = 20,
  broadH2 = 0.4, narrowh2 = 0, traitVar = 40,
  alleleFrequencies = runif(100, 0.05, 0.5)
)
pop <- attachEpiNet(pop)

# Find the epistatic contribution to the individuals' phenotypes
rowSums(getEpistasis(pop)) + getEpiOffset(pop)

# Compare with epistatic component from getComponents()
getComponents(pop)$Epistatic
```

getEpistasis *Calculate epistatic interactions.*

Description

Calculate epistatic interactions for members of the population.

Usage

```
getEpistasis(pop, scale = TRUE, geno = NULL)
```

Arguments

pop	a valid Population object with epistatic effects attached
scale	a boolean value indicating whether to scale the values to bring them in line with the desired initial variance
geno	by default, the function uses the current genotypes in the population; alternatively, geno is a user-supplied set of genotypes (limited to QTLs) on which to base the calculation

Details

This function calculates the values of each epistatic interaction per individual, returning a matrix whose rows are the individuals in the population and whose columns are the epistatic interactions. The sums of these rows represent the total epistatic contribution to the phenotype, once the epistatic offset is added.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

getEpiOffset

Examples

```
# Construct a new population with epistatic effects
pop <- Population(
  popSize = 20, map = map100snp, QTL = 20,
  broadH2 = 0.4, narrowh2 = 0, traitVar = 40,
  alleleFrequencies = runif(100, 0.05, 0.5)
)
pop <- attachEpiNet(pop)

# Find the epistatic contribution to the individuals' phenotypes
rowSums(getEpistasis(pop)) + getEpiOffset(pop)
```

```
# Compare with epistatic component from getComponents()
getComponents(pop)$Epistatic
```

---

getGeno *Get population unphased genotypes.*

Description

Retrieves the current unphased genotypes in the population.

Usage

```
getGeno(pop)
```

Arguments

pop	a valid Population object.

Details

getGeno retrieves the current unphased genotypes in the population, returning a single matrix with one individual per row and one SNP per column.

Value

Returns an unphased genotypes matrix.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

Population, getPhased, getHaplo

Examples

```
# Construct a population
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.9, narrowh2 = 0.6, traitVar = 40
)

# Retrieve genotypes
geno <- getGeno(pop)
```

getHaplo *Retrieve haplotypes.*

Description

Retrieve haplotypes from the population.

Usage

```
getHaplo(pop)
```

Arguments

pop	a valid Population object

Details

Retrieves the haplotypes from both the population which, when added together, form the genotypes.

Value

A list is returned with two elements, corresponding to the two haplotype matrices.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

getAddCoefs, getAddOffset, getPhased, getGeno

Examples

```
# Construct a new population with additive effects
pop <- Population(
  popSize = 20, map = map100snp, QTL = 20,
  broadH2 = 0.4, narrowh2 = 0.4, traitVar = 40,
  alleleFrequencies = runif(100, 0.05, 0.5)
)
pop <- addEffects(pop)

# Find the additive contribution to the individuals' phenotypes
hap <- getHaplo(pop)
hap <- (hap[[1]] + hap[[2]])[, getQTL(pop)$Index]
(hap %*% getAddCoefs(pop))[, 1] + getAddOffset(pop)
```

getIncMatrix *Incidence matrix retrieval.*

Description

Get an incidence matrix from a Population.

Usage

```
getIncMatrix(pop)
```

Arguments

pop	An object of class 'Population' which has an EpiNet object attached.

Details

getIncMatrix() retrieves the incidence matrix used in epistatic interactions within the given Population object. This is most useful for copying the network structure to a new Population object.

Value

An incidence matrix representing the epistatic network within the given Population object.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

attachEpiNet

Examples

```
# Create population
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.9, narrowh2 = 0, traitVar = 40
)

# Attach random epistatic network and retrieve incidence matrix
pop <- attachEpiNet(pop)
inc <- getIncMatrix(pop)

# Create second population
pop2 <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
```

```
  broadH2 = 0.8, narrowh2 = 0.6, traitVar = 40
)

# Attach epistatic network to second population
# using incidence matrix from first
pop2 <- attachEpiNet(pop2, incmat = inc)
```

`getInteraction` *Retrieve interaction values.*

Description

Retrieve values for an interaction within an epistatic network.

Usage

```
getInteraction(pop, n)
```

Arguments

`pop`	a valid population object
`n`	the interaction to return

Details

This function returns a k-dimensional array for a particular interaction, where k is the order of interaction and the array holds 3^k entries . This means that a 5-way interaction, for example, will return a 5-dimensional array consisting of $3^5 = 243$ entries.

Within each dimension, the three indices (1, 2 and 3) correspond to the homozygous genotype coded 0/0, the heterozygous genotype and the homozygous genotype coded 1/1, respectively. Each entry is drawn from a normal distribution. (Any offset needs to be applied manually using `getEpiOffset`.)

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

`attachEpiNet`, `getEpiOffset`

Examples

```
# Construct a new population
pop <- Population(
  popSize = 150, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100), broadH2 = 0.7,
  narrowh2 = 0.45, traitVar = 40
)
```

```
# Attach additive effects
pop <- addEffects(pop)

# Attach a network of epistatic effects
pop <- attachEpiNet(pop)

# Retrieve the possible values for the first two-way interaction
getInteraction(pop, 1)

# Retrieve the value for the case where, in the fourth two-way
# interaction, the first QTL in the interaction is heterozygous
# and the second QTL in the interaction is the homozygous
# reference genotype.
getInteraction(pop, 4)[2, 1]

# Retrieve the value for the case where, in the second two-way
# interaction, the first QTL in the interaction is the homozygous
# reference genotype and the second QTL in the interaction is the
# homozygous alternative genotype.
getInteraction(pop, 2)[1, 3]
```

`getPedigree` *Get population pedigree.*

Description

Retrieve the pedigree of a `Population` object.

Usage

```
getPedigree(pop)
```

Arguments

`pop`	a valid object of class `Population`.

Details

`getPedigree()` can be used to retrieve the pedigree of a population, including the phenotypic components of all individuals in the pedigree.

Value

A `data.frame` containing vectors for each individual's ID, its sire and dam IDs, and its phenotypic components, across the pedigree.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

runSim

Examples

```
# Construct a population with additive and epistatic effects
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.9, narrowh2 = 0.6, traitVar = 40
)
pop <- addEffects(pop)
pop <- attachEpiNet(pop)

# Run the simulator
pop2 <- runSim(pop, generations = 150)

# Retrieve the population pedigree from the simulation
ped <- getPedigree(pop2)

# Re-run the simulation using the same pedigree
pop3 <- runSim(pop, ped)
```

---

getPhased *Get population phased genotypes.*

Description

Retrieves the current phased genotypes in the population.

Usage

```
getPhased(pop)
```

Arguments

pop	a valid Population object.

Details

getPhased retrieves the current phased genotypes in the population, returning a single matrix with one individual per row and two columns per SNP.

Value

Returns a phased genotypes matrix.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

Population, getHaplo, getGeno

Examples

```
# Construct a population
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.9, narrowh2 = 0.6, traitVar = 40
)

# Retrieve genotypes
geno <- getPhased(pop)
```

---

getQTL *QTL retrieval.*

Description

Retrieve the QTLs being used for this population.

Usage

```
getQTL(pop)
```

Arguments

pop	An object of class 'Population'.

Details

getQTL retrieves the IDs and indices of the SNPs being used as QTLs for a given Population object.

Value

A data.frame containing the IDs and indices of all QTLs is returned.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

Population

Examples

```
# Create population
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.9, narrowh2 = 0.6, traitVar = 40
)

# Get the SNP IDs of the QTLs
getQTL(pop)
```

---

getSubPop *Get subpopulation*

Description

Retrieve a subset of a Population without recalculating effects

Usage

```
getSubPop(pop, ID)
```

Arguments

pop	a valid object of class Population.
ID	a vector giving the IDs of individuals to include in the subset

Details

getSubPop() returns a new Population object using the individuals with IDs specified by the vector ID.

Any additive and epistatic effects will be copied as-is to the new Population object, with heritability parameters recalculated.

Any IDs given but not present will be discarded.

Value

A new Population object containing the specified individuals is returned.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

Population, getComponents

Examples

```
# Construct a population with additive and epistatic effects
pop <- Population(
  popSize = 2000, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100)
)
pop <- addEffects(pop)
pop <- attachEpiNet(pop)

# Run the simulator
pop2 <- runSim(pop, generations = 10)

# Create a new subpopulation of 500 individuals
ID <- getComponents(pop2)$ID
ID <- sample(ID, 500)
pop3 <- getSubPop(pop2, ID)
```

---

loadGeno *Load epinetr genotype file.*

Description

Load genotypes from a previous epinetr session.

Usage

```
loadGeno(filename)
```

Arguments

filename	the filename for the epinetr genotypes file.

Details

When outputting all genotypes during an epinetr simulation run, the genotypes will be written to a serialised format unique to epinetr. The loadGeno function will load these genotypes into memory as a single matrix object.

Value

a numeric matrix holding the genotypes

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

Population

Examples

```
# Load genotype file
filename <- system.file("extdata", "geno.epi", package = "epinetr")
geno <- loadGeno(filename)

# Use genotypes as basis for new population
pop <- Population(
  map = map100snp, QTL = 20, genotypes = geno,
  broadH2 = 0.8, narrowh2 = 0.6, traitVar = 40
)
```

---

map100snp *Example map.*

Description

An example map data.frame for 100 SNPs across 22 chromosomes.

Usage

```
map100snp
```

Format

map100snp is a data.frame with 100 rows and 3 variables:

V1 SNP ID

V2 chromosome ID

v3 position on chromosome, in base pairs

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

Population, geno100snp

`plot.EpiNet` *Plot epistatic network.*

Description

Plot an epistatic network between a set of QTLs.

Usage

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

Arguments

`x`	an object of class `'EpiNet'`.
`...`	additional parameters (ignored)

Details

An object of class `EpiNet` is typically first retrieved from a `Population` object (using `getEpiNet`) before being plotted using `plot.EpiNet()`.

Value

A plot of the epistatic network is displayed.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

`Population`, `attachEpiNet`, `getEpiNet`

Examples

```
# Build a population with an epistatic network attached
pop <- Population(
  popSize = 100, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100), broadH2 = 0.9,
  narrowh2 = 0, traitVar = 40
)
pop <- attachEpiNet(pop)

# Retrieve and plot the epistatic network
epinet <- getEpiNet(pop)
plot(epinet)
```

`plot.Population` *Plot phenotypic value for a population.*

Description

Plot the phenotypic value for a population over the course of a prior simulation run.

Usage

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

Arguments

`x`	an object of class `'Population'` which has been run in the simulator
`...`	additional parameters (ignored)

Details

The plot is a line graph depicting the mean, minimum and maximum phenotypic value in the population across generations. This method can only be used if the population has been run via the simulator.

Value

A plot of the population's simulation run is displayed.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

`Population`, `runSim`, `addEffects`, `attachEpiNet`

Examples

```
# Build a population
pop <- Population(
  popSize = 100, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100), broadH2 = 0.9,
  narrowh2 = 0.5, traitVar = 40
)
pop <- addEffects(pop)
pop <- attachEpiNet(pop)

# Run population in simulation
pop <- runSim(pop)
```

```
# Plot population's run
plot(pop)
```

`Population` *Population constructor.*

Description

The constructor for the `Population` object.

Usage

```
Population(
  pop = NULL,
  popSize = NULL,
  vcf = NULL,
  map = NULL,
  QTL = NULL,
  genotypes = NULL,
  literal = TRUE,
  alleleFrequencies = NULL,
  broadH2 = NULL,
  narrowh2 = NULL,
  traitVar = NULL,
  h2est = NULL
)
```

Arguments

`pop`	an optional `Population` object to use for default values
`popSize`	an optional number of individuals in the new `Population` to be created
`vcf`	an optional VCF file which will provide the map and genotypes
`map`	an optional `data.frame` specifying the name, chromosome and position (in base-pairs) of each SNP
`QTL`	an optional argument giving either a single number which specifies the number of SNPs to randomly select as QTLs, or a vector of SNP IDs (from `map`) to select as QTLs
`genotypes`	an optional matrix of genotypes to use for the population; see below for details
`literal`	an optional `logical` value specifying whether to use the genotypes directly or to generate new genotypes based on the allele frequencies of the given genotypes
`alleleFrequencies`	an optional vector of allele frequencies for generating genotypes
`broadH2`	initial broad-sense heritability within the new `Population`
`narrowh2`	initial narrow-sense heritability within the new `Population`
`traitVar`	initial phenotypic variance within the new `Population`
`h2est`	suggested heritability estimate for the new `Population`

Details

Population() creates a new Population object based on arguments which optionally modify a previously defined Population object. If no Population object is given, the new Population is created using only the arguments given.

The arguments vcf, map, genotypes, literal and alleleFrequencies all work together in a specific way.

If a VCF file is supplied via the vcf argument, the map, genotypes and alleleFrequencies arguments are not needed, since a map and set of genotypes are given within the VCF file. If the number of individuals' genotypes given by the VCF file does not match the number of individuals specified by the popSize argument, the supplied genotypes within the VCF file are used to suggest allele frequencies only: this behaviour can also be forced by setting the literal argument to FALSE.

The genotypes argument supplies genotypes directly. In this case, the user should supply a phased, individual-major genotypes matrix: one individual per row and two columns per single nucleotide polymorphism (SNP). Odd columns are assumed to be the haplotypes inherited from the sires, while even columns are assumed to be the haplotypes inherited from the dams. As with genotypes supplied via a VCF file, if the number of individuals' genotypes given by the matrix does not match the number of individuals specified by the popSize argument, the supplied genotypes are used to suggest allele frequencies only; again, this behaviour can also be forced by setting the literal argument to FALSE.

When supplying genotypes either directly or via a VCF file, all SNPs should be biallelic and phased, with no missing values. Genotypes supplied directly should have variants coded as either 0 or 1.

Any map (either supplied directly or via a VCF file) will be sorted, such that all SNPs along the first chromosome listed will appear at the start of the map, sorted in terms of base-pair distance; the second chromosome to appear will then be treated similarly, and so on. SNPs will be referenced within the population in this order.

The alleleFrequencies argument is used when genotypes are not given directly. In this case, the literal argument has no meaning.

An example map data.frame has been included in the epinetr package as map100snp. Note that all chromosomes must be autosomal, whether given via the map parameter or via a VCF file.

The Population object will estimate breeding values once all necessary effects are given. An optional heritability estimate can be supplied using the h2est parameter.

When supplying an existing Population object, any additive effects and epistatic network will be carried over from the previous population unless new QTLs are supplied.

Note that if broadH2 is equal to narrowh2, no epistatic effects will be present; if narrowh2 is 0, no additive effects will be present; if broadH2 is 1, no environmental effects will be present.

The h2est argument bypasses REML-based heritability estimates by supplying a user-defined heritability estimate for use in calculating estimated SNP effects.

Value

The constructor creates a new object of class 'Population'.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

addEffects, attachEpiNet, print.Population

Examples

```
# Construct a new population of size 500, with random allele
# frequencies and 20 QTLs chosen at random, broad-sense
# heritability set to 0.9, narrow-sense heritability set to 0.75
# and overall trait variance set to 40.

pop <- Population(
  popSize = 500, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100), broadH2 = 0.9,
  narrowh2 = 0.75, traitVar = 40
)

# Construct a new population of size 500 using directly supplied
# genotypes and 20 QTLs chosen at random, broad-sense heritability
# set to 0.7, narrow-sense heritability set to 0.3 and overall
# trait variance set to 10.

pop2 <- Population(
  map = map100snp, genotypes = geno100snp,
  literal = TRUE, QTL = 20,
  broadH2 = 0.7, narrowh2 = 0.3, traitVar = 10
)

# Modify the previous population to have narrow-sense heritabilty
# set to 0.45 and overall trait variance set to 20.

pop2 <- Population(pop2, narrowh2 = 0.45, traitVar = 20)
```

---

print.Population *Print function for population.*

Description

Print a summary of the population object.

Usage

```
## S3 method for class 'Population'
print(x, ...)
```

Arguments

x	a valid Population object
...	additional parameters (ignored)

Details

This is an S3 method for printing a summary of a Population object. Displayed are the initial parameters for the population (i.e. population size, phenotypic variance, broad-sense heritability, narrow-sense heritability and the SNPs used as QTLs), followed by any current additive and epistatic variance in the population.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

Population, addEffects, attachEpiNet, runSim

Examples

```
# Build a population
pop <- Population(
  popSize = 10, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100), broadH2 = 0.9,
  narrowh2 = 0.5, traitVar = 40
)
pop <- addEffects(pop)
pop <- attachEpiNet(pop)

# Print the initial population
pop

# Run population in simulation
pop2 <- runSim(pop, generations = 50)

# Print the population following the simulation
pop2
```

---

rincmat100snp *Example incidence matrix.*

Description

An example incidence matrix for 19 pairwise interactions across 20 QTLs.

Usage

```
rincmat100snp
```

Format

rincmat100snp is a matrix with 20 rows and 19 columns, used in examples when constructing an epistatic network using a user-supplied incidence matrix.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

attachEpiNet

---

runSim *Run simulation on population.*

Description

Run a forward-time simulation on a Population object.

Usage

```
runSim(
  pop,
  pedigree = NULL,
  generations = 2,
  selection = "random",
  fitness = "phenotypic",
  burnIn = 0,
  truncSire = 1,
  truncDam = 1,
  roundsSire = 1,
  roundsDam = 1,
  litterDist = c(0, 0, 1),
  breedSire = 10,
  mutation = 10^-9,
  recombination = NULL,
  allGenoFileName = NULL
)
```

Arguments

pop	a valid Population object with all necessary additive and epistatic effects attached
pedigree	an optional data.frame giving the pedigree to follow for selection
generations	an optional integer giving the number of generations to iterate through in the simulation
selection	an optional string specifying random (the default) or linear ranking selection
fitness	an optional string specifying whether selection takes place based on phenotypic value (the default), true genetic value or estimated breeding value

burnIn	an optional integer giving the initial number of generations in which to use random selection without truncation, even when linear ranking selection or truncation is otherwise employed
truncSire	an optional value giving the proportion of the males in the population with the highest phenotypic value to select within
truncDam	an optional value giving the proportion of the females in the population with the highest phenotypic value to select within
roundsSire	an optional integer giving the maximum number of generations for a male to survive; see details below
roundsDam	an optional integer giving the maximum number of generations for a female to survive; see details below
litterDist	an optional vector giving the probability mass function for the litter sizes, starting with a litter of 0
breedSire	an optional integer indicating the maximum number of times that a sire can breed per generation
mutation	an optional value giving the rate of mutation per SNP
recombination	an optional vector giving the probabilities for recombination events between each SNP
allGenoFileName	a string giving a file name, indicating that all genotypes will be outputted to the file during the run

Details

runSim is the forward-time simulation engine of the epinetr package. A Population object with necessary additive and epistatic effects must be supplied; all other arguments are optional, though either pedigree or generations must be supplied.

pedigree should be a data.frame where the first three columns are the ID, sire ID and dam ID respectively. Sire and dam IDs of 0 indicate that the individual is in the first generation; each ID in the first generation should match an ID in the given Population object. The pedigree will be sorted into generations before running, where a 'generation' in this case is defined as the set of individuals whose parents are both from a previous generation. If a pedigree is supplied, all further arguments (which pertain to selection) will be ignored.

generations is the number of generations through which the simulation will iterate. The supplied population represents the first generation: the default value of 2 for this argument thus means that the simulator will simply return the next generation.

selection is a string specifying 'ranking' for linear ranking selection; any other string is interpreted as 'random' for random selection.

Linear ranking selection mimics natural selection: if the individuals in a population of size n are each given a rank r based on descending order of phenotypic value (i.e. the individual with the highest phenotypic value is given the rank $r_1 = 1$r_1 = 1 while the individual with the lowest phenotypic value is given the rank $r_n = n$r_n = n), the probability of an individual i being selected for mating is given by:

$$P(i \text{ is selected}) = \frac{2(n - r_i + 1)}{n(n+1)}$$

P(i is selected) = 2(n - r_i + 1) / n(n + 1)

Selection occurs by the population first being split into male and female sub-populations. Next, if the round is outside any initial burn-in period, each sub-population is truncated to a proportion of its original size per the values of `truncSire` and `truncDam`, respectively.

When linear ranking selection is used, females are exhaustively sampled, without replacement, for each mating pair using their linear ranking probabilities, as given above; males are sampled for each mating pair using their linear ranking probabilities but with replacement, where they are each only replaced a maximum number of times as specified by `breedSire`. Random selection occurs in the same manner, but all probabilities are uniform. During any initial `burnIn` period, random selection is enforced.

`fitness` specifies how fitness is determined for the purposes of selection: 'phenotypic' (the default) selects based on the phenotype while 'TGV' selects by ignoring environmental noise; 'EBV' selects based on estimated breeding values using estimated SNP effects.

Each mating pair produces a number of full-sibling offspring by sampling once from the litter-size probability mass function given by `litterDist` (with the default guaranteeing two full-sibling offspring per mating pair). The PMF is specified via a vector giving the probabilities for each litter size, starting with a litter size of 0. For example, `c(0.2, 0.0, 0.1, 0.4, 0.3)` gives a 20% chance of a litter size of 0, a 10% chance of litter size of 2, a 40% chance of a litter size of 3, a 30% chance of a litter size of 4 and a 0% chance of a litter size of 1 or greater than 4.

Half-siblings occur when sires can mate more than once per round (as given by `breedSire`) or when sires or dams survive beyond one round (as given by `roundsSire` and `roundsDam`, respectively). It is important to note that `roundsSire` and `roundsDam`, which specify the maximum number of generations for males and females to survive, respectively, will be ignored in the case where an insufficient number of offspring are produced to replace the individuals who have nonetheless survived the maximum number of rounds: in this case, younger individuals will be preserved in order to meet the population size.

`recombination` is a vector of recombination rates between SNPs. The length of this vector should be equal to the number of SNPs in the population's `map` minus the number of chromosomes. The order of the chromosomes is as per the `map`.

`allGenoFileName` is the name of a file in which the phased genotype for every individual will be stored. The output is serialised and can be read using `loadGeno`. If the `allGenoFileName` argument is not given, no genotypes will be written to file.

Value

A new `Population` object is returned.

Author(s)

Dion Detterer, Paul Kwan, Cedric Gondro

See Also

`Population`, `addEffects`, `attachEpiNet`, `print.Population`, `plot.Population`, `loadGeno`

Examples

```
# Create population
pop <- Population(
  popSize = 200, map = map100snp, QTL = 20,
  alleleFrequencies = runif(100),
  broadH2 = 0.9, narrowh2 = 0.6, traitVar = 40
)

# Attach additive effects using a normal distribution
pop <- addEffects(pop)

# Attach epistatic effects
pop <- attachEpiNet(pop)

# Run simulation for 150 generations
pop <- runSim(pop, generations = 150)

# Display results
pop

# Plot results
plot(pop)
```

Index