Package ‘isqg’

June 2, 2020

Type Package

Title In Silico Quantitative Genetics

Version 1.3

Date 2020-06-01

Author Fernando H. Toledo [aut, cre] (<https://orcid.org/0000-0003-0158-643X>)
International Maize and Wheat Improvement Center [cph]

Maintainer Fernando H. Toledo <f.toledo@cgiar.org>

Description Accomplish high performance simulations in quantitative genetics.
The molecular genetic components are represented by R6/C++ classes and methods.
The core computational algorithm is implemented using bitsets according to
<doi:10.1534/g3.119.400373>. A mix between low and high level interfaces
provides great flexibility and allows user defined extensions and a wide range of
applications.

License GPL-2 | file LICENSE

Encoding UTF-8

NeedsCompilation yes

SystemRequirements C++14

Depends R (>= 4.0.0)

Imports Rcpp (>= 1.0.4), R6, Rdpack

LinkingTo Rcpp, BH

Collate 'ISQG.R' 'Mating.R' 'R6Classes.R' 'Functions.R' 'Trait.R'
'RcppExports.R' 'Hooks.R'

LazyData true

RdMacros Rdpack

RoxygenNote 7.1.0

Repository CRAN

Date/Publication 2020-06-02 07:40:02 UTC

R topics documented:

`isqg-package` *isqg: A package to perform* **in silico** *quantitative genetics*

Description

`isqg` provides R6/C++ classes for in silico quantitative genetics. Mimic the meiosis recombination. Allows user defined extensions which provides great flexibility. Details of the implementation are described in Toledo et al. (2019).

Author(s)

Fernando H. Toledo <`f.toledo@cgiar.org`>

References

Toledo FH, Pérez-Rodríguez P, Crossa J, Burgueño J (2019). "isqg: A Binary Framework for in Silico Quantitative Genetics"." *G3: Genes, Genomes, Genetics*, **9**(8), 2425–2428. `https://doi.org/10.1534/g3.119.400373`.

`fitness` *Simulated Trait for Individuals According to Basic Models*

Description

Constructor of instances of the Trait class given the focal specie and the parameters to define infinitesimal or quantitative fitness.

Usage

```
set_infty(specie, m = 0, a = 1, d = 0, genes = NULL)

set_quant(specie, m, data)
```

Arguments

`specie`	an instance of the R6 class Specie with the genome's parameters.
`m, a, d`	a length-one numeric vector with respectively the mean, the additive and the dominant effects.
`genes`	a character vector with the putative genes.
`data`	a data frame with the genes (snp) and their additive and dominant effects

Details

Infinitesimal traits need the mean, the additive and the dominant effect and optionally the vector of the putative genes. Quantitative traits are defined given the mean and a data frame with the putative genes and their additive and dominant effects.

Value

Objects of R6 class with methods to mimic in silico Traits.

Examples

```
data(ToyMap)
spc <- set_specie(ToyMap)
AA <- founder(spc, "AA")
aa <- spc$founder("aa")

F1  <- cross(n = 1, AA, aa) # the hybrid

## set a infinitesimal & a quantitative fitness
infty <- set_infty(spc, m = 0, a = 1, d = .5) # partial dominance
genes <- data.frame(snp = sample(ToyMap$snp, 10), add = rnorm(10), dom = rnorm(10))
quant <- set_quant(spc, m = 0, data = genes)

## evaluating the breeding value
infty$alpha(AA)
quant$alpha(F1)
```

`founder` *Constructor of a Founder Instances of the Specimen Class*

Description

Constructor of instances of the Specimen class given the Specie from which the individual will belong where all loci will equal to the provided genotype.

Usage

```
founder(specie, code)
```

Arguments

`specie`	an instance of the R6 class Specie with the genome's parameters.
`code`	a length one character vector with one of the genotype codes: "AA", "Aa", "aA" or "aa".

Details

Genotypes can be coded as **AA**, **Aa**, **aA** or **aa**, that meant to represent both homozigous (**AA** and **aa**) as well as both heterozigous (**Aa** and **aA**).

Value

Objects of R6 class with methods to mimic in silico Specimens.

Examples

```
data(ToyMap)
spc <- set_specie(ToyMap)

## through standalone function
AA <- founder(spc, "AA")
aa <- founder(spc, "aa")

## or by the Specie's method
Aa <- spc$founder("Aa")
aA <- spc$founder("aA")
```

`genotype` *Codify Specimens' Genotypes.*

Description

Codify Specimens' genotypes instances as numeric codes [-1/0/1] or as character vector that keeps the phase information.

Usage

```
genotype(pop, phase = FALSE)
```

Arguments

`pop`	a list with instances of the R6 class Specimen.
`phase`	logical should the codes keep the phase.

Value

A numeric or character matrix with the codified Specimens' genotypes.

Examples

```
data(ToyMap)
spc <- set_specie(ToyMap)

Aa <- founder(spc, "Aa")
aA <- spc$founder("aA")

Both <- list(Aa = Aa, aA = aA)

## different ways
genotype(Both)               # as numeric
genotype(Both, phase = TRUE) # as character
```

`import` *Constructor of a Custom Instances of the Specimen Class*

Description

Constructor of instances of the Specimen class given the Specie from which the individual will belong where the loci will equal to the provided genotype from two strings one for each homologous.

Usage

```
import(specie, genotype)
```

Arguments

`specie`	an instance of the R6 class Specie with the genome's parameters.
`genotype`	a named character vector with the coded/phased genotypes.

Value

Objects of R6 class with methods to mimic in silico Specimens.

Examples

```
data(ToyMap)
spc <- set_specie(ToyMap)

## simulating what is very close to your real genotypes
Real <- sample(c('2 2', '2 1', '1 2', '1 1'), size = nrow(ToyMap), replace = TRUE)
names(Real) <- ToyMap$snp # ensure snp names!

## now you can play _in silico_
Virtual <- import(spc, Real)
S1 <- Virtual$selfcross(n = 10)
```

mating — *Breed Simulated Individuals According to Basic Mating Schemes*

Description

Performs the simple mating schemes bi-parental cross, self-cross and haploid duplication, respectively through the functions `cross`, `selfcross` and `dh` and return the respective size *n* progeny involving the parental individuals belonging to the same specie.

Usage

```
cross(n = 1, p1, p2)

selfcross(n = 1, gid)

dh(n = 1, gid)
```

Arguments

`n`	a length-one integer vector with the size of the progeny.
`p1, p2, gid`	are instances of the class specimen which will be used as the parents.

Details

Basically this family of functions take simulated individuals belonging to the same simulated specie, performs the meiosis that generates individual's gametes. According to the scheme applied the gametes are merged into new simulated individuals. These are wrap functions to the C++ class that mimic the meiosis recombination process.

Value

a size *n* list with instances of the class Specimen that represent new individuals belonging to the progeny of the respective mating scheme.

Examples

```
data(ToyMap)
spc <- set_specie(ToyMap)
AA <- founder(spc, "AA")
aa <- founder(spc, "aa")

## Mather Design
F1  <- cross(n = 1, AA, aa)
BC1 <- cross(n = 5, F1, AA)
BC2 <- F1$cross(n = 5, aa)   # using R6 methods
F2  <- selfcross(n = 10, F1)
RIL <- dh(n = 10, F1)
## chainable R6 methods
```

```
F3  <- F1$selfcross(n = 1, replace = TRUE)$selfcross(n = 1, replace = TRUE)
```

`set_specie` *Constructor of Instances of the Specie Class*

Description

Constructor of instances of the Specie class given the map of the genome and optionally a pointer to a C++ function which will drive the meiosis process.

Usage

```
set_specie(data, meiosis = NULL)
```

Arguments

`data`	A data frame with the map of the Genome to be simulates.
`meiosis`	A pointer to a C++ function of the meiosis process.

Details

By standard the meiosis recombination and de novo genetic variability is generated by means a count-location process (Karlin and Liberman 1978). Users may define other process, see extdata for the examples presented in Toledo et al. (2019).

Value

Objects of R6 class with methods to mimic in silico Genomes.

References

Karlin, Liberman (1978). “Classifications and comparisons of multilocus recombination distributions.” *Proceedings of the National Academy of Sciences of the United States of America*, **75**(12), 6332–6336. `https://doi.org/10.1073/pnas.75.12.6332`.

Toledo FH, Pérez-Rodríguez P, Crossa J, Burgueño J (2019). “isqg: A Binary Framework for in Silico Quantitative Genetics".” *G3: Genes, Genomes, Genetics*, **9**(8), 2425–2428. `https://doi.org/10.1534/g3.119.400373`.

Examples

```
data(ToyMap)
spc_standard <- set_specie(ToyMap)

## generate standard _de novo_ variability
spc_standard$gamete(n = 100)
```

```
## Not run:
## write your function in C++ and then wrap it as a pointer
## check the examples in extdata
## compile the code
Rcpp::sourceCpp(file = system.file("extdata", "Independent.cpp", package = "isqg"),
                rebuild = TRUE)

## define a specie w/ custom meiosis
spp_custom <- set_specie(ToyMap, meiosis = indepp())

## check meiosis process
spp_custom$gamete(n = 100)

## End(Not run)
```

`Specie` *Class providing object with methods to mimic in silico Genomes*

Description

Mean to mimic a in silico Genomes. It is the machine instances of the simulator.

Details

Object of R6 class that points to C++ objetcs.

Value

Objects of R6 class with methods to mimic in silico Genomes.

Public fields

`.ptr` External pointer to the instance of the C++ class Specie.

Methods

Public methods:

- `.R_Specie_ctor$new()`
- `.R_Specie_ctor$print()`
- `.R_Specie_ctor$founder()`
- `.R_Specie_ctor$gamete()`
- `.R_Specie_ctor$map()`
- `.R_Specie_ctor$clone()`

Method `new()`**:** Create an instance of a Specie.

Usage:

`.R_Specie_ctor$new(ptr)`

Arguments:

`ptr` an Smart pointer to an instance of a Specie C++ class.

Returns: A new ‘Specie‘ object.

Method `print()`**:** Print/Show an instance of the Specie class.

Usage:

`.R_Specie_ctor$print(...)`

Arguments:

`...` further arguments to be passed to print.

Method `founder()`**:** Constructor of a Founder Instances of the Specimen Class

Usage:

`.R_Specie_ctor$founder(code)`

Arguments:

`code` a length one character vector with one of the genotype codes: "AA", "Aa", "aA" or "aa".

Method `gamete()`**:** Generate gamete prototypes for this specie.

Usage:

`.R_Specie_ctor$gamete(n)`

Arguments:

`n` an integer number with the number of gametes prototype to be generated.

Returns: a vector of strings with the gamete prototypes.

Method `map()`**:** Retrieve the map of the current specie.

Usage:

`.R_Specie_ctor$map()`

Returns: a data.frame with the map of the specie.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`.R_Specie_ctor$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

`Specimen` *Class providing object with methods to mimic in silico Specimens*

Description

Mean to mimic a in silico Specimens. It is the working instances of the simulator.

Details

Object of R6 class that points to C++ objetcs.

Value

Objects of R6 class with methods to mimic in silico Specimens.

Public fields

`.ptr` External pointer to the instance of the C++ class Specie.

Methods

Public methods:

- `.R_Specimen_ctor$new()`
- `.R_Specimen_ctor$print()`
- `.R_Specimen_ctor$alpha()`
- `.R_Specimen_ctor$genotype()`
- `.R_Specimen_ctor$cross()`
- `.R_Specimen_ctor$selfcross()`
- `.R_Specimen_ctor$dh()`
- `.R_Specimen_ctor$mirror()`
- `.R_Specimen_ctor$look()`
- `.R_Specimen_ctor$clone()`

Method `new()`**:** Create an instance of a Specimen.

Usage:

```
.R_Specimen_ctor$new(ptr)
```

Arguments:

`ptr` an Smart pointer to an instance of a Specimen C++ class.

Returns: A new ‘Specimen‘ object.

Method `print()`**:** Print/Show an instance of the Specimen class.

Usage:

```
.R_Specimen_ctor$print(...)
```

Arguments:

`...` further arguments to be passed to print.

Method `alpha()`**:** Evaluates the breeding value.

Usage:

```
.R_Specimen_ctor$alpha(trait)
```

Arguments:

`trait` an instance of the class Trait.

Returns: the breeding value of the specimen for the given trait.

Method `genotype()`**:** Codify Specimen's Genotypes.

Usage:

```
.R_Specimen_ctor$genotype(phase = FALSE)
```

Arguments:

`phase` logical should the codes keep the phase.

Returns: A numeric or character vector with the codified Specimen's genotypes.

Method `cross()`**:** Performs the simple bi-parental cross.

Usage:

```
.R_Specimen_ctor$cross(n = 1, gid)
```

Arguments:

`n` a length-one integer vector with the size of the progeny.

`gid` instance of the class specimen which will be used to mate.

Returns: a size *n* list with instances of the class Specimen that represent new individuals belonging to the progeny of the respective mating scheme.

Method `selfcross()`**:** Performs the selfcross.

Usage:

```
.R_Specimen_ctor$selfcross(n = 1, replace = FALSE)
```

Arguments:

`n` a length-one integer vector with the size of the progeny.

`replace` logical scalar indicating if the outcome of the function will replace the current instance of the Specimen

Returns: a size *n* list with instances of the class Specimen that represent new individuals belonging to the progeny of the respective mating scheme.

Method `dh()`**:** Performs the double-haploid duplication

Usage:

```
.R_Specimen_ctor$dh(n = 1, replace = FALSE)
```

Arguments:

`n` a length-one integer vector with the size of the progeny.

`replace` logical scalar indicating if the outcome of the function will replace the current instance of the Specimen

Returns: a size *n* list with instances of the class Specimen that represent new individuals belonging to the progeny of the respective mating scheme.

Method `mirror()`**:** Generates a ‘mirrored‘ specimen.

Usage:

```
.R_Specimen_ctor$mirror()
```

Returns: an instance of the Specimen class with all loci mirrored.

Method `look()`**:** Acess specific locus’ value from specimen.

Usage:

```
.R_Specimen_ctor$look(snp, phase = FALSE)
```

Arguments:

`snp` an character string with the name of the locus to lookup.

`phase` logical should the codes keep the phase.

Returns: the genotype of the given locus.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
.R_Specimen_ctor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ToyMap` *Toy Example of a Map for* in silico *Quantitative Genetics*

Description

This data comprise 2 chromossomes with 2cM each and 21 monitored loci in each chromosome

Usage

```
data(ToyMap)
```

Format

a data.frame, 42 rows and 3 collumns (snp, chr, pos).

`Trait` *Class providing object with methods to mimic in silico Traits*

Description

Mean to mimic a in silico Trait. It is the working instances of the simulator.

Details

Object of R6 class that points to C++ objetcs.

Value

Objects of R6 class with methods to mimic in silico Traits.

Public fields

`.ptr` External pointer to the instance of the C++ class Specie.

Methods

Public methods:

- `.R_Trait_ctor$new()`
- `.R_Trait_ctor$print()`
- `.R_Trait_ctor$alpha()`
- `.R_Trait_ctor$clone()`

Method `new()`**:** Create an instance of a Trait.

Usage:

```
.R_Trait_ctor$new(ptr)
```

Arguments:

`ptr` an Smart pointer to an instance of a Trait C++ class.

Returns: A new ‘Trait‘ object.

Method `print()`**:** Print/Show an instance of the Trait class.

Usage:

```
.R_Trait_ctor$print(...)
```

Arguments:

`...` further arguments to be passed to print.

Method `alpha()`**:** Evaluates the breeding value.

Usage:

```
.R_Trait_ctor$alpha(gid)
```

Arguments:

`gid` an instance of the class Specimen.

Returns: the breeding value of the given specimen for the trait.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
.R_Trait_ctor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index