Package 'PopGenome'

January 31, 2020

Type Package

Title An Efficient Swiss Army Knife for Population Genomic Analyses

Version 2.7.5

Date 2020-01-29

Depends R (>= 2.14.2), ff

Imports methods

Suggests parallel, bigmemory, BASIX, WhopGenome, BlockFeST

Description Provides efficient tools for population genomics data analysis,
able to process individ-
ual loci, large sets of loci, or whole genomes. PopGenome <DOI:10.1093/molbev/msu136> not only implements a wide range of population genetics statistics, but also facilitates the easy implementation of new algorithms by other researchers. PopGenome is optimized for speed via the seamless integration of C code.

License GPL-3

LazyLoad yes

Copyright inst/COPYRIGHTS

SystemRequirements zlib headers and library.

Repository CRAN

NeedsCompilation yes

Author Bastian Pfeifer [aut, cre],
Ulrich Wittelsbuerger [ctb],
Heng Li [ctb],
Bob Handsaker [ctb]

Maintainer Bastian Pfeifer <bastianxpfeifer@gmail.com>

Date/Publication 2020-01-31 22:50:03 UTC

R topics documented:

`Achaz.stats-methods` *Achaz statistic*

Description

Achaz statistic

Usage

```
## S4 method for signature 'GENOME'
Achaz.stats(object,new.populations=FALSE,new.outgroup=FALSE,subsites=FALSE)
```

Arguments

`object`	an object of class `"GENOME"`
`new.populations`	list of populations. default:`FALSE`
`new.outgroup`	outgroup vector. default:`FALSE`
`subsites`	`"transitions"`: SNPs that are transitions. `"transversions"`: SNPs that are transversions. `"syn"`: synonymous sites. `"nonsyn"`: nonsynonymous sites. `"exon"`: SNPs in exon regions. `"intron"`: SNPs in intron regions. `"coding"`: SNPs in coding regions (CDS). `"utr"`: SNPs in UTR regions. `"gene"`: SNPs in genes. default:`FALSE`

Value

returned value is a modified object of class "GENOME"

The following Slots will be modified in the "GENOME" object

Yach	Achaz Y statistic

References

Achaz G.,2008 *Testing for neutrality in samples with sequencing errors.* Genetics 179: 1409.

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class <- Achaz.stats(GENOME.class)
# GENOME.class <- Achaz.stats(GENOME.class,list(1:7,8:12))
# show the result:
# GENOME.class@Yach
```

BayeScanR *An R implementation of BayeScan (Foll \& Gagiotti 2008)*

Description

BayeScanR is an R implementation of BayeScan for analysis of codominant markers.

Usage

```
BayeScanR(input,nb.pilot=10,pilot.runtime=2500,main.runtime=100000, discard=50000)
```

Arguments

input	textfile or an R-object returned by getBayes()
nb.pilot	number of pilot runs
pilot.runtime	length of pilot runs
main.runtime	length of main runs
discard	how many runs in the main.loop should be discarded?

Value

returned value is an object of class "BAYESRETURN"

The following Slots will be filled

---

alpha	alpha effects
beta	beta effects
var_alpha	variance of alpha values
a_inc	which alpha is included in the model
fst	FST values
P	P-value

References

[1] Foll M and OE Gaggiotti (2008). *A genome scan method to identify selected loci appropriate for both dominant and codominant markers: A Bayesian perspective.* Genetics 180: 977-993

Examples

```
# GENOME.class <- readData("...\Alignments")
# GENOME.class <- F_ST.stats(GENOME.class,list(1:5,6:10))
# Bayes.input  <- getBayes(GENOME.class)
# BAYES.class  <- BayeScanR(Bayes.input)
# BAYES.class
```

---

calc.fixed.shared-methods

Fixed and shared polymorphisms

Description

A generic function to calculate the number of fixed and shared polymorphisms.

Usage

```
## S4 method for signature 'GENOME'
calc.fixed.shared(object,
subsites=FALSE,
new.populations=FALSE,
fixed.threshold=1,
fixed.threshold.fst=1)
```

Arguments

object	An object of class "GENOME"
subsites	"transitions": SNPs that are transitions. "transversions": SNPs that are transversions. "syn": synonymous sites. "nonsyn": nonsynonymous sites. "exon": SNPs in exon regions. "intron": SNPs in intron regions. "coding": SNPs in coding regions (CDS). "utr": SNPs in UTR regions. "gene": SNPs in genes. "intergenic" : SNPs in intergenic regions.
new.populations	list of populations. default=FALSE
fixed.threshold	Polymorphisms are considered as fixed >= threshold value
fixed.threshold.fst	Polymorphisms are considered as fixed >= threshold value

Details

The nucleotide diversities have to be devided by GENOME.class@n.sites to give diversities per site.

Value

Returned value is a modified object of class "GENOME"

The following slots will be modified in the "GENOME" object

	Slot	Reference	Description
1.	n.fixed.sites	[x]	Number of fixed sites
2.	n.shared.sites	[x]	Number of shared sites

3. `n.monomorphic.sites` [x] Number of monomorphic sites

References

[x]

Examples

```
# GENOME.class <- readData("\home\Alignments")
# set.populations
# GENOME.class <- calc.fixed.shared(GENOME.class)
```

`calc.R2-methods` *Linkage statistics (R2, P-value, Distance)*

Description

This generic function calculates some linkage disequilibrium statistics.

Usage

```
## S4 method for signature 'GENOME'
calc.R2(object,subsites=FALSE,lower.bound=0,upper.bound=1)
```

Arguments

`object`	an object of class `"GENOME"`
`subsites`	same as in the other modules
`lower.bound`	sites with minor-allele-frequency>=lower.bound are considered
`upper.bound`	sites with minor-allele-frequency<=upper.bound are considered

Details

Note, the pairwise comparisons are computed via `combn(n.snps,2)`.

Value

The slot GENOME.class@region.stats@linkage.disequilibrium will be filled.
(R2,P-value,Distance)
Fisher's Exact Test is used for the P-values.

Examples

```
# GENOME.class <- readData("...\Alignments")
# GENOME.class
# GENOME.class <- calc.R2(GENOME.class)
# show the result:
# GENOME.class@region.stats@linkage.disequilibrium
# [[x]][[y]] x:region, y:population
```

`codontable` *Prints the codon table which is used in the PopGenome framework*

Description

This functions prints the nucleotide triplets (as numerical values) and the correspnding protein character strings.

Usage

```
codontable()
```

Arguments

	no arguments

Details

The returned value is a list including two matrices.
The first matrix contains the amino acids and the second matrix the corresponding nucleotide triplets. In the PopGenome Vignette you can see how to manipulate these tables to use alternative genetic codes.

Examples

```
# table <- codontable()
# table$Proteins
# table$Triplets
```

concatenate.classes *Concatenate GENOME classes*

Description

This function concatenates objects of class GENOME, allowing to stitch together larger datasets from smaller objects.

Usage

```
concatenate.classes(classlist)
```

Arguments

classlist	a list of GENOME objects

Value

The function creates an object of class "GENOME".

Examples

```
# a   <- readData("Three_Alignments/")
# b   <- readData("Two_Alignments/")
# ab  <- concatenate.classes(list(a,b))
# ab  <- neutrality.stats(ab)
# ab@Tajima.D
# ab@region.names
```

concatenate.regions *Concatenate regions*

Description

This function concatenates the regions/chunks contained in one GENOME object.

Usage

```
concatenate.regions(object)
```

Arguments

object	object of class GENOME

Value

The function creates an object of class "GENOME".

Examples

```
# GENOME.class <- readData("Three_Alignments/")
# WHOLE        <- concatenate.regions(GENOME.class)
# WHOLE        <- neutrality.stats(WHOLE)
# WHOLE@Tajima.D
```

---

count.unknowns-methods

Calculate missing nucleotide frequencies

Description

A generic function to calculate the missing nucleotide frequencies.

Usage

```
## S4 method for signature 'GENOME'
count.unknowns(object)
```

Arguments

object	An object of class "GENOME"

Value

Returned value is a modified object of class "GENOME"

The slot GENOME.class@missing.freqs for the missing frequencies for the whole region.
The slot GENOME.class@region.stats@missing.freqs for the missing frequencies for each SNP in a given region

Examples

```
# GENOME.class <- readData("VCF", format="VCF", include.unknown=TRUE)
# GENOME.class@region.stats
# GENOME.class <- count.unknowns(GENOME.class)
# GENOME.class@missing.freqs
# GENOME.class@region.stats@missing.freqs
```

---

create.PopGenome.method

Integration of own functions into the PopGenome-framework

Description

This function generates a skeleton for a PopGenome function. It thereby facilitates the effortless integration of new methods into the PopGenome framework.

Usage

```
create.PopGenome.method(function.name,population.specific=TRUE)
```

Arguments

function.name	name of your function
population.specific	TRUE:function returns one value per population.FALSE:function returns one value calculated across all populations (as in the case of FST measurements)

Details

This mechanism enables you to use your own functions in the PopGenome environment. The functions can also be applied to sliding windows or subsites.
Please look at the generated function, which documents where to place your own function in detail.

Examples

```
# GENOME.class <- readData(".../Alignments")
# create.PopGenome.method("myFunction")
# edit myFunction.R
# source("myFunction")
# value <- myFunction(test)
# value
```

`detail.stats-methods` *Several statistics*

Description

This generic function calculates some mixed statistics.

Usage

```
## S4 method for signature 'GENOME'
detail.stats(
object,
new.populations=FALSE,
new.outgroup=FALSE,
subsites=FALSE,
biallelic.structure=FALSE,
mismatch.distribution=FALSE,
site.spectrum=TRUE,
        site.FST=FALSE
        )
## S4 method for signature 'GENOME'
get.detail(object, biallelic.structure=FALSE)
```

Arguments

`object`	an object of class `"GENOME"`
`new.populations`	list of populations.
`new.outgroup`	outgroup sequences.
`subsites`	`"transitions"`: SNPs that are transitions. `"transversions"`: SNPs that are transversions. `"syn"`: synonymous sites. `"nonsyn"`: nonsynonymous sites. `"exon"`: SNPs in exon regions. `"intron"`: SNPs in intron regions. `"coding"`: SNPs in coding regions (CDS). `"utr"`: SNPs in UTR regions. `"gene"`: SNPs in genes.
`biallelic.structure`	fixed and shared polymorphisms (stored in `GENOME.class@region.stats`).

`mismatch.distribution`
: statistics based on mismatch distribution

`site.spectrum`
: minor allele frequency of each SNP

`site.FST`
: computes FST for each SNP

Value

The return value is a modified object of class `"GENOME"`

The following Slots will be modified in the `"GENOME"` object

`MDSD`
: ...

`MDG1`
: ...

`MDG2`
: ...

`region.stats`
: the slot `biallelic.structure` and `minor.allele.freqs` will be filled

The function `get.detail(GENOME.class,biallelic.structure=TRUE)`
returns a matrix for each region, where

`0`
: population is polymorphic, the remaining individuals are polymorphic

`1`
: population is polymorphic, the remaining individuals are monomorphic

`2`
: population is monomorphic, the remaining individuals are polymorphic

`3`
: population is monomorphic, the remaining individuals are monomorphic with the same value

`4`
: population is monomorphic, the remaining individuals are monomorphic with different values

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class <- set.populations(GENOME.class,list(1:10))
# GENOME.class <- detail.stats(GENOME.class)
# show the result:
# mismatch.values   <- get.detail(GENOME.class)
# bial.struc.values <- get.detail(GENOME.class, biallelic.structure=TRUE)
# GENOME.class@region.stats@biallelic.structure
```

```
# GENOME.class@region.stats@biallelic.structure[[1]]
```

`diversity.stats-methods` *Diversities*

Description

A generic function to calculate nucleotide & haplotype diversities.

Usage

```
## S4 method for signature 'GENOME'
diversity.stats(object,new.populations=FALSE,subsites=FALSE,pi=FALSE, keep.site.info=TRUE)
```

Arguments

`object`	An object of class `"GENOME"`
`new.populations`	list of populations. default=`FALSE`
`subsites`	`"transitions"`: SNPs that are transitions. `"transversions"`: SNPs that are transversions. `"syn"`: synonymous sites. `"nonsyn"`: nonsynonymous sites. `"exon"`: SNPs in exon regions. `"intron"`: SNPs in intron regions. `"coding"`: SNPs in coding regions (CDS). `"utr"`: SNPs in UTR regions. `"gene"`: SNPs in genes. `"intergenic"` : SNPs in intergenic regions.
`pi`	Nei's calculation of pi
`keep.site.info`	stores site specific values in `GENOME.class@region.stats`

Details

The nucleotide diversities have to be devided by `GENOME.class@n.sites` to give diversities per site.

Value

Returned value is a modified object of class "GENOME"

The following slots will be modified in the "GENOME" object

	Slot	Reference	Description
1.	nuc.diversity.within	[1,3]	Nucleotide diversity (within the population)
2.	Pi	[2]	Diversity from Nei (within the population)
3.	hap.diversity.within	[1]	Haplotype diversity (within the population)

References

[1] Hudson, R. R., M. Slatkin, and W.P. Maddison (1992). *Estimating of levels of gene flow from DNA sequence data.* Gentics 13(2),583-589

[2] Nei, M. (1987). *Molecular Evolutionary Genetics.* Columbia Univ. Press, New York.

[3] Wakeley, J. (1996).*The Variance of Pairwise Nucleotide Differences in Two Populations with Migration.* THEORETICAL POPULATION BIOLOGY. 49, 39-57.

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class <- diversity.stats(GENOME.class)
# GENOME.class <- diversity.stats(GENOME.class,list(1:4,5:10))
# GENOME.class <- diversity.stats(GENOME.class,
# list(c("seq1","seq5","seq3"),c("seq2","seq8")))
# show the result:
# GENOME.class@nuc.diversity.within
```

---

diversity.stats.between-methods
Diversities

Description

A generic function to calculate nucleotide & haplotype diversities between populations (dxy).

Usage

```
## S4 method for signature 'GENOME'
diversity.stats.between(object,new.populations=FALSE,subsites=FALSE,keep.site.info=FALSE,
haplotype.mode=FALSE, nucleotide.mode=TRUE)
```

Arguments

object	An object of class "GENOME"
new.populations	list of populations. default=FALSE
subsites	"transitions": SNPs that are transitions. "transversions": SNPs that are transversions. "syn": synonymous sites. "nonsyn": nonsynonymous sites. "exon": SNPs in exon regions. "intron": SNPs in intron regions. "coding": SNPs in coding regions (CDS). "utr": SNPs in UTR regions. "gene": SNPs in genes. "intergenic" : SNPs in intergenic regions.
keep.site.info	Store SNP specific values in the region.stats
haplotype.mode	Haplotype Diversities
nucleotide.mode	Nucleotide Diversities

Details

The nucleotide diversities have to be devided by GENOME.class@n.sites to give diversities per site.

Value

Returned value is a modified object of class "GENOME"

The following slots will be modified in the "GENOME" object

	Slot	Reference	Description
1.	nuc.diversity.between	[1,3]	Nucleotide diversity (between the population)
2.	hap.diversity.between	[1]	Haplotype diversity (between the population)

References

[1] Hudson, R. R., M. Slatkin, and W.P. Maddison (1992). *Estimating of levels of gene flow from DNA sequence data.* Gentics 13(2),583-589

[2] Wakeley, J. (1996).*The Variance of Pairwise Nucleotide Differences in Two Populations with Migration.* THEORETICAL POPULATION BIOLOGY. 49, 39-57.

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class <- diversity.stats.between(GENOME.class)
# GENOME.class <- set.populations(GENOME.class, list(...))
# GENOME.class <- diversity.stats.between(GENOME.class)
# GENOME.class <- diversity.stats(GENOME.class,
# list(c("seq1","seq5","seq3"),c("seq2","seq8")))
# show the result:
# GENOME.class@nuc.diversity.within
```

---

fasta_file *FASTA file (subdirectory "data")*

Description

The FASTA files (unpacked) in the subdirectory "data" of the PopGenome package have to be stored in a folder (multiple files can be stored in this folder). The folder name is then used as the input for the `readData` function.

F_ST.stats-methods *Fixation Index*

Description

A generic function to calculate some F-statistics and nucleotide/haplotype diversities.

Usage

```
## S4 method for signature 'GENOME'
F_ST.stats(
object,
new.populations=FALSE,
subsites=FALSE,
```

```
detail=TRUE,
mode="ALL",
only.haplotype.counts=FALSE,
FAST=FALSE
)

## S4 method for signature 'GENOME'
get.diversity(object,between=FALSE)
## S4 method for signature 'GENOME'
get.F_ST(object,mode=FALSE,pairwise=FALSE)
```

Arguments

object	An object of class "GENOME"
new.populations	list of populations. default:FALSE
subsites	"transitions": SNPs that are transitions. "transversions": SNPs that are transversions. "syn": synonymous sites. "nonsyn": nonsynonymous sites. "exon": SNPs in exon regions. "intron": SNPs in intron regions. "coding": SNPs in coding regions (CDS). "utr": SNPs in UTR regions. "gene": SNPs in genes. "intergenic" : SNPs in intergenic regions.
detail	detail statistics. Note: slower!
between	TRUE: show between-diversities. FALSE: show within-diversities
mode	mode="haplotype" or mode="nucleotide"
only.haplotype.counts	only calculate the haplotype counts
FAST	if TRUE only calculate a subset of statistics. see details!
pairwise	show paiwise comparisons. default:FALSE

Details

If FAST is switched on, this module only calculates nuc.diversity.within, hap.diversity.within, haplotype.F_ST, nucleotide.F_ST and pi.
Note:
1) The nucleotide diversities have to be devided by the size of region considered (e.g. GENOME@n.sites) to give diversities per site.
2) When missing or unknown nucleotides are included (include.unknown=TRUE) those sites are completely deleted in case of haplotype based statistics.
3) The function detail.stats(...,site.FST=TRUE) will compute SNP specific FST values which are then stored in the slot GENOME.class@region.stats@site.FST.

4) We recommend to use `mode="nucleotide"` in case you have many unknowns included in your dataset.

Value

	Slot	Reference	Description
1.	`haplotype.F_ST`	[1]	Fixation Index based on haplotype frequencies
2.	`nucleotide.F_ST`	[1]	Fixation Index based on minor.allele frequencies
3.	`Nei.G_ST`	[2]	Nei's Fixation Index
4.	`Hudson.G_ST`	[3]	see reference ...
5.	`Hudson.H_ST`	[3]	see reference ...
6.	`Hudson.K_ST`	[3]	see reference ...
7.	`nuc.diversity.within`	[1,5]	Nucleotide diversity (within the population)
8.	`hap.diversity.within`	[1]	Haplotype diversity (within the population)
9.	`Pi`	[4]	Nei's diversity (within the population)
10.	`hap.F_ST.vs.all`	[1]	Fixation Index for each population against all other individuals (haplotype)
11.	`nuc.F_ST.vs.all`	[1]	Fixation Index for each population against tall other individuals (nucleotide)
12.	`hap.diversity.between`	[1]	Haplotype diversities between populations
13.	`nuc.diversity.between`	[1,5]	Nucleotide diversities between populations
14.	`nuc.F_ST.pairwise`	[1]	Fixation Index for every pair of populations (nucleotide)
15.	`hap.F_ST.pairwise`	[1]	Fixation Index for every pair of populations (haplotype)
16.	`Nei.G_ST.pairwise`	[2]	Fixation Index for every pair of populations (Nei)
17.	`region.stats`		an object of class "region.stats" for detailed statistics

References

[1] Hudson, R. R., M. Slatkin, and W.P. Maddison (1992). *Estimating levels of gene flow from DNA sequence data.* Gentics 13(2),583-589

[2] Nei, M. (1973). *Analysis of gene diversity in subdivided populations.* Proc.Natl. Acad. Sci. USA 70: 3321-3323

[3] Hudson, R. R., Boos, D.D. and N. L. Kaplan (1992). *A statistical test for detecting population subdivison.* Mol. Biol. Evol. 9: 138-151.

[4] Nei, M. (1987). *Molecular Evolutionary Genetics.* Columbia Univ. Press, New York.

[5] Wakeley, J. (1996).*The Variance of Pairwise Nucleotide Differences in Two Populations with Migration.* THEORETICAL POPULATION BIOLOGY. 49, 39-57.

See Also

methods?F_ST.stats.2 #F_ST.stats.2

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class <- F_ST.stats(GENOME.class)
# GENOME.class <- F_ST.stats(GENOME.class,list(1:4,5:10),subsites="syn")
# GENOME.class <- F_ST.stats(GENOME.class,list(c("seq1","seq5","seq3"),c("seq2","seq8")))
# show the result:
# get.F_ST(GENOME.class)
# get.F_ST(GENOME.class, pairwise=TRUE)
# get.diversity(GENOME.class, between=TRUE)
# GENOME.class@Pi --> population specific view
# GENOME.class@region.stats
```

F_ST.stats.2-methods *Fixation Index (2)*

Description

A generic function to calculate some FST measurenments.

Usage

```
## S4 method for signature 'GENOME'
F_ST.stats.2(object,new.populations="list",subsites=FALSE,snn=TRUE,Phi_ST=FALSE)
```

Arguments

object	An object of class "GENOME"
new.populations	list of populations. default=FALSE
subsites	"transitions": SNPs that are transitions. "transversions": SNPs that are transversions. "syn": synonymous sites. "nonsyn": nonsynonymous sites. "exon": SNPs in exon regions. "intron": SNPs in intron regions. "coding": SNPs in coding regions (CDS). "utr": SNPs in UTR regions. "gene": SNPs in genes. "intergenic" : SNPs in intergenic regions.
snn	Snn statistic from Hudson
Phi_ST	Statistic from Excoffier et al.

Value

Returned value is an modified object of class "GENOME"

Following slots will be modified in the "GENOME" object

	Slot	Reference	Description
1.	Hudson.Snn	[1]	Snn statistic from Hudson (2000)
2.	Phi_ST	[2]	Phi_ST from Excoffier (1992)

References

[1] Hudson, R. R. (2000).*A new statistic for detecting genetic differentiation.* Genetics 155: 2011-2014.

[2] Excoffier, L., Smouse, P., Quattro, J. (1992),*Analysis of molecular variance inferred from metric distances among DNA haplotypes: application to human mitochondrial DNA restriction data.* Genetics 131: 479-91

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class <- F_ST.stats.2(GENOME.class)
# GENOME.class <- F_ST.stats.2(GENOME.class,list(1:4,5:10))
# GENOME.class <- F_ST.stats.2(GENOME.class,
# list(c("seq1","seq5","seq3"),c("seq2","seq8")))
# show the result:
# GENOME.class@Hudson.Snn
```

GENOME-class *Class "GENOME"*

Description

A class where all data and calculated values are stored

Slots

BIG.BIAL: Biallelic matrix as an ff-object

SLIDE.POS: Positions of biallelic sites (Sliding window mode)

big.data: ff-package ?

`gff.info`: Gff information ?

`snp.data`: SNP data ?

`basepath`: The basepath of the data

`project`: —-

`populations`: Populations definded before reading data

`poppairs`: —

`outgroup`: A vector of outgroup sequences

`region.names`: Names/identifier of each region

`feature.names`: Feature attributes of a given region

`genelength`: Number of regions

`keep.start.pos`: Start positions for sliding window

`n.sites`: Total number of sites

`n.sites2`: Total number of sites

`n.biallelic.sites`: Number of biallelic sites (SNPs)

`n.gaps`: Number of gaps observed in the data

`n.unknowns`: Number of unknown.positions

`n.valid.sites`: Sites without gaps

`n.polyallelic.sites`: Sites with more than two variants

`trans.transv.ratio`: Transition-transversion ratio

`Coding.region`: Number of nucleotides in CDS regions

`UTR.region`: Number of nucleotides in UTR regions

`Intron.region`: Number of nucleotides in Intron regions

`Exon.region`: Number of nucleotides in Exon regions

`Gene.region`: Number of nucleotides in Gene regions

`Pop_Neutrality`: Populations defined in the neutrality module

`Pop_FSTN`: Populations defined in the FST (nucleotide) module

`Pop_FSTH`: Populations defined in the FST (haplotype) module

`Pop_Linkage`: Populations defined in the Linkage module

`Pop_Slide`: —

`Pop_MK`: Populations defined in the MK module

`Pop_Detail`: Populations defined in the Detail module

`Pop_Recomb`: Populations defined in the Recombination module

`Pop_Sweeps`: Populations defined in the Selective sweeps module

`FSTNLISTE`: —

`nucleotide.F_ST`: Nucleotide FST

`nucleotide.F_ST2`: —

`nuc.diversity.between`: Nucleotide diversity between the populations

`nuc.diversity.within`: Nucleotide diversity within the populations

`nuc.F_ST.pairwise`: FST for each pair of populations

`nuc.F_ST.vs.all`: FST for one population vs. all other individuals

`n.haplotypes`: —

`hap.diversity.within`: Haplotype diversity withing the populations

`hap.diversity.between`: Haplotype diversity between the populations

`Pi`: Pi from Nei

`PIA_nei`: Pi between the populations

`haplotype.counts`: Counts of the haplotypes observed

`haplotype.F_ST`: Haplotype FST

`hap.F_ST.pairwise`: Haplotype diversity for each pair of populations

`Nei.G_ST.pairwise`: Haplotype diversity for each pair of populations

`hap.F_ST.vs.all`: FST for one population vs. all other individuals

`Nei.G_ST`: GST from Nei

`Hudson.G_ST`: GST from Hudson

`Hudson.H_ST`: HST from Hudson

`Hudson.K_ST`: KST from Hudson

`Hudson.Snn`: Snn from Hudson

`Phi_ST`: Fixation index from Excoffier

`hap.pair.F_ST`: —

`MKT`: Mcdonald-Kreitman values

`Tajima.D`: Tajima's D

`SLIDE`: —

`Fay.Wu.H`:

`Zeng.E`:

`theta_Tajima`:

`theta_Watterson`:

`theta_Fu.Li`:

`theta_Achaz.Watterson`:

`theta_Achaz.Tajima`:

`theta_Fay.Wu`:

`theta_Zeng`:

`Fu.Li.F`:

`Fu.Li.D`:

`Yach`:

`n.segregating.sites`: Total number of segregating sites

`Rozas.R_2`:

Fu.F_S:

Strobeck.S:

Kelly.Z_nS:

Rozas.ZZ:

Rozas.ZA:

Wall.B:

Wall.Q:

mult.Linkage: Linkage disequilibrium between regions

RM: Minimum number of recombination events (Hudson)

CL: Composite likelihood of SNPs (Nielsen et. al)

CLmax: Max. composite likelihood of SNPs (Nielsen et.al)

CLR: Composite likelihood ratio test (Nielsen et. al)

MDSD:

MDG1:

MDG2:

genes:

region.data: Detailed information about the data

region.stats: Detailed (site-specific) statistics

D Pattersons D statistic

f the fraction of the genome that is admixed

jack.knife jacknife mode

missing.freqs: Missing nucleotide frequency

n.fixed.sites: ...

n.shared.sites: ...

n.monomorphic.sites: ...

BD: ...

df: ...

D3: ...

Gmin: ...

df_bayes: ...

alpha_ABBA: ...

alpha_BABA: ...

beta_BBAA: ...

Bd_clr: ...

Bd_dir: ...

D.pval: ...

D.z: ...

`D.SE`: ...

`df.pval`: ...

`df.z`: ...

`df.SE`: ...

`P.Bd_clr`: ...

`RNDmin`: ...

Methods

detail.stats Several misc. statistics

diversity.stats Haplotype and nucleotide diversities

diversity.between Haplotype and nucleotide diversities

F_ST.stats.2 Snn from Hudson

F_ST.stats Fixation index

getBayes Get the input for BayeScanR

get.detail Get the results from the Detail module

get.codons Get information about the nature of codon changes

get.diversity Get diversities from the FST module

get.F_ST Get FST values from the FST module

get.linkage Get the values from the Linkage module

get.MKT Get Mcdonald-Kreitman values

getMS —

get.neutrality Get the values from the Neutrality module

get.status Status of calculations

get.sum.data Get some data observed from the alignments

linkage.stats Linkage disequilibrium

calc.R2 Linkage disequilibrium

mult.linkage.stats Linkage disequilibrium between regions

recomb.stats Recombination statistics

sweeps.stats Selective sweeps

Achaz.stats Achaz's statistics

get.recomb Get the values from the Recombination module

get.sweeps Get the values frome the Selective Sweep module

set.ref.positions Set the SNP positions

set.synnonsyn Verify synonymous positions

splitting.data Split the data into subsites

MKT MKT Test

neutrality.stats Neutrality statistics

popFSTN Internal function

get.biallelic.matrix Print the biallelic.matrix

set.populations Define the populations

set.outgroup Define the outgroup

get.individuals get the names/IDs of individuals

region.as.fasta Extract the region as a fasta file

show —

show.slots Show slots of the class GENOME

sliding.window.transform Transform a GENOME object into a new object suitable for sliding window analysis

usage —

PG_plot.biallelic.matrix Plot the biallelic matrix

introgression.stats Methods to measure archaic admixture

count.unknowns Calculates the frequencies of missing nucleotides

calc.fixed.shared Calculates the frequencies of missing nucleotides

set.filter SNP Filtering

weighted.jackknife weighted jackknife

Author(s)

Bastian Pfeifer

References

See the documentation for each module

Examples

```
#GENOME.class <- readData("Alignments")
#GENOME.class@n.sites
#GENOME.class@region.names
```

`get.biallelic.matrix-methods` *Get the biallelic matrix*

Description

This function returns the biallelic matrix of a specific region.

Usage

```
## S4 method for signature 'GENOME'
get.biallelic.matrix(object,region)
```

Arguments

object	An object of class "GENOME"
region	ID of the region

Value

Biallelic matrix
rows: names of individuals
columns: biallelic sites

Examples

```
# GENOME.class <- readData("\home\Alignments")
# get.biallelic.matrix(GENOME.class,7) # biallelic matrix of the 7th alignment
```

---

get.codons-methods *Detailed information about the nature of codon changes*

Description

This generic function returns some information about the codon changes resulting from the observed SNPs.

Usage

```
## S4 method for signature 'GENOME'
get.codons(object, regionID)
```

Arguments

object	an object of class "GENOME"
regionID	what region/alignment should be analyzed ?

Details

The slot GENOME.class@region.data@synonymous and GENOME.class@region.data@codons have to be set.
The data have to be read in with the correponding GFF file.
The function set.synnonsyn(..., save.codons=TRUE) sets the syn/nonsny sites in case of SNP data and stores the corresponding codon changes.

Value

The function get.codons returns a data.frame with the following information

1	Position of the SNPs
2	Major Codon
3	Minor Codon
4	Major amino acid
5	Minor amino acid
6	synonymous (TRUE/FALSE)
7	Polarity of the major amino acid
8	Polarity of the minor amino acid

Examples

```
# Alignments
# GENOME.class <- readData("FASTA",gffpath="GFF")
# get.codons(GENOME.class,1)
# SNP data
# GENOME.class <- readData("VCF",gffpath="GFF")
# GENOME.class <- set.synnonsyn(GENOME.class, ref.chr="ref.fas",save.codons=TRUE)
# get.codons(GENOME.class,1)
```

---

get.feature.names *Feature informations and GFF-attributes*

Description

Returns feature names and additional attributes for a given region

Usage

```
get.feature.names(object, gff.file, chr)
```

Arguments

object	An object of class GENOME
gff.file	The corresponding GFF file
chr	The chromosome/scaffold identifier

Details

The algorithm uses the information stored in `GENOME.class.split@region.names` to iterate over the GFF file and returns attribute plus feature informations for each given region.
Note, the functions splitting.data, split_data_into_GFF_attributes or sliding.window.transform should be performed prior to that.

The slot `region.names` must have the following form: "pos1 - pos2".

Value

The returned value is a character vector of length `length(GENOME.class.split@region.names)`

Examples

```
# GENOME.class <- readVCF("chr1.vcf.gz",1000,"1",1,100000)
# GENOME.class.split <- split_data_into_GFF_attributes(GENOME.class,"Homo_sapiens.GRCh37.73.gtf",
# "1", "gene_name")
# GENOME.class.split@region.names
# info <- get.feature.names(GENOME.class.split, gff.file="Homo_sapiens.GRCh37.73.gtf", chr="1")
# GENOME.class.split <- splitting.data(GENOME.class, subsites="gene")
# GENOME.class.split@region.names
# info <- get.feature.names(GENOME.class.split, gff.file="Homo_sapiens.GRCh37.73.gtf", chr="1")
```

`get.individuals-methods`

Print the names/IDs of individuals

Description

Extract the names/IDs of individuals.

Usage

```
## S4 method for signature 'GENOME'
get.individuals(object,region=FALSE)
```

Arguments

`object`	an object of class `"GENOME"`
`region`	a vector of regions. Default: ALL

Examples

```
# GENOME.class <- readData("\home\Alignments")
# get.individuals(GENOME.class)
```

`get.status-methods` *State of calculations*

Description

Some information about the definitions of populations and subsites.

Usage

```
## S4 method for signature 'GENOME'
get.status(object)
```

Arguments

`object`	An object of class `"GENOME"`

Examples

```
# get.status(GENOME.class)
```

`getBayes-methods` *Get values for BayeScanR*

Description

This function returns the values that are necessary to run BayeScanR.

Usage

```
## S4 method for signature 'GENOME'
getBayes(object,snps=FALSE)
```

Arguments

`object`	An object of class `"GENOME"`
`snps`	SNPs are considered seperately

Value

coming soon !

References

Foll M and OE Gaggiotti (2008). *A genome scan method to identify selected loci appropriate for both dominant and codominant markers: A Bayesian perspective.* Genetics 180: 977-993

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class <- F_ST.stats(GENOME.class,list(1:4,5:10))
# Bayes.input  <- getBayes(GENOME.class)
# Bayes.class  <- BayeScanR(Bayes.input)
```

`get_gff_info` *Annotation info*

Description

This function extracts annotation information from a GTF/GFF file.

Usage

```
get_gff_info(object=FALSE,gff.file,chr,position,feature=FALSE,extract.gene.names=FALSE)
```

Arguments

`object`	object of class `GENOME`
`gff.file`	basepath of the GTF/GFF file
`chr`	the chromosome
`position`	reference positions or region id (when `object` is specified)
`feature`	feature to search for in the gff-file. returns a list of positions
`extract.gene.names`	returns the gene names of the chromosome

Details

This function extracts annotation information from a GTF/GFF file.

Examples

```
# get_gff_info("Arabidopsis.gff",chr=1,200202)
# get_gff_info(GENOME.class,"Arabidopsis.gff",chr=1,position=3)
```

`gff_file` *GFF file (subdirectory "data")*

Description

A typical GFF file which should be stored in a folder (for example in "GFF"). This folder is the input for the `readData(...,gffpath="GFF")` function. The corresponding FASTA file is stored in the "data" subdirectory of the PopGenome package. It has to be stored in a folder with the SAME NAME as the GFF file (for example in "FASTA"). `readData("FASTA",gffpath="GFF")`

`GFF_split_into_scaffolds` *Split a GFF file into multiple scaffold-GFFs*

Description

This function splits a GFF file into multiple GFFs including data for exactly one scaffold each.

Usage

```
GFF_split_into_scaffolds(GFF.file, output.folder)
```

Arguments

`GFF.file`	the basepath of the GFF file
`output.folder`	name of the folder where the GFFs should be stored

Details

The algorithm splits the GFF into multiple scaffold based GFFs and
stores the files in a given folder. This folder can be used as an input for readData(,gffpath="")

Value

`TRUE`

Examples

```
# GFF_split_into_scaffolds("GFFfile.gff","scaffoldGFFs")
# test <- readData("scaffoldVCFs", format="VCF", gffpath="scaffoldGFFs")
```

`introgression.stats-methods` *Introgression statistics*

Description

A generic function to estimate archaic admixture.

Usage

```
## S4 method for signature 'GENOME'
introgression.stats(object,
subsites=FALSE,
do.D=TRUE,
do.df=TRUE,
keep.site.info=TRUE,
block.size=FALSE,
do.RNDmin=FALSE,
l.smooth=FALSE)
```

Arguments

`object`	An object of class `"GENOME"`
`subsites`	`"transitions"`: SNPs that are transitions. `"transversions"`: SNPs that are transversions. `"syn"`: synonymous sites. `"nonsyn"`: nonsynonymous sites. `"exon"`: SNPs in exon regions. `"intron"`: SNPs in intron regions. `"coding"`: SNPs in coding regions (CDS). `"utr"`: SNPs in UTR regions. `"gene"`: SNPs in genes. `"intergenic"` : SNPs in intergenic regions.
`do.D`	Pattersons D and Martin's f statistic
`do.df`	d-fraction
`keep.site.info`	keep site specific values (`GENOME.class@region.stats`)
`block.size`	the block size for jackknife
`do.RNDmin`	RNDmin (Rosenzweig, 2016)
`l.smooth`	laplace smoothing for Bd-fraction

Details

To perform the D and f statistic one needs to define 3 populations via the function `set.populations`, where the third population represent the archaic population. In addition, an outgroup is required and have to be set via the function `set.outgroup`. Here, only SNPs where the outgroup is monomorphic are considered. f is the fraction of the genome that is admixed [2].

Value

Returned value is a modified object of class `"GENOME"`

The following slots will be modified in the `"GENOME"` object

	Slot	Reference	Description
1.	`D`	[1;eq. 2]	Pattersons D statistic
2.	`f`	[2]	f statistic
3.	`df`	[3]	Bd-fraction
4.	`RNDmin`	[x]	RNDmin
5.	`D.z`	[x]	z values (jackknife)
6.	`D.pval`	[x]	P values (jackknife)
7.	`df.z`	[3]	z values (jackknife)
8.	`df.pval`	[3]	P values (jackknife)

References

[1] Durand, E. Y., Patterson, N. J., Reich, D., & Slatkin, M. (2011). *Testing for ancient admixture between closely related populations.*
Molecular Biology and Evolution, 28(8), 2239–2252. doi:10.1093/molbev/msr048
[2] Simon H Martin, Kanchon K Dasmahapatra, Nicola J Nadeau, et al. (2013). *Genome-wide evidence for speciation with gene flow in Heliconius butterflies.*
Genome Res. doi:10.1101/gr.159426.113
[3] Bastian Pfeifer and Durrell D. Kapan (2019). *Estimates of introgression as a function of pairwise distances.*
BMC Bioinformatics. https://doi.org/10.1186/s12859-019-2747-z

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class <- set.populations(GENOME.class,list(1:3,4:8,9:12))
# GENOME.class <- set.outgroup(GENOME.class,13)
# GENOME.class <- introgression.stats(GENOME.class, do.D=TRUE)
# show the result:
# GENOME.class@D
# GENOME.class@f
# GENOME.class <- introgression.stats(GENOME.class, do.df=TRUE)
# show the result:
# GENOME.class@df
```

jack.knife.transform *Jacknife Transformation*

Description

This generic function transforms an existing object of class "GENOME" into another object of class "GENOME", in which each region corresponds to the (JACKNIFE !) window. Each jacknife window will be excluded from the analyses and the calculation will be applied to the union of all other windows.

Usage

```
## S4 method for signature 'GENOME'
jack.knife.transform(object,
width=7, jump=5,
type=1,
start.pos=FALSE,end.pos=FALSE
)
```

Arguments

object	an object of class "GENOME"
width	window size. default:7
jump	jump size. default:5
type	1 scan only biallelic positions (SNPs), 2 scan the genome. default:1
start.pos	start position
end.pos	end position

Value

The function creates a transformed object of class "GENOME".

Note

This function currently is only available for SNP data formats. PopGenome will scan the data from position 1 to the last observed SNP if start or end-positions are not specified. This mechanism can also be applied to the splitting.data() function. Just set split.GENOME.class@jack.knife <- TRUE after splitting the data.

Examples

```
# GENOME.class      <- readData("...", format="VCF")
# jack.GENOME.class <- jack.knife.transform(GENOME.class,100,100)
# jack.GENOME.class <- neutrality.stats(jack.GENOME.class)
# jack.GENOME.class@Tajima.D
```

linkage.stats-methods *Linkage Disequilibrium*

Description

A generic function to calculate some linkage disequilibrium statistics.

Usage

```
## S4 method for signature 'GENOME'
linkage.stats(object,new.populations=FALSE,subsites=FALSE,detail=FALSE,
do.ZnS,do.WALL=TRUE)
## S4 method for signature 'GENOME'
get.linkage(object)
```

Arguments

object	An object of class "GENOME"
new.populations	list of populations. default=FALSE
subsites	"transitions": SNPs that are transitions. "transversions": SNPs that are transversions. "syn": synonymous sites. "nonsyn": nonsynonymous sites. "exon": SNPs in exon regions. "intron": SNPs in intron regions. "coding": SNPs in coding regions (CDS). "utr": SNPs in UTR regions. "gene": SNPs in genes. default:FALSE
detail	if you want to calculate some detailed statistics. This can be considerably slower! default:FALSE
do.ZnS	calculate ZnS, ZA and ZZ
do.WALL	calculate Wall's B/Q

Details

Note, the pairwise comparisons are computed via combn(n.snps,2).

Value

The return value is a modified object of class "GENOME"

The following slots will be modified in the "GENOME" object

	Slot	Reference	Description
1.	Wall.B	[2]	Wall B statistic (only adjacent positions are considered)
2.	Wall.Q	[2]	Wall Q statistic (only adjacent positions are considered)
3.	Kelly.Z_nS	[3]	Kelly Z_nS statistic (if detail==TRUE)
4.	Rozas.ZA	[1]	Rozas ZA statistic (adjacent positions, if detail==TRUE)
5.	Rozas.ZZ	[1]	Rozas ZZ statistic ($ZZ=ZA-Z_nS$, if detail==TRUE)

References

[1] Rozas, J., M.Gullaud, G.Blandin, and M.Aguade(2001). *DNA variation at the rp49 gene region of Drosophila simulans: evolutionary inferences from an unusual haplotype structure.* Genetics 158(3),1147-1155

[2] Wall, J.(1999). *Recombination and the power of statistical tests of neutrality.* Genet Res 74, 65-79

[3] Kelly,J.K. (1997). *A test of neutrality based on interlocus associations.* Genetics 146: 1197-1206

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class <- linkage.stats(GENOME.class)
# GENOME.class <- linkage.stats(GENOME.class,list(1:4,5:10),subsites="syn")
# GENOME.class <- linkage.stats(GENOME.class,list(c("seq1","seq5","seq3"),
# c("seq2","seq8")))
# GENOME.class <- linkage.stats(GENOME.class, detail=TRUE)
# show the result:
# get.linkage(GENOME.class)
# GENOME.class@Wall.B --> population specific view
# GENOME.class@region.stats
```

---

load.session *Loading a PopGenome session*

Description

This function loads a PopGenome session (more precisely: the corresponding "GENOME" object) from the current workspace.

Usage

```
load.session(folder)
```

Arguments

folder	name of the folder/object

Details

This function has to be used in the same workspace (folder) where the object of class "GENOME" was saved.

Value

An object of class "GENOME".

Examples

```
# GENOME.class <- readData("...\Alignments")
# save.session(GENOME.class,folder="GENOME.class")
# q()
# R
# library(PopGenome)
# load.session("GENOME.class")
```

MKT-methods *McDonald-Kreitman Test (McDonald \& Kreitman 1991)*

Description

This generic function calculates an approximate version of the McDonald-Kreitman Test.

Usage

```
## S4 method for signature 'GENOME'
MKT(object,
new.populations=FALSE,
do.fisher.test=FALSE,
fixed.threshold.fst=FALSE,
subsites=FALSE)

## S4 method for signature 'GENOME'
get.MKT(object)
```

Arguments

`object`	an object of class `"GENOME"`
`new.populations`	list of populations. default:`FALSE`
`do.fisher.test`	P-value calculation out of the Dn,Ds,Pn,Ps table
`fixed.threshold.fst`	Fixed threshold
`subsites`	Subsites

Details

This approximate version of the McDonald-Kreitman test assumes that the probability that two single nucleotide polymorphisms (SNPs) occur in the same codon is very small. Thus, only codons with a single SNP are examined.
If no gff-file was specified when the data was read in, it is assumed that the alignment is in the correct reading frame (starting at a first codon position).
The outgroup has to be defined as a population !

Value

The return value is a modified object of class "GENOME"

The following slots will be modified in the "GENOME" object

`MKT` a matrix which includes the following values:

	Columns	Description
1.	`P_nonsyn`	nonsynonymous sites
2.	`P_syn`	synonymous sites
3.	`D_nonsyn`	fixed nonsynonymous sites
4.	`D_syn`	fixed synonymous sites
5.	`neutrality.index`	$(P_nonsyn/P_syn)/(D_nonsyn/D_syn)$
6.	`alpha`	1-neutrality.index

References

McDonald, J. H.; Kreitman, M. (1991). *Adaptive protein evolution at the Adh locus in Drosophila.* Nature 351 (6328): 652-654

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class <- MKT(GENOME.class)
# GENOME.class <- MKT(GENOME.class,list(1:7,8:12))
# show the result:
# get.MKT(GENOME.class)
```

MS *Coalescent simulation with or without selection*

Description

This function uses Hudson's MS and Ewing's MSMS to compare simulated data with the observed data.

Usage

```
MS(GENO,niter=10,thetaID="user",params=FALSE,detail=FALSE,
neutrality=FALSE,linkage=FALSE,F_ST=FALSE,MSMS=FALSE,big.data=FALSE)
```

Arguments

GENO	an object of class "GENOME"
niter	number of samples per locus
thetaID	"Tajima","Watterson" or "user". default:"user"
neutrality	Calculate neutrality tests. default=FALSE
linkage	Calculate linkage disequilibrium. default=FALSE
F_ST	Calculate fixation index. default=FALSE
params	an object of class "test.params". see ?test.params
detail	detailed statistics. Note: slower computations! default=FALSE
MSMS	specify parameter for MSMS simulation with selection (has to be specified as a string)
big.data	if TRUE the ff-package is used

Details

You can choose different mutation rate estimators to generate simulation data. When thetaID="user", you have to define the theta values in an object of class "test.params". The "test.params" class can also be used to specify some additional parameter like migration and/or recombination rates... (?test.params).

Please read the MSMS documentation for the correct use of coalescent simulations to assess statistical significance.

Value

The function creates an object of class "cs.stats"

Note

The executable file ms has to be stored in the current workspace.
If you want to use the MSMS application, put the msms folder including the corresponding executable files in the current workspace.
Both programs can be obtained from their websites (see references).

References

Hudson, R. R. (2002). *Generating samples under a Wright-Fisher neutral model of genetic variation.* Bioinformatics 18: 337-338

Gregory Ewing and Joachim Hermisson, *MSMS: A Coalescent Simulation Program Including Recombination, Demographic Structure, and Selection at a Single Locus.* Bioinformatics 2010, doi: 10.1093/bioinformatics/btq322

Examples

```
# GENOME.class <- readData("...\Alignments")
# GENOME.class <- neutrality.stats(GENOME.class,list(1:6))
# MS.class <- MS(GENOME.class,thetaID="Tajima",neutrality=TRUE)
# MS.class <- MS(GENOME.class,thetaID="Tajima",neutrality=TRUE,
#                MSMS="-N 1000 -SAA 200 -SaA 100 -SF 1e-2")
# MS.class
# MS.class@obs.val
# MS.class@locus[[1]]
```

MS_getStats *Get the simulated MS/MSMS statistics*

Description

This function extracts the simulated values from the class cs.stats

Usage

```
MS_getStats(object,locus=1,population=1)
```

Arguments

object	object of class "cs.stats"
locus	the locus ID
population	the population ID

Value

The return value is a matrix containing the simulation results of different statistical tests.
(see MS())

Examples

```
# GENOME.class <- readData("...\Alignments")
# GENOME.class <- neutrality.stats(GENOME.class)
# ms <- MS(GENOME.class,thetaID="Tajima",neutrality=TRUE)
# MS_getStats(ms)
```

---

mult.linkage.stats-methods

Multilocus linkage statistics

Description

This generic function calculates the linkage disequilibrium between regions.

Usage

```
## S4 method for signature 'GENOME'
mult.linkage.stats(object,lower.bound=0,upper.bound=1,pairs=FALSE)
```

Arguments

object	an object of class "GENOME"
lower.bound	sites with minor-allele-frequency>=lower.bound are considered
upper.bound	sites with minor-allele-frequency<=upper.bound are considered
pairs	permutation matrix of pairwise comparisons

Details

pairs is a matrix. Each column contains the pairwise comparison region IDs.

1 1
2 3

compares region 1 with 2, and region 1 with 3.

Value

The return value is a modified object of class "GENOME"

The following slots will be modified in the "GENOME" object

---

mult.Linkage Some linkage statistics for each pair of regions (R2, P-value, Distance)

The Fisher-Exact-Test is used to calculate the P-values.

Examples

```
# GENOME.class <- readData("...\Alignments")
# GENOME.class
# GENOME.class <- mult.linkage.stats(GENOME.class)
# show the result:
# GENOME.class@mult.Linkage
```

---

neutrality.stats-methods

Neutrality Statistics

Description

This generic function calculates some neutrality statistics.

Usage

```
## S4 method for signature 'GENOME'
neutrality.stats(object,new.populations=FALSE,new.outgroup=FALSE,
subsites=FALSE,detail=FALSE, FAST=FALSE, do.R2=FALSE)
## S4 method for signature 'GENOME'
get.neutrality(object,theta=FALSE,stats=TRUE)
```

Arguments

object	an object of class "GENOME"
new.populations	list of populations. default:FALSE
new.outgroup	vector of outgroup sequences. default:FALSE
subsites	"transitions": SNPs that are transitions. "transversions": SNPs that are transversions. "syn": synonymous sites. "nonsyn": non-synonymous sites. "exon": SNPs in exon regions. "intron": SNPs in intron regions. "coding": SNPs in coding regions (CDS). "utr": SNPs in UTR regions. "gene": SNPs in genes. default:FALSE
detail	default:FALSE, TRUE for some detailed statistics. Note: slows down calculations!
FAST	Fast computation. only works if there is no outgroup defined.
do.R2	Ramos-Onsins' & Rozas' R2
stats	show the results of each statistic. default:TRUE
theta	show the theta values. default:FALSE

Value

The return value is a modified object of class "GENOME"

The following slots will be modified in the "GENOME" object

	Slot	Reference	Description
1.	n.segregating.sites		Total number of segregating sites
2.	Tajima.D	[1]	Tajima's' D statistic 1989
3.	Fu.Li.F	[3]	Fu & Li's' F* statistic 1993
4.	Fu.Li.D	[3]	Fu & Li's D* statistic 1993
5.	Fay.Wu.H	[6]	Fay & Wu's H statistic 2000
6.	Zeng.E	[7]	Zeng's E statistic 2006
7.	Strobeck.S	[5]	Strobeck's S statistic 1987 (if detail==TRUE)
8.	Fu.F_S	[4]	Fu's F$_S$ statistic 1997 (if detail==TRUE)
9.	Rozas.R_2	[2]	Ramos-Onsins' & Rozas' R_2 statistic 2002
10.	theta_Tajima	[1]	
11.	theta_Watterson		
12.	theta_Fu.Li	[3]	
13.	theta_Achaz.Watterson		
14.	theta_Achaz.Tajima		
15.	theta_Fay.Wu	[6]	

16. `theta_Zeng` [7]

References

[1] Tajima, F.(1989) *Statistical Method for Testing the Neutral Mutation Hypothesis by DNA Polymorphism.* Genetics, 123(3): 585-595.

[2] Ramos-Onsins, S.E. and J.Rozas (2002). *Statistical Properties of New Neutrality Tests Against Population Growth.* Mol.Biol.Evol.19(12),2092-2100

[3] Fu, Y.X. and W.H.Li (1993). *Statistical Tests of Neutrality of Mutations.* Genetics 133(3),693-709

[4] Fu, Y.-X.(1997). *Statistical Tests of Neutrality of mutations against population growth, hitchhiking and background selection.* Genetics 147(2),915-925.

[5] Strobeck, C. (1987). *Average number of nucleotide differences in a sample from a single subpopulation: a test for population subdivision.* Genetics 117, 149-153

[6] Fay, J.C. and C.-I. Wu (2000). *Hitchhiking under positive Darwinian selection.* Genetics 155 (3),1405-1413

[7] Zeng, K., Y.-X. Fu, S. Shi, and C.-I. Wu (2006). *Statistical tests for detecting positive selection by utilizing high-frequency variants.* Genetics 174, 1431-1439

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class <- neutrality.stats(GENOME.class, FAST=TRUE)
# GENOME.class <- neutrality.stats(GENOME.class,list(1:4,5:10),subsites="syn")
# GENOME.class <- neutrality.stats(GENOME.class,list(c("seq1","seq5","seq3"),
# c("seq2","seq8")))
# GENOME.class <- neutrality.stats(GENOME.class,detail=TRUE)
# show the result:
# get.neutrality(GENOME.class)
# GENOME.class@Tajima.D --> population specific view
# detail = TRUE
# GENOME.class@region.stats
```

`PG_plot.biallelic.matrix-methods`

Plot the biallelic matrix

Description

This function plots the biallelic matrix of a specific region.

Usage

```
## S4 method for signature 'GENOME'
PG_plot.biallelic.matrix(object,region, ind.names = FALSE , cex.axis = 0.5,
title="")
```

Arguments

object	object of class "GENOME"
region	the region ID
ind.names	individual names/IDs. default:ALL
cex.axis	size of text (y-axis)
title	Title of the plot

Examples

```
# GENOME.class <- readData("...\Alignments")
# PG_plot.biallelic.matrix(GENOME.class, region = 1)
```

PopGenome *PopGenome*

Description

R-package for Population genetic & genomic analyses

Details

Index:

```
F_ST.stats              Fixation index
diversity.stats         Diversities
MKT                     McDonald & Kreitman test
MS                      Coalescent simulations
detail.stats            Several misc. statistics
linkage.stats           Linkage disequilibrium
neutrality.stats        Neutrality statistics
readData                Reading alignments and calculating summary data
readSNP           Read data in .SNP format (e.g., from the 1001 Arabidopsis Genomes project)
```

```
readVCF             Read data in VCF format (e.g., from the 1000 human Genomes project)
readHapMap                Read data in HapMap format
sliding.window.transform Sliding window transformation
splitting.data            Split data into subsites
test.params               Set parameters for coalescent simulations.
```

Author(s)

Bastian Pfeifer Maintainer: Bastian Pfeifer <Bastian.Pfeifer@uni-duesseldorf.de>

See Also

?readData readData

Examples

```
# GENOME.class <- readData("...\Alignments")
# GENOME.class <- neutrality.stats(GENOME.class)
# values       <- get.neutrality(GENOME.class)
# GENOME.class <- F_ST.stats(GENOME.class,list(1:5,6:10))
# values       <- get.F_ST(GENOME.class)
```

PopGplot *Smoothed line-plot for multiple populations*

Description

This function plots values with smoothed lines using spline interpolation.

Usage

```
PopGplot(values,colors=FALSE,span=0.1,ylab="",xlab="",
ylim=c(min(values,na.rm=TRUE),max(values,na.rm=TRUE)))
```

Arguments

values	the statistical values (matrix); columns=populations
colors	the colors for each population (character vector)
span	the degree of smoothing
ylab	a title for the y axis
xlab	a title for the x axis
ylim	ranges for the y axis

Examples

```
# GENOME.class <- readSNP("Arabidopsis",CHR=1)
# GENOME.class.slide <- sliding.window.transform(test,1000,1000)
# GENOME.class.slide <- diversity.stats(GENOME.class.slide)
# values <- GENOME.class.slide@nuc.divesity.within
# PopGplot(values)
```

---

read.big.fasta *Reading large FASTA alignments*

Description

This function splits FASTA alignments that are too large to fit into the computer memory into chunks.

Usage

```
read.big.fasta(filename,populations=FALSE,outgroup=FALSE,window=2000,
               SNP.DATA=FALSE,include.unknown=FALSE,
               parallized=FALSE,FAST=FALSE,big.data=TRUE)
```

Arguments

filename	the basepath of the FASTA alignment
outgroup	vector of outgroup sequences
populations	list of populations
window	chunk size: number of columns/nucleotide sites
SNP.DATA	should be switched to TRUE if you use SNP data in alignment format
include.unknown	include unknown positions in the biallelic.matrix
parallized	Use parallel computations to speed up the reading - works only on UNIX systems!
FAST	Fast computation. see readData()
big.data	use the ff-package

Details

The algorithm reads the data for each individual and stores the information on disk. The data can be analyzed as regions of the defined window size, or can be concatenated in the PopGenome framework via the function concatenate.regions. This function should only be used when the FASTA file does not fit into the RAM; else, use the function readData.

Value

The function creates an object of class "GENOME"

The following slots will be filled in the "GENOME" object

	Slot	Description
1.	n.sites	total number of sites
2.	n.biallelic.sites	number of biallelic sites
3.	region.names	names of regions
4.	region.data	some detailed information about the data

Examples

```
# GENOME.class <- read.big.fasta("Alignment.fas", big.data=TRUE)
# GENOME.class
# GENOME.class@region.names
# CON <- concatenate.regions(GENOME.class)
# CON@region.data@biallelic.sites
# GENOME.class.slide <- sliding.window.transform(GENOME.class,100,100)
# GENOME.class <- neutrality.stats(GENOME.class,FAST=TRUE)
# show the result:
# get.sum.data(GENOME.class)
# GENOME.class@region.data
```

---

readData *Read alignments and calculate summary data*

Description

This function reads alignments/SNP data in several formats and calculates some summary data.

Usage

```
readData(path,populations=FALSE,outgroup=FALSE,include.unknown=FALSE,
         gffpath=FALSE,format="fasta",parallized=FALSE,
         progress_bar_switch=TRUE, FAST=FALSE,big.data=FALSE,
         SNP.DATA=FALSE
        )

## S4 method for signature 'GENOME'
get.sum.data(object)
```

Arguments

object	object of class "GENOME"
path	the basepath (folder) of the alignments
outgroup	vector of outgroup sequences
include.unknown	if positions with unknown nucleotides should be considered.
populations	list of populations. default:FALSE
gffpath	the basepath (folder) of the corresponding GFF-files. default:FALSE
format	data formats. "fasta" is default. See details !
parallized	parallel processing to accelerate the reading process. See details !
progress_bar_switch	progress_bar
FAST	fast computation. See details !
big.data	use the ff-package
SNP.DATA	important for reference positions; should be TRUE if you use SNP-data in alignment format

Details

All data (alignments or SNP-files) have to be stored in one folder. The folder is the input of this function. If no GFF file (which also have to be stored in a folder) is specified, an alignment in the correct reading frame (starting at a first codon position) is expected.
Otherwise synonymous and non-synonymous positions are not identified correctly.

Note:
The GFF-files have to be EXACTLY the same names (without any extensions like .fas or .gff) as the files storing the nucleotide data to ensure correct matching

format:
"fasta","nexus","phylip",
"MAF","MEGA"
"HapMap","VCF"
"RData"
Valid nucleotides are T,t,U,u,G,g,A,a,C,c,N,n,-

parallized:
- will speed up calculations if you use a very large amount of alignments

FAST:
- will not classify synonymous/non-synonymous SNPs directly
- fast computation (via compiled C code) of biallelic matrix, biallelic sites, transversions/transitions

and biallelic substitutions
- can be switched to TRUE in case of SNP data without loss of information

big.data:
- use the ff-package
- ff mechanism is used for biallelic.matrix and GFF/GTF information
- is automatically activated for readVCF or readSNP
- Note! you should set this to TRUE if you use big chunks of data
and you want to later concatenate them in the PopGenome framework
(for example: sliding windows of the whole dataset).

SNP.DATA:
- should be switched to TRUE if you use SNP-data in alignment format.
- the corresponding SNP positions can be set via set.ref.positions

Value

The function creates an object of class "GENOME"

The following slots will be filled in the "GENOME" object

	Slot	Description
1.	n.sites	total number of sites
2.	n.biallelic.sites	number of biallelic sites
3.	n.gaps	number of sites with gaps
4.	n.unknowns	number of sites with unknown nucleotides
5.	n.valid.sites	number of valid sites
6.	n.polyallelic.sites	number of sites with >2 nucleotides
7.	trans.transv.ratio	transition/transversion ratio of biallelic sites
8.	region.names	names of regions
9.	region.data	some detailed information about the data read

Examples

```
# GENOME.class <- readData("...\Alignments", FAST=TRUE)
# GENOME.class <- readData("VCF", format="VCF")
# Note, "Alignments" and "VCF" are folders !
# GENOME.class@region.names
# GENOME.class <- readData("...\Alignments", big.data=TRUE)
# object.size(GENOME.class)
# GENOME.class <- readData("...\Alignments",gffpath="...\Alignments_GFF")
# GENOME.class
```

```
# show the result:
# get.sum.data(GENOME.class)
# GENOME.class@region.data
```

---

readHapMap *Read SNP data from the HapMap consortium*

Description

This function reads HapMap data.

Usage

```
readHapMap(folder,hap_gffpath,populations=FALSE,outgroup=FALSE)
```

Arguments

folder	the basepath of the variant_calls
hap_gffpath	the basepath of the corresponding GFF files. Note! The HapMap GFF file does not contain information about subsites. see details!
populations	list of populations
outgroup	vector of outgroup sequences

Details

PopGenome reads the GFF file distributed on the HapMap plattform only to verify the reference positions of the chromosomes. In the next release, this function will also handle GFF/GTF files to get information about subsites (exons, introns, ...). The input folder should include the files of different individuals for one chromosome. This facilitates FST calculations of the HapMap data. See also readData("...", format="HapMap") which can read the files of single populations directly.

Value

The function creates an object of class "GENOME"

The following slots will be filled in the "GENOME" object

	Slot	Description
1.	n.sites	total number of sites
2.	n.biallelic.sites	number of biallelic sites
3.	region.data	some detailed information about the data read

Examples

```
# GENOME.class <- readHapMap("...\HapMapData")
# GENOME.class
# show the result:
# get.sum.data(GENOME.class)
# GENOME.class@region.data
```

---

readMS *Read output data from MS and MSMS*

Description

This function reads data produced from the coalescent simulation programs MS (Hudson, 2002) and MSMS (Greg, 2010)

Usage

```
readMS(file,big.data=FALSE)
```

Arguments

file	the basepath of the MS/MSMS output
big.data	The ff package is used

Value

An object of class GENOME

References

Hudson, R. R. (2002). *Generating samples under a Wright-Fisher neutral model of genetic variation.* Bioinformatics 18: 337-338

Gregory Ewing and Joachim Hermisson, *MSMS: A Coalescent Simulation Program Including Recombination, Demographic Structure, and Selection at a Single Locus.* Bioinformatics 2010, doi: 10.1093/bioinformatics/btq322

Examples

```
# GENOME.class <- readMS("ms.output.txt")
# GENOME.class@region.names
```

readSNP *Read data in .SNP format*

Description

This function reads data in .SNP (quality_variant) format, as distributed by the 1001 Genomes project (Arabidopsis).

Usage

```
readSNP(folder,populations=FALSE,outgroup=FALSE,gffpath=FALSE,
CHR=FALSE,ref.chr=FALSE,snp.window.size=FALSE,
parallized=FALSE,ffpackagebool=TRUE,
include.unknown=FALSE
)
```

Arguments

folder	the basepath of the variant_calls
outgroup	vector of outgroup sequences
populations	list of populations
gffpath	the corresponding GFF file
CHR	which chromosome ?, default: all chromosomes
ref.chr	reference chromosome (to classify synonymous/non-synonymous positions)
snp.window.size	scan SNP chunks
parallized	multicore computation
ffpackagebool	use the ff-package to save memory space. (slower)
include.unknown	include positions with unknown nucleotides

Details

The ff-package we use to store the SNP information limits the data size to
individuals * (number of SNPs) <= .Machine$integer.max
The text files containing the SNP information of each individual have to be stored in one folder.
The slots transitions, biallelic.sites, and biallelic.substitutions of the class "regions.data" will be filled.
At this time, if a GFF/GTF is used the data should be organized in a way that
the "CHR" is a numerical value. The prefix "Chr" or "chr" is also supported.

Value

The function creates an object of class "GENOME"

Following Slots will be filled in the "GENOME" object

	Slot	Description
1.	n.sites	total number of sites
2.	n.biallelic.sites	number of biallelic sites
3.	region.data	some detailed information about the data read
4.	region.names	names of regions

Examples

```
# GENOME.class <- readSNP("...\SNPData")
# GENOME.class <- readSNP("...\SNPData", CHR=1)
# GENOME.class <- readSNP("...\SNPData", CHR=1, gffpath="Gff_file.gff")
# GENOME.class
# GENOME.class <- neutrality.stats(GENOME.class,FAST=TRUE)
# show the result:
# get.sum.data(GENOME.class)
# GENOME.class@region.data
```

---

readVCF *Read SNP data in tabixed VCF format*

Description

This function reads tabixed VCF-files, as distributed from the 1000 Genomes project (human).

Usage

```
readVCF(filename, numcols, tid, frompos, topos,
        samplenames=NA, gffpath = FALSE, include.unknown=FALSE, approx=FALSE,
out="", parallel=FALSE)
```

Arguments

filename	the corresponding tabixed VCF-file
numcols	number of SNPs that should be read in as a chunk

tid	which chromosome ? (character)
frompos	start of the region
topos	end of the region
samplenames	a vector of individuals
gffpath	the corresponding GFF file
include.unknown	includ positions with unknown/missing nucleotides
approx	see details !
out	a folder suffix where the temporary files should be saved
parallel	parallel computation using mclapply

Details

The readVCF function expects a tabixed VCF file with a diploid GT field.
In case of haploid data, the GT field has to be transformed to a pseudo-diploid field (such as 0 -> 0|0). An alternative is to use readData(..., format="VCF"), which can read non-tabixed haploid and any kind of polyploid VCFs directly.
When `approx=TRUE`, the algorithm will apply a logical OR to the GT-field: (0|0=0,1|0=1,0|1=1,1|1=1). Note, this is an approximation for diploid data, which will speed up calculations. In case of haploid data, `approx` should be switched to `TRUE`.
If `approx=FALSE`, the full diploid information will be considered.
The ff-package PopGenome uses to store the SNP information limits total data size to individuals * (number of SNPs) <= .Machine$integer.max
In case of very large data sets, the bigmemory package will be used;
this will slow down calculations (e.g. this package have to be installed first !!!).
Use the function `vcf_handle <-.Call("VCF_open",filename)`
to open a VCF-file and `.Call("VCF_getSampleNames",vcf_handle)`
to get and define the individuals which should be considered in the analysis.
See also readData(..., format="VCF") !

Value

The function creates an object of class `"GENOME"`

The following slots will be filled in the `"GENOME"` object

	Slot	Description
1.	`n.sites`	total number of sites
2.	`n.biallelic.sites`	number of biallelic sites
3.	`region.data`	some detailed information about the data read
4.	`region.names`	names of regions

Examples

```
# GENOME.class <- readVCF("...\chr1.vcf.gz", 1000, "1", 1, 100000)
# GENOME.class
# GENOME.class@region.names
# GENOME.class <- neutrality.stats(GENOME.class,FAST=TRUE)
# show the result:
# get.sum.data(GENOME.class)
# GENOME.class@region.data
```

---

recomb.stats-methods *Recombination statistics*

Description

This generic function calculates the Four-Gamete test (Hudson 1985).

Usage

```
## S4 method for signature 'GENOME'
recomb.stats(object,new.populations=FALSE,subsites=FALSE)
## S4 method for signature 'GENOME'
get.recomb(object)
```

Arguments

object	an object of class "GENOME"
new.populations	list of populations. default:FALSE
subsites	"transitions": SNPs that are transitions. "transversions": SNPs that are transversions. "syn": synonymous sites. "nonsyn": nonsynonymous sites. "exon": SNPs in exon regions. "intron": SNPs in intron regions. "coding": SNPs in coding regions (CDS). "utr": SNPs in UTR regions. "gene": SNPs in genes. default:FALSE

Value

The return value is a modified object of class "GENOME"

The following slots will be modified in the "GENOME" object

Hudson.RM Four-gamete test

References

Hudson, R. K. (1985). *Statistical Properties of the Number of Recombination Events in the History of a Sample of DNA Sequences* Genetics 111 (1): 147-164.

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class <- recomb.stats(GENOME.class)
# GENOME.class <- recomb.stats(GENOME.class,list(1:7,8:12))
# show the result:
# recomb.values <- get.recomb(GENOME.class)
# recomb.values[[1]] # first population !
# GENOME.class@region.stats@Hudson.RM
```

---

region.as.fasta-methods

Extract a region and write it to a FASTA file

Description

This generic function writes a FASTA file of the observed biallelic positions to the current workspace.

Usage

```
## S4 method for signature 'GENOME'
region.as.fasta(object,region.id=FALSE,filename=FALSE,type=1,ref.chr=FALSE)
```

Arguments

object	an object of class "GENOME"
region.id	region of the genome
filename	name of the FASTA file
type	1: extract SNPs; 2: extract all nucleotides
ref.chr	reference sequence

Details

In case of type=2 we recommend to use the function splitting.data(,positions=list(...), type=2) before and apply the region.as.fasta() to this splitted object afterwards. The type=1 method will write a FASTA file including only the biallelic.sites.
region.id is the the region number specified in the PopGenome class GENOME.

Examples

```
#GENOME.class       <- readSNP("Arabidopsis",CHR=1)
# split the data into the genmic positions 100 to 2000
#GENOME.class.split <- splitting.data(GENOME.class, positions=list(100:2000),type=2)
#GENOME.class.split@region.names
#region.as.fasta(GENOME.class.split,1,"my_fasta_file.fas",type=2, ref.chr="chrom1.fas")
```

---

save.session *Save the "GENOME" object of a PopGenome session*

Description

This function saves the "GENOME" object of a PopGenome session to the current workspace. The object can be loaded again with load.session().

Usage

```
save.session(object,folder)
```

Arguments

object	object of class "GENOME"
folder	name of the folder/object

Details

Saving R and ff-objects created by the ff-package in a folder.

Examples

```
# GENOME.class <- readData("...\Alignments")
# save.session(GENOME.class,"GENOME.class")
# load.session("GENOME.class")
```

`set.filter-methods` *Setting filter to the analysis*

Description

A generic function to set filter regarding e.g missing data.

Usage

```
## S4 method for signature 'GENOME'
set.filter(object,
missing.freqs=TRUE,
minor.freqs=FALSE,
maf.lower.bound=0,
maf.upper.bound=1,
miss.lower.bound=0,
miss.upper.bound=1)
```

Arguments

`object`	An object of class `"GENOME"`
`missing.freqs`	Set filter for missing data
`minor.freqs`	Set filter for the MAF
`maf.lower.bound`	frequency of the MAF
`maf.upper.bound`	...
`miss.lower.bound`	frequency of the missing freq.
`miss.upper.bound`	...

Details

This function sets the slot `region.data@included`.

Value

Returned value is a modified object of class "GENOME"

The following slots will be modified in the "GENOME" object

	Slot	Reference	Description
1.	region.data@included	[x]	...

References

[x]

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class <- set.filter(GENOME.class, missing.freqs=TRUE,
# miss.lower.bound=0, miss.upper.bound=0.2)
# now apply any statistic to the filtered data set.
# GENOME.class <- calc.fixed.shared(GENOME.class, subsites="included")
```

---

set.outgroup-methods *Define an outgroup*

Description

This generic function defines the outgroup by matching the specified vector against each region.

Usage

```
## S4 method for signature 'GENOME'
set.outgroup(object,new.outgroup=FALSE, diploid=FALSE)
```

Arguments

object	an object of class "GENOME"
new.outgroup	a vector of outgroup individuals
diploid	if diploid data is present

Examples

```
# GENOME.class <- readData("\home\Alignments")
# outgroup <- c("seq1","seq2")
# GENOME.class <- set.outgroup(GENOME.class,new.outgroup=outgroup)
# GENOME.class <- neutrality.stats(GENOME.class)
```

`set.populations-methods`

Define populations

Description

This generic function defines the populations.
Using this function obviates the need to specify the populations for each calculation separately.
The populations can be set differently for different PopGenome modules by applying the function between module calls.

Usage

```
## S4 method for signature 'GENOME'
set.populations(object,new.populations=FALSE, diploid=FALSE,
triploid=FALSE,tetraploid=FALSE)
```

Arguments

`object`	an object of class `"GENOME"`
`new.populations`	list of populations. default:`FALSE`
`diploid`	if diploid data is present
`triploid`	if triploid data is present
`tetraploid`	if tetraploid data is present

Examples

```
# GENOME.class <- readData("\home\Alignments")
# pop.1 <- c("seq1","seq2")
# pop.2 <- c("seq3","seq4","seq1")
# GENOME.class <- set.populations(GENOME.class,list(pop.1,pop.2))
# GENOME.class@region.data@populations2
# GENOME.class <- neutrality.stats(GENOME.class)
```

set.ref.positions-methods *Set reference positions for SNP data*

Description

This generic function sets the positions of the SNP data. Should be used if you use alignment formats to store SNP data (i.e., data restricted to the polymorphic positions).

Usage

```
## S4 method for signature 'GENOME'
set.ref.positions(object, positions)
```

Arguments

object	an object of class "GENOME"
positions	a list of reference positions

Value

returned value is a modified object of class "GENOME"

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class <- set.ref.positions(GENOME.class,list(c(1000,2001,3000),
#                 c(3200,12000)))
```

set.synnonsyn-methods *Set synonymous positions for SNP data*

Description

This generic function classifies the observed biallelic positions read from SNP data files into synonymous and non-synonymous SNPs.

Usage

```
## S4 method for signature 'GENOME'
set.synnonsyn(object,ref.chr,save.codons=FALSE)
```

Arguments

object	an object of class "GENOME"
ref.chr	the reference chromosome in FASTA format
save.codons	save codon changes

Value

The return value is a modified object of class "GENOME" storing
syn/nonsyn informations in the slot GENOME.class@region.data@synonymous
for each SNP. (1=synonymous,0=non-synonymous)
When save.codons is TRUE the SNP related codon changes
are saved in the corresponding slot GENOME.class@region.data@codons.
(see also get.codons(), codontable() and codonise64())

Note

The data has to be read in with a corresponding GFF/GTF file (CDS fields must be specified); otherwise a correct classification is not possible. The set.synnonsyn() function does not work for splitted objects e.g produced via sliding.window.transform() or splitting.data(). Note, transcripts which are in the same CDS region but have different reading frames are
are not specified correctly. PopGenome can also handle coding regions on reverse strands. We have used the program SNPeff to validate our results.

Examples

```
# GENOME.class <- readData("VCF",format="VCF",gffpath="GFF.Folder")
# GENOME.class <- set.synnonsyn(GENOME.class,ref.chr="ref.fas")
# GENOME.class@region.data@synonymous
```

---

show.slots-methods *Show Slots of class GENOME*

Description

coming soon ...

Methods

object = "GENOME" coming soon ...

Examples

```
# show.slots(GENOME.class)
```

`sliding.window.transform-methods`
Sliding Window Transformation

Description

This generic function transforms an existing object of class `"GENOME"` into another object of class `"GENOME"`, in which each region corresponds to one window. This allows to apply the full spectrum of PopGenome methods to sliding window data.

Usage

```
## S4 method for signature 'GENOME'
sliding.window.transform(object,
width=7, jump=5,
type=1,
start.pos=FALSE,end.pos=FALSE,
whole.data=TRUE
)
```

Arguments

`object`	an object of class `"GENOME"`
`width`	window size. default:`7`
`jump`	jump size. default:`5`
`type`	`1` scan only biallelic positions (SNPs), `2` scan the genome. default:`1`
`start.pos`	start position
`end.pos`	end position
`whole.data`	scan the complete data by concatenating the regions in "object". If FALSE, each region is scanned seperately.

Value

The function creates a transformed object of class `"GENOME"`.

Note

If you want to scan regions seperately (whole.data=FALSE), you may not use the big.data option in the readData function. PopGenome will scan the data from position 1 to the last observed SNP if start or end-positions are not specified.

Examples

```
# GENOME.class       <- readData("...\Alignments")
# slide.GENOME.class <- sliding.window.transform(GENOME.class)
# slide.GENOME.class <- sliding.window.transform(GENOME.class,100,100)
# slide.GENOME.class <- neutrality.stats(slide.GENOME.class)
# slide.GENOME.class@region.names
# values             <- get.neutrality(slide.GENOME.class)
# GENOME.class       <- readSNP("Arabidopsis", CHR=1)
# GENOME.slide       <- sliding.window.transform(GENOME.split, 10000, 10000, type=2,
# start.pos=10000000, end.pos=12000000)
# GENOME.slide@region.names
```

`snp_file` *.SNP file (variant call data from 1001 Arabidopsis Genomes project)*

Description

A .SNP file stored in the directory "data" of the PopGenome package. The file contains variant calls for exactly one individual. Put all files (individuals of interest) into one folder (for example "SNP"). `readSNP("SNP",CHR=1)`

`splitting.data-methods`
Split data into subsites

Description

This generic function splits the data into subsites, if GFF/GTF information is present or if positions are defined accordingly.

Usage

```
## S4 method for signature 'GENOME'
splitting.data(object,subsites=FALSE,positions=FALSE,type=1,
          whole.data=TRUE)
```

Arguments

object	an object of class "GENOME"
positions	list of positions
subsites	"exon": SNPs in exon regions. "intron": SNPs in intron regions. "coding": SNPs in coding regions (CDS). "utr": SNPs in UTR regions. "gene": SNPs in genes.
type	1: SNP positions 2: Genome positions
whole.data	Scan the whole data by concatenating the regions. If FALSE, the regions are scanned seperately

Details

Note, if whole.data=FALSE data with n.biallelic.sites==0 should be removed.

Value

The return value is a modified object of class "GENOME".

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class
# GENOME.class.split <- splitting.data(GENOME.class,subsites="exon")
# GENOME.class.split@region.names
# GENOME.class.split <- splitting.data(GENOME.class,positions=list(1:7,8:12))
# GENOME.class.split <- splitting.data(GENOME.class,
# positions=list(2000:3000,12000:13000),type=2)
# GENOME.class.split
```

---

split_data_into_GFF_attributes

Split the data into GFF attributes

Description

Splits the data into GFF attributes defined by the user.

Usage

```
split_data_into_GFF_attributes(object, gff.file, chr, attribute)
```

Arguments

object	An object of class GENOME
gff.file	The corresponding GFF file
chr	The chromosome/scaffold identifier
attribute	The attribute to use for splitting

Details

The algorithm splits the data into attributes.
An attribute can be "gene_name", "Parent" or just a single
gene name like "geneXYZ".

Value

The returned value is an object of class "GENOME"
See GENOME.class.split@region.names and GENOME.class.split@region.names
after splitting the data.

Examples

```
# GENOME.class <- readVCF("chr1.vcf.gz",1000,"1",1,100000)
# GENOME.class.split <- split_data_into_GFF_attributes(GENOME.class,"Homo_sapiens.GRCh37.73.gtf",
# "1", "gene_name")
# GENOME.class.split@region.names
# GENOME.class.split@feature.names
```

---

split_data_into_GFF_features

Split the data into GFF features

Description

Splits the data into GFF feautures defined by the user.

Usage

```
split_data_into_GFF_features(object, gff.file, chr, feature)
```

Arguments

object	An object of class GENOME
gff.file	The corresponding GFF file
chr	The chromosome/scaffold identifier
feature	The feature used for splitting

Details

The algorithm splits the data into features.
A feature can be "gene", "exon" etc.
depending on what is specified in the GFF3 file.

Value

The returned value is an object of class "GENOME"
See GENOME.class.split@region.names and GENOME.class.split@region.names
after splitting the data.

Examples

```
# GENOME.class <- readVCF("chr1.vcf.gz",1000,"1",1,100000)
# GENOME.class.split <- split_data_into_GFF_features(GENOME.class,"Homo_sapiens.GRCh37.73.gtf",
# "1", "gene")
# GENOME.class.split@region.names
```

---

sweeps.stats-methods *Selective Sweeps*

Description

This module calculates some statistics to detect selective sweeps.

Usage

```
## S4 method for signature 'GENOME'
sweeps.stats(object,new.populations=FALSE,subsites=FALSE,
 freq.table=FALSE, FST=FALSE)
## S4 method for signature 'GENOME'
get.sweeps(object)
```

Arguments

object	an object of class "GENOME"
new.populations	list of populations. default:FALSE
subsites	"transitions": SNPs that are transitions. "transversions": SNPs that are transversions. "syn": synonymous sites. "nonsyn": non-synonymous sites. "exon": SNPs in exon regions. "intron": SNPs in intron regions. "coding": SNPs in coding regions (CDS).

	"utr": SNPs in UTR regions. "gene": SNPs in genes. default:FALSE
freq.table	the frequency counts for the CLR test. "list"
FST	use FST values instead of the minor allele frequencies

Details

The freq.table contains the global sets of frequency counts. It can be produced with the module detail.stats. The values in the slot GENOME.class@region.stats@minor.allele.frequencies can be used to create this global set. (use the R function table) freq.table is a list of length n.pops.

Value

The return value is a modified object of class "GENOME"

The following slots will be modified in the "GENOME" object

CL	Composite Likelihood of SNPs
CLR	Nielsen's CLR test

References

Cai JJ (2008) *PGEToolbox: A Matlab toolbox for population genetics and evolution* Journal of Heredity Jul-Aug;99(4):438-40.doi:10.1093/jhered/esm127

Nielson, R. (2005). *Genomic scans for selective sweeps using SNP data* Genome Res. 2005 15: 1566-1575

Examples

```
# Reading one alignment stored in the folder Aln
# GENOME.class <- readData("\home\Aln")
#
# CL
# GENOME.class <- sweeps.stats(GENOME.class)
# GENOME.class@CL
#
# CLR
# create global set
# GENOME.class  <- detail.stats(GENOME.class)
# freq <- GENOME.class@region.stats@minor.allele.freqs[[1]]
# freq.table <- list()
```

```
# freq.table[[1]] <- table(freq)
# define the region of interest
# GENOME.class.split <- splitting.data(GENOME.class, positions= ...)
# calculate CLR
# GENOME.class.split <- sweeps.stats(GENOME.class.split, freq.table=freq.table)
# GENOME.class@CLR
```

`test.params-class` *Set parameters for coalescent simulations with Hudson's MS and Ewing's MSMS.*

Description

The object that contains the set parameter values can be passed to the function MS. This class simpliifes the process of passing on all necessary values to the MS function.

Arguments

`theta`	mutation parameter theta (4Nmu), where N is the diplod effective population size and mu the mutation rate per locus. It needs to be provided as a vector of length n.regions
`seeds`	specify 3 random number seeds. a vector of length 3 with positive integer values is expected
`fixedSegsites`	usually the number of segregating sites varies in each iteration. Please provide a single numeric value if the number of segregating sites needs to be fixed.
`recombination`	provide a vector of format: c(p, nsites), p = cross-over parameter rate, nsites is the number of sites between which recombination occurs
`geneConv`	in addition to recombination, intra-locus non-cross-over exchange gene conversion can be included in the simulation; the expected format is c(f, gamma), where f denotes the ratio g/r (r is the probability per generation of crossing-over between adjacent sites (see Wiuf and Hein 2000), and gamma is the mean conversion tract length.
`growth`	population size is assumed to be $N(t) = N0 exp^alpha*t$. Provide alpha as an integer value. Negative values indicate that population was larger in the past than present, positive values indicate that it was smaller.
`migration`	specify the migration rate between populations. Please provide a single numeric value.
`demography`	vector of length 3 or 4 with first value denoted as 'type' valid 'types' for vectors of length 3 are as follows: - 1 set a growth rate change alpha at a certain time t: c(1, t, alpha) - 2 set all sub-populations to size $x * N_0$ and growth rate to zero:

c(2, t, x)

- 3 set all elements of the migration matrix to $x/(npop-1)$:
c(3, t, x)

valid 'types' for vectors of length 4 are as follows:
- 4 set growth rate of sub-population i to alpha at time z:
c(4, t, i, alpha)

- 5 set sub-population i size to $x * N_0$ at time t and growth rate to zero:
c(5, t, i, x)

- 6 split sub-population i into sub-population i and a new sub-population, labeled npop + 1. Each ancestral lineage in sub-population i is randomly assigned to sub-population i with probability p and sub-population npop + 1 with probability 1 - p. The size of sub-population npop + 1 is set to N_0. Migration rates to and from the new sub-population are assumed to be zero and the growth rate of the new sub-population is set to zero:
c(6, t, i, p)

- 7 move all lineages in sub-population i to sub-population j at time t. Migration rates from sub-population i are set to zero:
c(7, t, i, j)

Author(s)

Bastian Pfeifer

See Also

MS

Examples

```
# params           <- new("test.params")
# params@theta     <- rep(5,n.regions)
# params@migration <- 3
```

`vcf_file` *VCF file (subdirectory "data")*

Description

A VCF file stored in the directory "data" of the PopGenome package. The file (unpacked) has to be stored in a folder (for example "VCF"). Note that many VCF-files can be stored in this folder and are read consecutively. If the VCF file is too large to fit into the computer's main memory, split it into chunks (by position) ! PopGenome is able to concatenate these chunks afterwards. `readData("VCF",format="VCF",FAST=TRUE)`

`VCF_split_into_scaffolds` *Split a VCF file into multiple scaffold-VCFs*

Description

This function splits a VCF file into multiple VCFs including data for exactly one scaffold each.

Usage

```
VCF_split_into_scaffolds(VCF.file, output.folder)
```

Arguments

`VCF.file`	the basepath of the VCF file
`output.folder`	name of the folder where the VCFs should be stored

Details

The algorithm splits the VCF into multiple scaffold based VCFs and
stores the files in a given folder. This folder can be used as an input for readData(,format="VCF")

Value

`TRUE`

Examples

```
# VCF_split_into_scaffolds("VCFfile.vcf","scaffoldVCFs")
# test <- readData("scaffoldVCFs", format="VCF")
```

`weighted.jackknife-methods` *Weighted Jackknife*

Description

Weighted Jackknife calculations.

Usage

```
## S4 method for signature 'GENOME'
weighted.jackknife(object,
do.D=TRUE,
do.df=TRUE,
per.region=FALSE,
block.size=1)
```

Arguments

`object`	An object of class `"GENOME"`
`do.D`	Pattersons D and Martin's f statistic
`do.df`	d-fraction (distance fraction)
`per.region`	jacknife within regions
`block.size`	the block size for jackknife (SNPs)

Details

To perform the D and f statistic one needs to define 3 populations via the function `set.populations`, where the third population represent the archaic population. In addition, an outgroup is required and have to be set via the function `set.outgroup`. Here, only SNPs where the outgroup is monomorphic are considered. f is the fraction of the genome that is admixed [2].

Value

Returned value is a modified object of class `"GENOME"`

The following slots will be modified in the `"GENOME"` object

	Slot	Reference	Description
1.	`df.z`	[3]	z values (jackknife)
2.	`df.SE`	[3]	standard error (jackknife)
3.	`df.pval`	[3]	P values (jackknife)

4.	D.z	[x]	z values (jackknife)
5.	D.SE	[x]	standard error (jackknife)
6.	D.pval	[x]	P values (jackknife)

References

[1] Durand, E. Y., Patterson, N. J., Reich, D., & Slatkin, M. (2011). *Testing for ancient admixture between closely related populations.*
Molecular Biology and Evolution, 28(8), 2239–2252. doi:10.1093/molbev/msr048
[2] Simon H Martin, Kanchon K Dasmahapatra, Nicola J Nadeau, et al. (2013). *Genome-wide evidence for speciation with gene flow in Heliconius butterflies.*
Genome Res. doi:10.1101/gr.159426.113
[3] Bastian Pfeifer and Durrell D. Kapan (2019). *Estimates of introgression as a function of pairwise distances.*
BMC Bioinformatics. https://doi.org/10.1186/s12859-019-2747-z

Examples

```
# GENOME.class <- readData("\home\Alignments")
# GENOME.class <- set.populations(GENOME.class,list(1:3,4:8,9:12))
# GENOME.class <- set.outgroup(GENOME.class,13)
# GENOME.class <- introgression.stats(GENOME.class, do.D=TRUE)
# show the result:
# GENOME.class@D
# GENOME.class@f
# GENOME.class <- introgression.stats(GENOME.class, do.df=TRUE)
# show the result:
# GENOME.class@df
```

Whop_readVCF *Reading tabixed VCF files (an interface to WhopGenome)*

Description

This function provides an interface to the WhopGenome package which is specialized to read tabix-indexed VCF files.

Usage

```
Whop_readVCF(v, numcols, tid, frompos, topos,
        samplenames=NA, gffpath = FALSE, include.unknown=FALSE)
```

Arguments

v	a vcf_handle returned from vcf_open()
numcols	number of SNPs that should be read in as one chunk
tid	which chromosome ? (character)
frompos	start of the region
topos	end of the region
samplenames	a vector of individual names/IDs
gffpath	the corresponding GFF file
include.unknown	including positions with unknown nucleotides

Details

WhopGenome is required ! require(WhopGenome) WhopGenome provides some powerful filter meachanisms which can be applied to the VCF reading process. The filter rules can be set via WhopGenome functions. Whop_readVCF expects a vcf_handle returned from vcf_open. The Whop_readVCF function expects a tabixed VCF with a diploid GT-field.
In case of haploid data, the GT-field has to be transformed to a pseudo- diploid field (0 -> 0|0 etc.).
An alternative is to use readData(..., format="VCFhap") which can read non-tabixed haploid VCFs directly.
The ff-package we use limits the data size to individuals * (number of SNPs) <= .Machine$integer.max
In case of very large data sets, the bigmemory package will be used.
This may slow down calculations.
See also readData(..., format="VCF") !

Value

The function creates an object of class "GENOME"

The following slots will be filled in the "GENOME" object

	Slot	Description
1.	n.sites	total number of sites
2.	n.biallelic.sites	number of biallelic sites
3.	region.data	some detailed information on the data read
4.	region.names	names of regions

Examples

```
# require(WhopGenome)
# vcf_handle   <- vcf_open("chr2.vcf.gz")
```

```
# GENOME.class <- Whop_readVCF(vcf_handle, 1000, "2", 1, 100000)
# GENOME.class
# GENOME.class@region.names
```

Index