Package ‘PopGenome’

May 12, 2019

Type Package
Title An Efficient Swiss Army Knife for Population Genomic Analyses
Version 2.7.1
Date 2019-05-10
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 individual 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
URL http://popgenome.weebly.com
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 2019-05-11 22:11:18 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achaz.stats-methods</td>
<td>3</td>
</tr>
<tr>
<td>BayeScanR</td>
<td>4</td>
</tr>
<tr>
<td>calc.fixed.shared-methods</td>
<td>6</td>
</tr>
<tr>
<td>calc.R2-methods</td>
<td>7</td>
</tr>
<tr>
<td>codontable</td>
<td>8</td>
</tr>
<tr>
<td>concatenate.classes</td>
<td>9</td>
</tr>
<tr>
<td>concatenate.regions</td>
<td>10</td>
</tr>
<tr>
<td>count.unknowns-methods</td>
<td>10</td>
</tr>
<tr>
<td>create.PopGenome.method</td>
<td>11</td>
</tr>
<tr>
<td>detail.stats-methods</td>
<td>12</td>
</tr>
<tr>
<td>diversity.stats-methods</td>
<td>14</td>
</tr>
<tr>
<td>diversity.stats.between-methods</td>
<td>16</td>
</tr>
<tr>
<td>fasta_file</td>
<td>17</td>
</tr>
<tr>
<td>F_ST.stats-methods</td>
<td>18</td>
</tr>
<tr>
<td>F_ST.stats.2-methods</td>
<td>20</td>
</tr>
<tr>
<td>GENOME-class</td>
<td>22</td>
</tr>
<tr>
<td>get.biallelic.matrix-methods</td>
<td>27</td>
</tr>
<tr>
<td>get.codons-methods</td>
<td>28</td>
</tr>
<tr>
<td>get.feature.names</td>
<td>29</td>
</tr>
<tr>
<td>get.individuals-methods</td>
<td>30</td>
</tr>
<tr>
<td>get.status-methods</td>
<td>31</td>
</tr>
<tr>
<td>getBayes-methods</td>
<td>31</td>
</tr>
<tr>
<td>get_gff_info</td>
<td>32</td>
</tr>
<tr>
<td>gff_file</td>
<td>33</td>
</tr>
<tr>
<td>GFF_split_into_scaffolds</td>
<td>33</td>
</tr>
<tr>
<td>introgression.stats-methods</td>
<td>34</td>
</tr>
<tr>
<td>jack.knife.transform</td>
<td>36</td>
</tr>
<tr>
<td>linkage.stats-methods</td>
<td>37</td>
</tr>
<tr>
<td>load.session</td>
<td>38</td>
</tr>
<tr>
<td>MKT-methods</td>
<td>39</td>
</tr>
<tr>
<td>MS</td>
<td>41</td>
</tr>
<tr>
<td>MS_getStats</td>
<td>42</td>
</tr>
<tr>
<td>mult.linkage.stats-methods</td>
<td>43</td>
</tr>
<tr>
<td>neutrality.stats-methods</td>
<td>44</td>
</tr>
<tr>
<td>PG_plot.biallelic.matrix-methods</td>
<td>46</td>
</tr>
<tr>
<td>PopGenome</td>
<td>47</td>
</tr>
<tr>
<td>PopGplot</td>
<td>48</td>
</tr>
<tr>
<td>read.big.fasta</td>
<td>49</td>
</tr>
<tr>
<td>readData</td>
<td>50</td>
</tr>
<tr>
<td>readHapMap</td>
<td>53</td>
</tr>
<tr>
<td>readMS</td>
<td>54</td>
</tr>
<tr>
<td>readSNP</td>
<td>55</td>
</tr>
<tr>
<td>readVCF</td>
<td>56</td>
</tr>
<tr>
<td>recomb.stats-methods</td>
<td>58</td>
</tr>
<tr>
<td>region.as.fasta-methods</td>
<td>59</td>
</tr>
<tr>
<td>save.session</td>
<td>60</td>
</tr>
</tbody>
</table>
Achaz.stats-methods

Description
Achaz statistic

Usage

```r
## 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


Examples

```r
# 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 & Gaggiotti 2008)

Description

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

Usage

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

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


Examples

```r
# 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
```


**Description**

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

**Usage**

```r
## 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 divided 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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n.fixed.sites</td>
<td>[x]</td>
<td>Number of fixed sites</td>
</tr>
<tr>
<td>n.shared.sites</td>
<td>[x]</td>
<td>Number of shared sites</td>
</tr>
<tr>
<td>n.monomorphic.sites</td>
<td>[x]</td>
<td>Number of monomorphic sites</td>
</tr>
</tbody>
</table>

References

[x]

Examples

```r
# 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

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

Arguments

<table>
<thead>
<tr>
<th>object</th>
<th>an object of class &quot;GENOME&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>subsites</td>
<td>same as in the other modules</td>
</tr>
<tr>
<td>lower.bound</td>
<td>sites with minor-allele-frequency&gt;=lower.bound are considered</td>
</tr>
<tr>
<td>upper.bound</td>
<td>sites with minor-allele-frequency&lt;=upper.bound are considered</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th><code>codontable</code></th>
<th>Prints the codon table which is used in the PopGenome framework</th>
</tr>
</thead>
</table>

Description

This function prints the nucleotide triplets (as numerical values) and the corresponding 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.
### concatenate.classes

#### Concatenate GENOME classes

**Description**

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

**Usage**

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

**Arguments**

- `classlist`: a list of GENOME objects

**Value**

The function creates an object of class "GENOME".

**Examples**

```r
# a <- readData("Three Alignments/")
# b <- readData("Two Alignments/")
# ab <- concatenate.classes(list(a,b))
# ab <- neutrality.stats(ab)
# ab@Tajima.D
# ab@region.names
```
**concatenateRegions**  
*Concatenate regions*

**Description**

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

**Usage**

```r
concatenateRegions(object)
```

**Arguments**

- `object`: object of class `genome`

**Value**

The function creates an object of class "GENOME".

**Examples**

```r
# GENOME.class <- readData("Three Alignments/")
# WHOLE <- concatenateRegions(GENOME.class)
# WHOLE <- neutrality.stats(WHOLE)
# WHOLE@Tajima.D
```

---

**count.unknowns**

*Calculate missing nucleotide frequencies*

**Description**

A generic function to calculate the missing nucleotide frequencies.

**Usage**

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

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

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

Description

This generic function calculates some mixed statistics.

Usage

```r
## 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
population is monomorphic, the remaining individuals are monomorphic with different values.

Examples

```r
# 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**

```r
## 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.

\[ \text{pi} \] Nei's calculation of pi

\[ \text{keep\_site\_info} \] stores site specific values in GENOME\_class@region\_stats

Details

The nucleotide diversities have to be divided 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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nuc_diversity_within</td>
<td>[1,3]</td>
<td>Nucleotide diversity (within the population)</td>
</tr>
<tr>
<td>( \text{Pi} )</td>
<td>[2]</td>
<td>Diversity from Nei (within the population)</td>
</tr>
<tr>
<td>hap_diversity_within</td>
<td>[1]</td>
<td>Haplotype diversity (within the population)</td>
</tr>
</tbody>
</table>

References


Examples

# GENOME\_class <- readData("\home\Alignments")
# GENOME\_class
# GENOME\_class <- diversity\_stats(GENOME\_class)
# GENOME\_class <- diversity\_stats(GENOME\_class,list(1:4:10))
# 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

```r
## 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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. nuc.diversity.between</td>
<td>[1,3]</td>
<td>Nucleotide diversity (between the population)</td>
</tr>
<tr>
<td>2. hap.diversity.between</td>
<td>[1]</td>
<td>Haplotype diversity (between the population)</td>
</tr>
</tbody>
</table>

**References**


**Examples**

```r
# 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.
Description

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

Usage

```r
## 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 pairwise 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 divided 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
<table>
<thead>
<tr>
<th>Slot</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>haplotype.F_ST</td>
<td>[1]</td>
<td>Fixation Index based on haplotype frequencies</td>
</tr>
<tr>
<td>nucleotide.F_ST</td>
<td>[1]</td>
<td>Fixation Index based on minor.allele frequencies</td>
</tr>
<tr>
<td>Nei.G_ST</td>
<td>[2]</td>
<td>Nei’s Fixation Index</td>
</tr>
<tr>
<td>Hudson.G_ST</td>
<td>[3]</td>
<td>see reference ...</td>
</tr>
<tr>
<td>Hudson.H.ST</td>
<td>[3]</td>
<td>see reference ...</td>
</tr>
<tr>
<td>Hudson.K.ST</td>
<td>[3]</td>
<td>see reference ...</td>
</tr>
<tr>
<td>nuc.diversity.within</td>
<td>[1,5]</td>
<td>Nucleotide diversity (within the population)</td>
</tr>
<tr>
<td>hap.diversity.within</td>
<td>[1]</td>
<td>Haplotype diversity (within the population)</td>
</tr>
<tr>
<td>Pi</td>
<td>[4]</td>
<td>Nei’s diversity (within the population)</td>
</tr>
<tr>
<td>hap.F_ST.vs.all</td>
<td>[1]</td>
<td>Fixation Index for each population against all other individuals (haplotype)</td>
</tr>
<tr>
<td>nuc.F_ST.vs.all</td>
<td>[1]</td>
<td>Fixation Index for each population against all other individuals (nucleotide)</td>
</tr>
<tr>
<td>hap.diversity.between</td>
<td>[1]</td>
<td>Haplotype diversities between populations</td>
</tr>
<tr>
<td>nuc.diversity.between</td>
<td>[1,5]</td>
<td>Nucleotide diversities between populations</td>
</tr>
<tr>
<td>nuc.F.ST.pairwise</td>
<td>[1]</td>
<td>Fixation Index for every pair of populations (nucleotide)</td>
</tr>
<tr>
<td>hap.F.ST.pairwise</td>
<td>[1]</td>
<td>Fixation Index for every pair of populations (haplotype)</td>
</tr>
<tr>
<td>Nei.G.ST.pairwise</td>
<td>[2]</td>
<td>Fixation Index for every pair of populations (Nei)</td>
</tr>
<tr>
<td>region.stats</td>
<td></td>
<td>an object of class &quot;region.stats&quot; for detailed statistics</td>
</tr>
</tbody>
</table>

References
See Also

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

Examples

```r
# 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("cseq1","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 measurements.

**Usage**

```r
## S4 method for signature 'GENOME'
F_ST.stats.2(object,new.populations="list",subsites=FALSE,smn=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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>

References


Examples

```r
# 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 defined 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 within 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 jackknife mode
missing.freqs: Missing nucleotide frequency
n.fixed.sites: ...
n.shared.sites: ...
n.monomorphic.sites: ...
Bd: ...
df: ...
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
deretail 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 from 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
get.biallelic.matrix-methods

Get the biallelic matrix

Description
This function returns the biallelic matrix of a specific region.

Usage

```r
## 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

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

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

Usage

```r
## 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 corresponding GFF file.
The function set.synnonsyn(..., save.codons=TRUE) sets the syn/nonsyn 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:

- Position of the SNPs
- Major Codon
- Minor Codon
- Major amino acid
- Minor amino acid
- synonymous (TRUE/FALSE)
- Polarity of the major amino acid
- Polarity of the minor amino acid
**get.feature.names**

**Feature informations and GFF-attributes**

**Description**

Returns feature names and additional attributes for a given region.

**Usage**

```r
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**

```r
# 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.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)
**Description**

Some information about the definitions of populations and subsites.

**Usage**

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

**Arguments**

- `object` An object of class "GENOME"

**Examples**

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

---

**getBayes-methods**  
*Get values for BayeScanR*

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

`coming soon !`
get_gff_info

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

```r
# 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)
```

<table>
<thead>
<tr>
<th>get_gff_info</th>
<th>Annotation info</th>
</tr>
</thead>
</table>

Description

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

Usage

```r
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

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

**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

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

*Description*

A generic function to estimate archaic admixture.

*Usage*

```r
## 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`: Bd-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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>[1: eq. 2]</td>
<td>Pattersons D statistic</td>
</tr>
<tr>
<td>f</td>
<td>[2]</td>
<td>f statistic</td>
</tr>
<tr>
<td>df</td>
<td>[3]</td>
<td>Bd-fraction</td>
</tr>
<tr>
<td>RNDmin</td>
<td>[x]</td>
<td>RNDmin</td>
</tr>
<tr>
<td>D.z</td>
<td>[x]</td>
<td>z values (jackknife)</td>
</tr>
<tr>
<td>D.pval</td>
<td>[x]</td>
<td>P values (jackknife)</td>
</tr>
<tr>
<td>df.z</td>
<td>[3]</td>
<td>z values (jackknife)</td>
</tr>
<tr>
<td>df.pval</td>
<td>[3]</td>
<td>P values (jackknife)</td>
</tr>
</tbody>
</table>

References


Examples

```r
# 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@d
```
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

```r
## 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

```r
# 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
```
Description

A generic function to calculate some linkage disequilibrium statistics.

Usage

```r
## 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.snp, 2)`.

Value

The return value is a modified object of class "GENOME"
The following slots will be modified in the "GENOME" object

<table>
<thead>
<tr>
<th>Slot</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallNb</td>
<td>[2]</td>
<td>Wall $B$ statistic (only adjacent positions are considered)</td>
</tr>
<tr>
<td>wallNq</td>
<td>[2]</td>
<td>Wall $Q$ statistic (only adjacent positions are considered)</td>
</tr>
<tr>
<td>Rozas.ZA</td>
<td>[1]</td>
<td>Rozas $ZA$ statistic (adjacent positions, if detail==TRUE)</td>
</tr>
<tr>
<td>Rozas.ZZ</td>
<td>[1]</td>
<td>Rozas $ZZ$ statistic ($ZZ=ZA-Z_nS$, if detail==TRUE)</td>
</tr>
</tbody>
</table>

References


Examples

```r
# 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@wallNb --> 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")

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:

<table>
<thead>
<tr>
<th>Columns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. P_nonsyn</td>
<td>nonsynonymous sites</td>
</tr>
<tr>
<td>2. P_syn</td>
<td>synonymous sites</td>
</tr>
<tr>
<td>3. D_nonsyn</td>
<td>fixed nonsynonymous sites</td>
</tr>
<tr>
<td>4. D_syn</td>
<td>fixed synonymous sites</td>
</tr>
<tr>
<td>5. neutrality.index</td>
<td>$(P_nonsyn/P_syn)/(D_nonsyn/D_syn)$</td>
</tr>
<tr>
<td>6. alpha</td>
<td>1-neutrality.index</td>
</tr>
</tbody>
</table>

References


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)

---

**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**

```r
MS(GENO,niter=10,thetaID="user",params=FALSE,detail=FALSE,
neutral=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"
- **neutral=FALSE** 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


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**

```r
# GENOME.class <- readData("...
# GENOME.class <- neutrality.stats(GENOME.class)
# ms <- MS(GENOME.class,thetaID="Tajima",neutralitiy=TRUE)
# MS.getStats(ms)
```

---

**Description**

This generic function calculates the linkage disequilibrium between regions.

**Usage**

```r
## 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

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

Description

This generic function calculates some neutrality statistics.

Usage

```r
## 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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n.segregating.sites</td>
<td></td>
<td>Total number of segregating sites</td>
</tr>
<tr>
<td>Fu.Li.F</td>
<td>[3]</td>
<td>Fu &amp; Li’s’ F* statistic 1993</td>
</tr>
<tr>
<td>Fu.Li.D</td>
<td>[3]</td>
<td>Fu &amp; Li’s D* statistic 1993</td>
</tr>
<tr>
<td>Strobeck.S</td>
<td>[5]</td>
<td>Strobeck’s S statistic 1987 (if detail==TRUE)</td>
</tr>
<tr>
<td>Fu.F_S</td>
<td>[4]</td>
<td>Fu’s FS_S$ statistic 1997 (if detail==TRUE)</td>
</tr>
<tr>
<td>theta_Tajima</td>
<td>[1]</td>
<td></td>
</tr>
<tr>
<td>theta_Watterson</td>
<td></td>
<td></td>
</tr>
<tr>
<td>theta_Fu.Li</td>
<td>[3]</td>
<td></td>
</tr>
<tr>
<td>theta_Achaz.Watterson</td>
<td></td>
<td></td>
</tr>
<tr>
<td>theta_Achaz.Tajima</td>
<td></td>
<td></td>
</tr>
<tr>
<td>theta_Fay.Wu</td>
<td>[6]</td>
<td></td>
</tr>
</tbody>
</table>
16. theta_Zeng

References


Examples

```r
# 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
```

Plot the biallelic matrix
Description

This function plots the biallelic matrix of a specific region.

Usage

```r
## 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

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

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)
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

```r
# 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.diversity.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**

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

**Arguments**

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

**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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. n.sites</td>
<td>total number of sites</td>
</tr>
<tr>
<td>2. n.biallelic.sites</td>
<td>number of biallelic sites</td>
</tr>
<tr>
<td>3. region.names</td>
<td>names of regions</td>
</tr>
<tr>
<td>4. region.data</td>
<td>some detailed information about the data</td>
</tr>
</tbody>
</table>

Examples

```r
# 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

```r
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)
**readData**

**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!
- **parallelized**: 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,-

**parallelized:**
- 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

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

Examples

```r
# 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  

Describe
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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. n.sites</td>
<td>total number of sites</td>
</tr>
<tr>
<td>2. n.biallelic.sites</td>
<td>number of biallelic sites</td>
</tr>
<tr>
<td>3. region.data</td>
<td>some detailed information about the data read</td>
</tr>
</tbody>
</table>
Examples

```r
# 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**

```r
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**


**Examples**

```r
# 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"

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. n.sites</td>
<td>total number of sites</td>
</tr>
<tr>
<td>2. n.biallelic.sites</td>
<td>number of biallelic sites</td>
</tr>
<tr>
<td>3. region.data</td>
<td>some detailed information about the data read</td>
</tr>
<tr>
<td>4. region.names</td>
<td>names of regions</td>
</tr>
</tbody>
</table>

**Examples**

```r
# GENOME.class <- readSNP("\nSNPData")
# GENOME.class <- readSNP("\nSNPData", CHR=1)
# GENOME.class <- readSNP("\nSNPData", 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
```

**Description**

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

**Usage**

```r
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
readVCF

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

**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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n.sites</td>
<td>total number of sites</td>
</tr>
<tr>
<td>n.biallelic.sites</td>
<td>number of biallelic sites</td>
</tr>
<tr>
<td>region.data</td>
<td>some detailed information about the data read</td>
</tr>
<tr>
<td>region.names</td>
<td>names of regions</td>
</tr>
</tbody>
</table>
Examples

```r
# 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**

```r
## 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


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

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

```r
#GENOME.class <- readSNP("Arabidopsis",CHR=1)
# split the data into the genomic 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

```r
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

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

Description

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

Usage

```r
## 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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>region.data@included</td>
<td>[x]</td>
<td>...</td>
</tr>
</tbody>
</table>

References

[x]

Examples

```r
# 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")
```

Description

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

Usage

```r
## 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)
```

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)
```
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

```r
## 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

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

Description

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

```r
## 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 not specified correctly. PopGenome can also handle coding regions on reverse strands. We have used the program SNPeff to validate our results.

Examples

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

Description

coming soon ...

Methods

- `object = "GENOME"` coming soon ...
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 separately.

Value

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

Note

If you want to scan regions separately (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

```r
# 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=1000000, 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**

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

**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**

```r
# 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
```

---

**Description**

Splits the data into GFF features defined by the user.

**Usage**

```r
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

```r
# 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

```r
## 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

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


Examples

```r
# 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()
```
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 simplifies 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 $N(t) = N_0 \exp^{\alpha t}$. 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.

Examples

```
readData("VCF", format="VCF");
```
### Weighted Jackknife

**Description**

Weighted Jackknife calculations.

**Usage**

```r
## 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** Patterson's 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**

<table>
<thead>
<tr>
<th>Slot</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>df.z</td>
<td>[3]</td>
<td>z values (jackknife)</td>
</tr>
<tr>
<td>df.SE</td>
<td>[3]</td>
<td>standard error (jackknife)</td>
</tr>
<tr>
<td>df.pval</td>
<td>[3]</td>
<td>P values (jackknife)</td>
</tr>
</tbody>
</table>
4. D.z [x] z values (jackknife)
5. D.SE [x] standard error (jackknife)
6. D.pval [x] P values (jackknife)

References


Examples

```r
# 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

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

**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 mechanisms 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 -> 00 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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <code>n.sites</code></td>
<td>total number of sites</td>
</tr>
<tr>
<td>2. <code>n.biallelic.sites</code></td>
<td>number of biallelic sites</td>
</tr>
<tr>
<td>3. <code>region.data</code></td>
<td>some detailed information on the data read</td>
</tr>
<tr>
<td>4. <code>region.names</code></td>
<td>names of regions</td>
</tr>
</tbody>
</table>

**Examples**

```r
# 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

*Topic classes
  GENOME-class, 22
  test.params-class, 72

*Topic datasets
  fasta_file, 17
  gff_file, 33
  snp_file, 67
  vcf_file, 73

*Topic methods
  Achaz.stats-methods, 3
  BayeScanR, 4
  calc.fixed.shared-methods, 6
  calc.R2-methods, 7
  codontable, 8
  concatenate.classes, 9
  concatenate.regions, 10
  count.unknowns-methods, 10
  create.PopGenome.method, 11
  detail.stats-methods, 12
  diversity.stats-methods, 14
  diversity.stats.between-methods, 16
  F_ST.stats-methods, 18
  F_ST.stats.2-methods, 20
  get.biallelic.matrix-methods, 27
  get.codons-methods, 28
  get.individuals-methods, 30
  get.status-methods, 31
  get_gff_info, 32
  getBayes-methods, 31
  introgression.stats-methods, 34
  jackknife.transform, 36
  linkage.stats-methods, 37
  load.session, 38
  MKT-methods, 39
  MS, 41
  MS.getStats, 42
  mult.linkage.stats-methods, 43
  neutrality.stats-methods, 44
  PG_plot.biallelic.matrix-methods, 46
  PopGPLOT, 48
  readData, 50
  readHapMap, 53
  readMS, 54
  readSNP, 55
  readVCF, 56
  recomb.stats-methods, 58
  region.as.fasta-methods, 59
  save.session, 60
  set.filter-methods, 61
  set.outgroup-methods, 62
  set.populations-methods, 63
  set.ref.positions-methods, 64
  set.synnonsyn-methods, 64
  show.slots-methods, 65
  sliding.window.transform-methods, 66
  splitting.data-methods, 67
  sweeps.stats-methods, 70
  weighted.jackknife-methods, 75
  whop_readVCF, 76

*Topic package
  PopGenome, 47

Achaz.stats (GENOME-class), 22
Achaz.stats, GENOME-method
  (Achaz.stats-methods), 3
Achaz.stats-methods, 3

BayeScanR, 4

calc.fixed.shared (GENOME-class), 22
calc.fixed.shared, GENOME-method
  (calc.fixed.shared-methods), 6
calc.fixed.shared-methods, 6
calc.R2 (GENOME-class), 22
calc.R2, GENOME-method
  (calc.R2-methods), 7
calc.R2-methods, 7
codontable, 8
concatenate.classes, 9
concatenate.regions, 10
count.unknowns (GENOME-class), 22
count.unknowns, GENOME-method
  (count.unknowns-methods), 10
count.unknowns-methods, 10
create.PopGenome.method, 11
detail.stats (GENOME-class), 22
detail.stats, GENOME-method
  (detail.stats-methods), 12
detail.stats-methods, 12
diversity.stats (GENOME-class), 22
diversity.stats, GENOME-method
  (diversity.stats-methods), 14
diversity.stats-methods, 14
diversity.stats.between, GENOME-method
  (diversity.stats.between-methods), 16
diversity.stats.between-methods, 16
F_ST.stats (GENOME-class), 22
F_ST.stats, GENOME-method
  (F_ST.stats-methods), 18
F_ST.stats-methods, 18
F_ST.stats.2, 20
F_ST.stats.2, GENOME-method
  (F_ST.stats.2-methods), 20
F_ST.stats.2-methods, 20
fasta_file, 17
GENOME-class, 22
get.biallelic.matrix (GENOME-class), 22
get.biallelic.matrix, GENOME-method
  (get.biallelic.matrix-methods), 27
get.biallelic.matrix-methods, 27
get.codons (GENOME-class), 22
get.codons, GENOME-method
  (get.codons-methods), 28
get.codons-methods, 28
get.detail (GENOME-class), 22
get.detail, GENOME-method
  (get.detail-methods), 12
get.detail-methods
  (get.detail-methods), 12
get.diversity (GENOME-class), 22
get.diversity, GENOME-method
  (get.diversity-methods), 18
get.diversity-methods
  (get.diversity-methods), 18
get.F_ST (GENOME-class), 22
get.F_ST, GENOME-method
  (get.F_ST-methods), 18
get.F_ST-methods (get.F_ST-methods), 18
get.feature.names, 29
get.individuals (GENOME-class), 22
get.individuals, GENOME-method
  (get.individuals-methods), 30
get.individuals-methods, 30
get.linkage (GENOME-class), 22
get.linkage, GENOME-method
  (get.linkage-methods), 37
get.linkage-methods
  (get.linkage-methods), 37
get.MKT (GENOME-class), 22
get.MKT, GENOME-method (MKT-methods), 39
get.MKT-methods (MKT-methods), 39
get.neutrality (GENOME-class), 22
get.neutrality, GENOME-method
  (get.neutrality-methods), 44
get.neutrality-methods
  (get.neutrality-methods), 44
get.recomb (GENOME-class), 22
get.recomb, GENOME-method
  (get.recomb-methods), 58
get.recomb-methods
  (get.recomb-methods), 58
get.status (GENOME-class), 22
get.status, GENOME-method
  (get.status-methods), 31
get.status-methods, 31
get.sum.data (GENOME-class), 22
get.sum.data, GENOME-method (readData), 50
get.sum.data-methods (readData), 50
get.sweeps (GENOME-class), 22
get.sweeps, GENOME-method
  (get.sweeps-methods), 70
get.sweeps-methods
  (get.sweeps-methods), 70
get_gff_info, 32
getBayes (GENOME-class), 22
getBayes, GENOME-method
(getBayes-methods), 31
getBayes-methods, 31
getMS, GENOME-method (GENOME-class), 22
gff_file, 33
GFF_split_into_scaffolds, 33
introgression.stats (GENOME-class), 22
introgression.stats, GENOME-method
(introgression.stats-methods), 34
introgression.stats-methods, 34
jack.knife.transform, 36
jack.knife.transform (GENOME-class), 22
jack.knife.transform, GENOME-method
(jack.knife.transform), 36
jack.knife.transform-methods
(jack.knife.transform), 36
linkage.stats (GENOME-class), 22
linkage.stats, GENOME-method
(linkage.stats-methods), 37
linkage.stats-methods, 37
load.session, 38
MKT (GENOME-class), 22
MKT, GENOME-method (MKT-methods), 39
MKT-methods, 39
MS, 41, 73
MS_getStats, 42
mult.linkage.stats (GENOME-class), 22
mult.linkage.stats, GENOME-method
(mult.linkage.stats-methods), 43
mult.linkage.stats-methods, 43
neutrality.stats (GENOME-class), 22
neutrality.stats, GENOME-method
(neutrality.stats-methods), 44
neutrality.stats-methods, 44
PG_plot.biallelic.matrix
(GENOME-class), 22
PG_plot.biallelic.matrix, GENOME-method
(PG_plot.biallelic.matrix-methods), 46
PG_plot.biallelic.matrix-methods, 46
popFSTN, GENOME-method (GENOME-class), 22
PopGenome, 47
PopGPlot, 48
read.big.fasta, 49
readData, 48, 50
readHapMap, 53
readMS, 54
readSNP, 55
readVCF, 56
recomb.stats (GENOME-class), 22
recomb.stats, GENOME-method
(recomb.stats-methods), 58
recomb.stats-methods, 58
region.as.fasta (GENOME-class), 22
region.as.fasta, GENOME-method
(region.as.fasta-methods), 59
region.as.fasta-methods, 59
save.session, 60
set.filter (GENOME-class), 22
set.filter, GENOME-method
(set.filter-methods), 61
set.filter-methods, 61
set.outgroup (GENOME-class), 22
set.outgroup, GENOME-method
(set.outgroup-methods), 62
set.outgroup-methods, 62
set.populations (GENOME-class), 22
set.populations, GENOME-method
(set.populations-methods), 63
set.populations-methods, 63
set.ref.positions (GENOME-class), 22
set.ref.positions, GENOME-method
(set.ref.positions-methods), 64
set.ref.positions-methods, 64
set.synnonsyn (GENOME-class), 22
set.synnonsyn, GENOME-method
(set.synnonsyn-methods), 64
set.synnonsyn-methods, 64
show, GENOME-method (GENOME-class), 22
show, GENOME-method
(show.slots-methods), 65
show, GENOME-method
(show.slots-methods), 65
sliding.window.transform
(GENOME-class), 22
sliding.window.transform, GENOME-method
(sliding.window.transform-methods), 66
sliding.window.transform-methods, 66
snp_file, 67
split_data_into_GFF_attributes, 68
split_data_into_GFF_features, 69
splitting.data (GENOME-class), 22
splitting.data, GENOME-method
   (splitting.data-methods), 67
splitting.data-methods, 67
sweeps.stats (GENOME-class), 22
sweeps.stats, GENOME-method
   (sweeps.stats-methods), 70
sweeps.stats-methods, 70
test.params (test.params-class), 72
test.params-class, 72
usage, GENOME-method (GENOME-class), 22
vcf_file, 73
VCF_split_into_scaffolds, 74
weighted.jackknife (GENOME-class), 22
weighted.jackknife, GENOME-method
   (weighted.jackknife-methods), 75
weighted.jackknife-methods, 75
Whop_readVCF, 76