PopGenome Session

Bastian Pfeifer

December 14, 2013

1 Introduction

PopGenome is a new package for population genomic analysis and method development. PopGenome includes, e.g., a wide range of polymorphism, neutrality statistics and FST estimates, which are applicable to sequence data stored in alignment format, as well as whole genome SNP data from the 1000/1001 Genome projects. The full range of methods can be applied to sliding-windows based on either the whole genome or only the SNPs. PopGenome is also able to handle GFF/GTF annotation files and automatically specifies the SNPs located in, e.g., exon or intron regions. Those subsites can be analyzed at once (e.g., all introns together) or each region seperately (e.g., one value per intron). The PopGenome framework is linked to Hudson's MS program for significance tests using coalescent simulations.

The following sections explain how to use the PopGenome package. Detail informations about the functions and their corresponding parameter can be found in the PopGenome manual on CRAN.

2 Install PopGenome

Install the package via R

```
> install.packages("PopGenome")
```

Loading the PopGenome package

```
> library(PopGenome)
```

3 Reading data (alignments)

Reading three alignments in FASTA-format (*4CL1tl.fas, C4Htl.fas and CADtl.fas*) stored in the folder "FASTA". (An example FASTA-file can be found in the *data* subdirectory of the PopGenome package on CRAN. Alignment formats like *Phylip, MEGA, MAF* are also accepted.
Note: valid nucleotides are A,a,C,c,T,t,U,u,G,g,-(gap),N,n(unknown). Internally those nucleotides are coded into numeric values:

- $T, U \rightarrow 1$

- $C \rightarrow 2$
- $G \rightarrow 3$
- $A \rightarrow 4$
- $unknown \rightarrow 5$
- $- \rightarrow 6$

```
> GENOME.class <- readData("FASTA")
```

GENOME.class is an object of class GENOME. When printing GENOME.class we get some informations about the main methods provided by PopGenome and how to get the results. The GENOME class is the input for every function printed below.
Note: GENOME.class ist just a variable, you can choose here whatever you want.

```
> GENOME.class
```

```
-----
Modules:
-----
                 Calculation             Description         Get.the.Result
1                   readData            Reading data           get.sum.data
2           neutrality.stats        Neutrality tests         get.neutrality
3              linkage.stats  Linkage disequilibrium            get.linkage
4              recomb.stats           Recombination             get.recomb
5                F_ST.stats          Fixation index get.F_ST,get.diversity
6                       MKT  McDonald-Kreitman test                get.MKT
7              detail.stats        Mixed statistics             get.detail
8                        MS   Coalescent simulation                      @
9            --------------             -----------          -------------
10          set.populations Defines the populations
11 sliding.window.transform          Sliding window
12           splitting.data         Splits the data
13               show.slots        ?provided slots?
14               get.status  Status of calculations
```

The class GENOME contains all observed data and statistic values which are presentable in a multi-locus-scale. Use the function show.slots(GENOME.class) to get an overview or check out the PopGenome manual on CRAN. To access those values we use the @-operator.

How many sites were analyzed in each alignment ?

```
> GENOME.class@n.sites
```

```
4CL1tl.fas  C4Htl.fas  CADtl.fas
      2979       2620       2930
```

```
> GENOME.class@region.names
```

```
[1] "4CL1tl.fas" "C4Htl.fas"  "CADtl.fas"
```

To get some summary information from the alignments use the `get.sum.data` function. This function extracts the values from the class `GENOME` and puts them into a matrix. We can also look at those values seperately with the @-operator (`GENOME.class@n.biallelic.sites`).

```
> get.sum.data(GENOME.class)
```

```
           n.sites n.biallelic.sites n.gaps n.unknowns n.valid.sites
4CL1tl.fas    2979               176    617          0          2362
C4Htl.fas     2620                84   1454          0          1161
CADtl.fas     2930               197    740          0          2189
           n.polyallelic.sites trans.transv.ratio
4CL1tl.fas                   0           1.120482
C4Htl.fas                    5           1.470588
CADtl.fas                    1           0.970000
```

The Slot `region.data` contains some detail (site specific) informations, which are not presentable in a multi-locus-scale. `region.data` is another class and its slots are accessable with the @ operator. See also the figure in section `PopGenome classes`.

```
> GENOME.class@region.data
```

```
-----
SLOTS:
-----
                     Slots                           Description
1              populations    Samples of each population (rows)
2             populations2   Samples of each population (names)
3                 outgroup                  Samples of outgroup
4              transitions           Biallelic site transitions
5         biallelic.matrix                     Biallelic matrix
6             n.singletons                Number of singletons
7          biallelic.sites          Position of biallelic sites
8                reference                          SNP reference
9            n.nucleotides   Number of nucleotides per sequence
10  biallelic.compositions Nucleotides per sequence (biallelic)
11              synonymous           Synonymous biallelic sites
12 biallelic.substitutions              Biallelic substitutions
13       polyallelic.sites            Sites with >2 nucleotides
14         sites.with.gaps             Sites with gap positions
15     sites.with.unknowns         Sites with unknown positions
16           minor.alleles                          Minor alleles
17                 codons    Codons of biallelic substitutions
18              IntronSNPS                SNPs in intron region
19                 UTRSNPS                   SNPs in UTR region
20              CodingSNPS                SNPs in coding region
21                ExonSNPS                  SNPs in exon region
22                GeneSNPS                  SNPs in gene region
```

```
---------------
These are the Slots (class region.data)
```

The first 10 biallelic positions (`[1:10]`) of the first alignment (`[[1]]`):

```
> GENOME.class@region.data@biallelic.sites[[1]][1:10]

 [1]  12  13  31  44  59 101 121 154 165 202
```

Which of those biallelic sites are transitions ?

```
> GENOME.class@region.data@transitions[[1]][1:10]

 [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE
```

3.1 The slots of the class `region.data`

populations
'list' of length n.populations. Contains the row identifiers (biallelic.matrix) of each individual
populations2
list of length n.populations. Contains the character names of each individual
outgroup
contains the row identifiers (biallelic.matrix) of the outgroup individuals
transitions
a boolean vector of length n.snps. `TRUE` if the substitution producing the SNP was a transition
biallelic.matrix
all calculations are based on this matrix. It contains zeros (mayor allele) and ones (minor allele). rows=individuals. columns=SNPs (see `get.biallelic.matrix in the manual`) If the parameter `include.unknown` of the `readData` function is switched to TRUE, the unknown nucleotides are `NA` in the biallelic matrix.
n.singletons
vector of length n.individuals. Number of SNPs where exactly one individual causes a SNP.
biallelic.sites
positions of the single nucleotide polymorphisms (SNP)
n.nucleotides
number of valid nucleotides for each individual.
biallelic.composition
the nucleotide distribution for each individual
synonymous
vector of length=n.snps. `TRUE`:synonymous, `FALSE`:non-synonymous,`NA`:non-coding region
biallelic.substitutions
The correspondig nucleotides of the SNPs:
first row: minor allele, second row: mayor allele
polyallelic.sites
position of polyallelic sites (>2 nucleotides)
sites.with.gaps
sites including gaps (those sites are excluded)

sites.with.unknowns
sites with unknown positions (N,n,?). Those sites are included if the parameter `include.unknown` ist `TRUE`
minor.alleles
The minor allele of the SNP represented as a numerical value
codons
a list of length=n.snps. The codon changes are represented as numerical values. This slot is only available for data in alignment format.
For SNP data we provide the function `set.synnonsyn` because of memory issues. See also `get.codons` for some detail informations about the codon-changes and `codontable()` to define your own genetic code.
<FEATURE>SNPS
boolean vector of length=n.snps, `TRUE`, if the SNP lies in a (coding, exon, intron or UTR) region. This slot will be present after reading data with the corresponding GFF-file.

4 Reading data with GFF/GTF information

The GFF folder contains GFF-files for each alignment stored in the FASTA folder. The GFF-files should have the same names as the corresponding FASTA-files (in this example: *4CL1tl.gff, C4Htl.gff and CADtl.gff*) to ensure the right matching.

```
> GENOME.class <- readData("FASTA",gffpath="GFF")
```

Which of the first 10 SNPs (`[1:10]`) of the second (`[[2]]`) alignment are part of an synonymous mutation ?

```
> GENOME.class@region.data@synonymous[[2]][1:10]
```

```
 [1] TRUE TRUE TRUE TRUE TRUE TRUE   NA   NA   NA   NA
```

`NA` values indicate that the sites are not in a coding region

```
> GENOME.class@region.data@CodingSNPS[[2]][1:10]
```

```
 [1] 1413 1428 1446 1455 1482 1488 1744 1756 1798 1802
```

4.1 Splitting the data in subsites

PopGenome can scan the data based on the features defined in the GFF file. In this example we are splitting into `coding` (CDS) regions. The returned value is again an object of class `GENOME`.

```
> GENOME.class.split <- splitting.data(GENOME.class, subsites="coding")
```

Each region contains now the SNP-informations of each coding region defined in the gff-files. In case of whole-genome SNP data this mechanism can be very useful. (see manual:`readSNP`,`readVCF` and section (*Reading data (SNP files)*))

```
> GENOME.class.split@n.sites
```

```
 [1] 1056  413  103   96  785  132  595   92  112  226  438  220
```

```
> GENOME.class.split <- neutrality.stats(GENOME.class.split)
```

Apply the neutrality module to all synonymous SNPs in the coding regions.

```
> GENOME.class.split <- neutrality.stats(GENOME.class.split, subsites="syn")
```

```
> GENOME.class.split@Tajima.D
```

The function `get.gff.info` provides additional features to extract annotation informations out of a GFF/GTF file.

5 Define the populations

Define two poulations as a list.

```
> GENOME.class <- set.populations(GENOME.class,list(
+                  c("CON","KAS-1","RUB-1","PER-1","RI-0","MR-0","TUL-0"),
+                  c("MH-0","YO-0","ITA-0","CVI-0","COL-2","LA-0","NC-1")
+                 ))
```

Individual names are returned by the function `get.individuals(GENOME.class)`

6 Define an outgroup

If one ore more outgroup sequences are defined, PopGenome will only consider SNPs where the outgroup is monomorh. The monomorhic nucleotide is than automatically defined as the major allele 0 (non mutant).

```
> GENOME.class <- set.outgroup(GENOME.class,c("Alyr-1","Alyr-2"))
```

7 Statistics

The methods and statistical tests provided by PopGenome are listed in the user manual. The corresponding references are in the *references* section.

7.1 Neutrality statistics

```
> GENOME.class  <- neutrality.stats(GENOME.class)
```

Getting the result from the object of class `GENOME`.

```
> get.neutrality(GENOME.class)
```

```
      neutrality stats
pop 1 Numeric,27
pop 2 Numeric,27
```

Lets look at the first population `[[1]]`.

```
> get.neutrality(GENOME.class)[[1]]

             Tajima.D n.segregating.sites Rozas.R_2    Fu.Li.F    Fu.Li.D
4CL1tl.fas -1.1791799                 16        NA -0.9247377 -1.1331823
C4Htl.fas   0.6987394                 17        NA  0.6742517  0.4167836
CADtl.fas   0.5503743                 14        NA  0.4458431  0.1590690
           Fu.F_S Fay.Wu.H Zeng.E Strobeck.S
4CL1tl.fas     NA      NaN    NaN         NA
C4Htl.fas      NA      NaN    NaN         NA
CADtl.fas      NA      NaN    NaN         NA
```

The `NA` values indicates that the statistics could not be calculated. This can have several reasons.

- the statistic needs an outgroup
- the statistic was not switched on
- there are no SNPs in the entire region

In each module you can switch on/off statistics and define an outgroup. (check out the PopGenome manual on CRAN). PopGenome also provides a population specific view of each statistic value.

```
> GENOME.class@Tajima.D

               pop 1      pop 2
4CL1tl.fas -1.1791799 -0.0702101
C4Htl.fas   0.6987394  1.1819777
CADtl.fas   0.5503743  0.2682897
```

If we have read in the data with the corresponding GFF files PopGenome can also analyse subsites like `exon,coding,utr or intron regions`.

```
> GENOME.class  <- neutrality.stats(GENOME.class, subsites="coding")

> GENOME.class@Tajima.D

              pop 1     pop 2
4CL1tl.fas -1.023785 0.2626617
C4Htl.fas   1.013372 1.9121846
CADtl.fas   1.981520 1.5191652
```

Or each subsite-region seperately by splitting the data as described in section 2.1.

```
> GENOME.class.split <- splitting.data(GENOME.class, subsites="coding")

> GENOME.class.split  <- neutrality.stats(GENOME.class.split)

> GENOME.class.split@Tajima.D
```

```
                  pop 1      pop 2
240 - 1295  -0.2749244 -0.3186974
1890 - 2302 -1.0062306  0.7546749
2679 - 2781 -1.0062306  0.5590170
2884 - 2979 -1.0062306         NaN
3465 - 4249          NA          NA
4337 - 4468         NaN         NaN
4696 - 5290 -1.6097384  2.1259529
6181 - 6272         NaN         NaN
6412 - 6523         NaN         NaN
7320 - 7545  0.2390231  1.8112198
7643 - 8080 -0.3018700  1.1684289
8176 - 8395         NaN         NaN
```

The `splitting.data` function just transforms the class into another object of class `GENOME`, as a consequence we can apply all methods to the transformed class `GENOME.class.split`. Lets for example analyse all non-synonymous SNPs in the coding regions.

```
> GENOME.class.split  <- neutrality.stats(GENOME.class.split, subsites="nonsyn")
```

The PopGenome framework provides several modules to calculate statistics. All methods will work in the same way as the `neutrality.stats()` function described above. Every time the input is an object of class `GENOME`.

7.2 The slot region.stats

The slot `region.stats` includes some site-specific statistics or values that can not be shown in a multi-locus-scale. See also the section `PopGenome classes`.

```
> GENOME.class@region.stats

-----
SLOTS:
-----
                   Slots                    Description  Module
1   nucleotide.diversity           Nucleotide diversity     FST
2    haplotype.diversity            Haplotype diversity     FST
3       haplotype.counts         Haplotype distribution     FST
4     minor.allele.freqs       Minor allele frequencies  Detail
5 linkage.disequilibrium         Linkage disequilibrium Linkage
6    biallelic.structure Shared and fixed polymorphisms  Detail

---------------
These are the Slots (class region.data)

> GENOME.class  <- F_ST.stats(GENOME.class)
or
> GENOME.class  <- diversity.stats(GENOME.class)

> GENOME.class@region.stats@nucleotide.diversity
```

```
[[1]]
         pop 1    pop 2
pop 1 5.142857       NA
pop 2 6.163265 5.238095

[[2]]
         pop 1 pop 2
pop 1 7.809524    NA
pop 2 8.816327     4

[[3]]
         pop 1    pop 2
pop 1 6.285714       NA
pop 2 5.836735 4.285714
```

nucleotide.diversity
The nucleotide diversity (average pairwaise nucleotide differences) within and between the populations. Have to be devided by the slot `GENOME.class@n.sites` (see also `diversity.stats`)
haplotype.diversity
The haplotype diversity (average pairwaise haplotype differences) within and between the populations. (see also: `diversity.stats`)
haplotype.counts
A vector of length=n.indivuals. Number of times a sequence of a specific indivual-sequence appears in the whole population
minor.allele.freqs
The minor allele (0) frequencies for each SNP calculated with the function `detail.stats`
linkage.disequilibrium
The function `linkage.stats(...,detail=TRUE)` calculates some linkage disequilibrium measurenments for each pair of SNP (r^2, D'...). See also: `R2.stats`
biallelic.structure
Can be calculated with the function
`detail.stats(GENOME.class, biallelic.structure=TRUE)`.
To extract the results use the function
`get.detail(GENOME.class,biallelic.structure=TRUE)`
The returned values (for each SNP) are described in the user manual.

8 Sliding Window Analysis

The `sliding.window.transform()` transforms an object of class `GENOME` in another object of class `GENOME`. This mechanism enables the user to apply all methods existing in the PopGenome framework.

PopGenome tries to concatenate the data if the parameter `whole.data` is switched to `TRUE`. This mechanism enables the user to calculate really big data which can be splitted into smaller chunks stored in the input folder. PopGenome is able to concatenate them afterwards. Functions like `readVCF` and `readSNP` will do this automatically. (see also `concatenate.regions`) If `whole.data=FALSE` the

regions are scanned seperately.

`type=1`: Scanning the SNPs
`type=2`: Scanning the nucleotides

8.1 Scanning the whole data

```
> GENOME.class.slide  <- sliding.window.transform(GENOME.class,width=50,
+                        jump=50,type=1,whole.data=TRUE)

> GENOME.class.slide@region.names

[1] "1 - 50 :"    "51 - 100 :"  "101 - 150 :" "151 - 200 :" "201 - 250 :"
[6] "251 - 300 :" "301 - 350 :" "351 - 400 :" "401 - 450 :"

> GENOME.class.slide  <- linkage.stats(GENOME.class.slide)

> get.linkage(GENOME.class.slide)[[1]]

               Wall.B    Wall.Q   Rozas.ZA    Rozas.ZZ  Kelly.Z_nS
1 - 50 :    0.6666667 0.7500000 0.66666667  0.29166667 0.375000000
51 - 100 :        NaN       NaN 0.00000000  0.00000000 0.000000000
101 - 150 : 0.0000000 0.0000000 0.01851852 -0.05266204 0.071180556
151 - 200 : 0.6250000 0.6666667 0.37847222  0.10206619 0.276406036
201 - 250 : 0.5833333 0.6923077 5.40972222  1.05354208 4.356180145
251 - 300 : 0.0000000 0.0000000 0.01388889 -0.17860000 0.192488889
301 - 350 : 0.0000000 0.0000000 0.01388889  0.00462963 0.009259259
351 - 400 : 0.4000000 0.5000000 3.95688889  2.19704321 1.759845679
401 - 450 : 0.5000000 0.6000000 1.81250000  1.31916667 0.493333333
```

The slot `GENOME.class.slide@region.names` can be used to generate the x-values for e.g, a plot along the chromosome.

```
> xaxis <- strsplit(GENOME.class.slide@region.names,split=" ; ")
> xaxis <- sapply(GENOME.class.slide@region.names,function(x){
                  return(mean(as.numeric(x)))
                  })
> plot(xaxis,GENOME.class.slide@Wall.B)
```

8.2 Scanning the regions separately

```
> GENOME.class.slide  <- sliding.window.transform(GENOME.class,width=50,
+                        jump=50,type=1,whole.data=FALSE)

> GENOME.class.slide@region.names

[1] "1:4CL1tl.fas" "2:4CL1tl.fas" "3:4CL1tl.fas" "4:C4Htl.fas"  "5:CADtl.fas"
[6] "6:CADtl.fas"  "7:CADtl.fas"

> GENOME.class.slide  <- linkage.stats(GENOME.class.slide)

> get.linkage(GENOME.class.slide)[[1]]
```

```
                Wall.B Wall.Q   Rozas.ZA    Rozas.ZZ Kelly.Z_nS
1:4CL1tl.fas 0.6666667   0.75 0.66666667  0.29166667 0.37500000
2:4CL1tl.fas       NaN    NaN 0.00000000  0.00000000 0.00000000
3:4CL1tl.fas 0.0000000   0.00 0.01851852 -0.05266204 0.07118056
4:C4Htl.fas  0.6666667   0.80 0.54086420 -0.09315802 0.63402222
5:CADtl.fas  0.0000000   0.00 2.09259259 -0.04456019 2.13715278
6:CADtl.fas  0.0000000   0.00 0.01388889 -1.37808642 1.39197531
7:CADtl.fas  0.5000000   0.60 0.88888889 -0.27527778 1.16416667
```

9 Reading data (SNP files)

PopGenome can handle SNP-data formats like VCF (1000 Genome project), HapMap and .SNP (1001 Genome project). VCF files can be read in with the function `readData(,format="VCF")`. The VCF file (same as with alignments) have to be stored in a folder. To study whole genomes, VCFs could be splitted in fairly sized chunks (by position) which should be numbered consecutively and stored in one folder. PopGenome can concatenate them afterwards in the PopGenome framework. Alternatively use the function `readVCF` which can read in a tabixed VCF-file like it is published from the 1000 Genome project. `readVCF` supports fast access of defined subregions of the file and automatically splits the data in chunks in cases when the region of interest is too big for the available RAM.
The function `readSNP` reads data published from the 1001 Genome project (Arabidopsis), where the *quality-variant.txt* files, which include variant calls from every single individual, have to be stored in a folder. The `readData` function can also read HapMap data. (`readData(,format="HapMap")`) In case of using SNP-data the `FAST` parameter can be switched to TRUE. `readData(, format="VCF", FAST=TRUE)` !!! Example files can be found in the subdirectory *data* of the PopGenome package.

9.1 Example

Reading data from the 1001 Genome project (Arabidopsis)

```
# reading chromosome 1
> GENOME.class <- readSNP("Arabidopsis", CHR=1)
# scan the data with consecutive windows
# window size: 1000 nucleotides (type=2)
# jump size: 1000 nucleotides (type=2)
> GENOME.class.slide <- sliding.window.transform(GENOME.class,1000,1000,type=2)
# calculate diversity statistics for all individuals
> GENOME.class.slide <- diversity.stats(GENOME.class.slide)
# Get the results ([[1]], because only one pop is defined)
> get.diversity(GENOME.class.slide)[[1]]
# alternative: directly access the nucleotide diversity
> plot(GENOME.class.slide@nuc.diversity.within)
```

`readSNP and readVCF` also accept a GFF-file as an input. To scan alle exons of chromosome 1 and only calculate the diversity of the nonsynonymous sites, do the following:

```
# read chromosome 1 with the corresponding GFF-file
> GENOME.class <- readSNP("Arabidopsis", CHR=1, gffpath="Ara.gff")
# verify the nonsyn/syn SNPs (we need the reference sequence as a FASTA file !)
> GENOME.class <- set.synnonsyn(GENOME.class, ref.chr="chr1.fas")
# split the data in exon regions
> GENOME.class.exons <- splitting.data(GENOME.class,subsites="exon")
# calculate the nonsynonymous diversities
> GENOME.class.exons <- diversity.stats(GENOME.class.exons, subsites="nonsyn")
```

We can split the data into `genes, exons, introns, UTRs and coding` regions if the features are present in the GFF file. See also `get.gff.info` in the manual.

10 Coalescent simulation

PopGenome supports the Coalescent simulation program `MS` from Hudson as well as the `MSMS` simulation tool from Greg Ewing. The obsereved statistics are tested against the simulated values. You have to specify the θ value and the module you want to apply to the simulated data. An new object of class `cs.stats` will be created. The main input is an object of class `GENOME`

```
> MS.class <- MS(GENOME.class,thetaID="Tajima",neutrality=TRUE)

> MS.class

-----
SLOTS:
-----
       Slots                                              Description
1  prob.less             Prob. that sim.val <=  obs.val P(sim <= obs)
2 prob.equal                Prob. that sim.val = obs.val P(sim = obs)
3 valid.iter             number of valid iter. for each test and loci
4    obs.val                                  obs.values for each test
5     n.loci                                 number of loci considered
6     n.iter                         number of iterations for each loci
7    average      average values of each statistic  (across all loci)
8   variance      variance values of each statistic (across all loci)
9      locus list of loc.stats objects, (detail stats for each locus)

---------------
```

Lets look at the data of the first region

```
> MS.class@locus[[1]]

   Length     Class      Mode
        1 loc.stats        S4
-----
SLOTS:
-----
            Slots                                                  Description
1           n.sam                        number of samples for each iteration
```

```
2          n.iter                                          number of iteration
3           theta                                           mutation parameter
4         obs.val                  vector with observed values for each test
5       positions                          position of each polymorphic site
6           trees                if printtree=1, gene tree in Newick format
7           seeds                   random numbers used to generate samples
8     halplotypes                                haplotypes in each iteration
9           stats                    variety of test stats compiled a matrix
10  loc.prob.less Prob. that simulated val. <= to observed val. P(Sim <= Obs)
11 loc.prob.equal   Prob. that simulated val = to  observed val. P(Sim = Obs)
12 loc.valid.iter                   number of valid iteration for each test
13      quantiles                               13 quantiles for each test

---------------
[1] "These are the Slots"
```

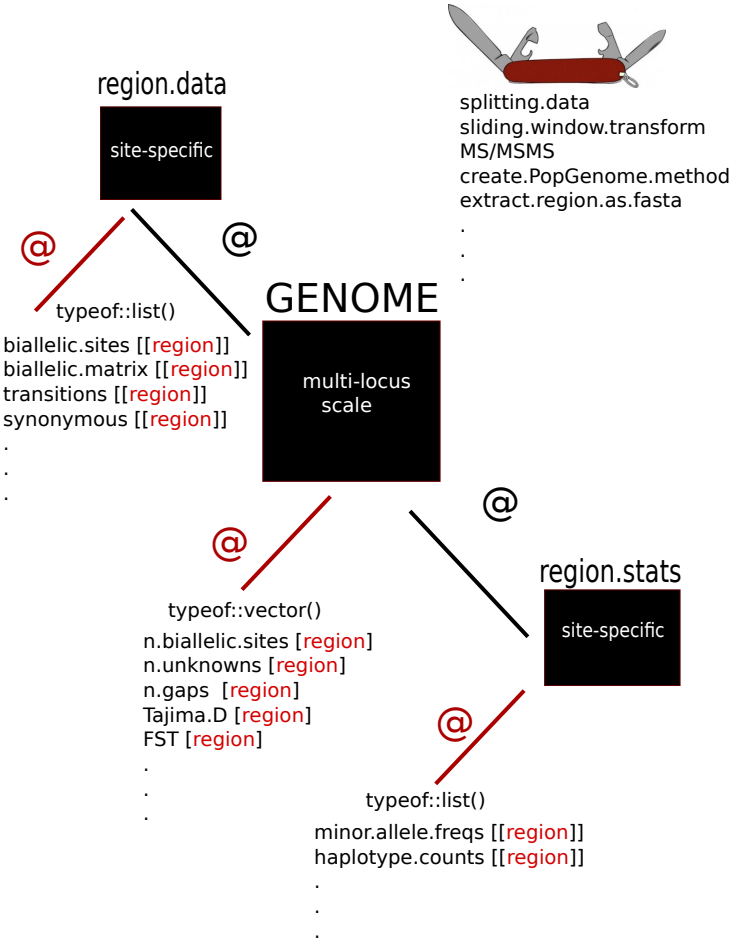
10.1 The function `readMS`

Reading data produced from the coalescent simulation programs MS (Hudson) and MSMS (Ewing).

```
> GENOME.class <- readMS(file="...")
```

After reading in the full range of methods can be applied to this data

11 PopGenome classes



12 PopGenome internals

12.1 Synonymous & Non-Synonymous Sites

PopGenome will consider every single nucleotide polymorphism (SNP) seperately an verify if the SNP is part of a synonymous or nonsynonymous change. When there is a unknown or gap position in the entire site nucleotide-triplet of a specific individual, PopGenome will ignore those sequences and will try to find a valid codon and will interpret this change. If there is one nonsynonymous change PopGenome will set this SNP as a nonsynonymous SNP, also when there are additional synonymous changes. However, the slot `GENOME.class@region.data@codons` includes all codon changes and the function `get.codons` will also give more inside. If neccessary the user can redefine the syn/nonsyn changes by manipulating the `GENOME.class@region.data@synonymous` slot or define subpositions of

interest with the `splitting.data` function.
When typing `codontable` in R the Codon-Table is printed where the rows of the second matrix of the list corresponds to the numerical values of the slot `GENOME.class@region.data@codons`.

```
> codonTable  <- codontable()
> codonTable[[2]]
```

The first matrix of this list (codonTable[[1]]) codes the corresponding Proteins of the nucleotide Triplets. PopGenome will always use the first row of this matrix (standard code) to interpret whether a change is synonymous or nonsynonymous. Here you can change the coding in the first row and load your own file in the R-environment

```
# change the file codontable.R
> library(PopGenome)
> source(".../codontable.R")
```

The function have to be `codontable()`