Integration of new Methods

PopGenome

Bastian Pfeifer

March 15, 2017

Accessing the class GENOME

The whole framework of PopGenome is based on a single class named `GENOME`. A `GENOME` object contains virtually all information about the observed data, and also stores the results of calculations. To ensure a whole-genome perspective despite limited computer memory, we use the ff-package (`http://cran.r-project.org/web/packages/ff/index.html`). PopGenome provides an effortless access to the data stored in the class `GENOME`. This obviates the need to re-implement functions already included in PopGenome and thus facilitates the easy integration of new methods. In the next sections, we will discuss the integration of new methods on the basis of alignments. Note that the same approach also works for regions of whole genome SNP data.
Let us implement Watterson's homozygosity test of neutrality:

$$H = \sum_{i=1}^{k} x_i^2$$

, where k is the total number of haplotypes, and x_i is the frequency of the i-th haplotype with:

$$\sum_{i=1}^{k} x_i = 1$$

(1) Reading the data

In this case we read three alignments stored in the folder `"Alignments"`.

```
GENOME.class <- readData("Alignments")
```

(2) Calculation

The `FST` module calculates the haplotype counts needed for the new statistic.

```
GENOME.class <- F_ST.stats(GENOME.class)
# A faster version would be:
GENOME.class <- F_ST.stats(GENOME.class,mode="haplotype",
                           only.haplotype.counts=TRUE)
```

(3) Getting the results

The haplotype counts of each alignment or region are stored in the sub-class `region.stats`.

```
haplotype.counts <- GENOME.class@region.stats@haplotype.counts
haplotype.counts
[[1]]
      CON RI-0 MR-0 TUL-0 MH-0 ITA-0 CVI-0 COL-2 LA-0 ME-0 GR-5 CHA-0 WS-0
pop.1   4    1    1     2    1     1     1     1    3    1    1     1    1
      RSCH-0 Alyrata
pop.1      1       1

[[2]] ...
[[3]] ...
```

(4) Writing your own function

```
EW_Test <- function(GENOME.class){
    GENOME.class     <- F_ST.stats(GENOME.class,only.haplotype.counts=TRUE)
    haplotype.counts <- GENOME.class@region.stats@haplotype.counts
    frequencies      <- lapply(haplotype.counts,function(x){return((x/sum(x))^2)})
    EW_values        <- sapply(frequencies,sum)
  return(EW_values}
}

EW_Test(GENOME.class)
[1] 0.09297052 0.18367347 0.07482993
```

Embedding new methods into the PopGenome framework

PopGenome provides a mechanism to fully integrate your own functions into the PopGenome framework. The next subsections will guide you through this mechanism. Let's integrate the Ewens Watterson Test.

(1) Skeleton of a PopGenome function

Use the function `create.PopGenome.method` to generate the new function.

```
# one value for one population
create.PopGenome.method("myFunction", population.specific=TRUE)
# one value for all populations
create.PopGenome.method("myFunction", population.specific=FALSE)
# site specific values
create.PopGenome.method("myFunction", site.specific=TRUE)
```

Now you find the R script *myFunction.R* in your workspace. The script itself describes where to put your function.

```
106     if(!NEWPOP) {if(length(object@region.data@popmissing[[xx]])!=0){popmiss
    popmissing]}else{respop <- 1:npops}}
107
108 # END: DO NOT EDIT
109
110 # define here your own function in the PopGenome framework.
111 # default: bial (biallelic matrix), populations (the defined populations)
112 # ... choose here everthing you want from the GENOME class
113
114 new.value[xx,respop]         <- your_intern_function(bial,populations,...)
115
116
117  }
118 }
119
120   return(new.value)
121 })
```

Figure 1: *myFunction.R (population specific)*

(2) Writing your own function

Let's fully integrate the Ewens Watterson test described above in the PopGenome framework. To better illustrate the integration of new functions, we will do this without accessing the slot `region.stats@haplotype.counts`. The following variables are useful:

bial

The variable `bial` (short for biallelic matrix) contains the SNPs of the alignments. The rownames are the individuals, and the columns correspond to the positions of the observed SNPs. (Manual:`get.biallelic.matrix`)

```
bial[1:5,1:5]
```

```
      12 13 31 44 59
CON    0  1  0  1  0
KAS-1  0  0  0  1  1
RUB-1  1  0  1  1  0
PER-1  0  0  0  0  0
RI-0   0  1  0  0  0
```

The Biallelic Matrix contains zeros (*major alleles*) and ones (*minor alleles*) with respect to the whole dataset. Because of that, the third SNP (position: 44) contains 3 minor alleles and 2 major alleles. PopGenome will manage this automatically and will redefine those values for every subset. Nevertheless, you should keep that in mind when you write your own functions.

populations

The R object `populations` contains the defined populations as rownumbers of the Biallelic Matrix. We recommend using this object, as sometimes not all individuals are present in an alignment. (`region.data@populations`)

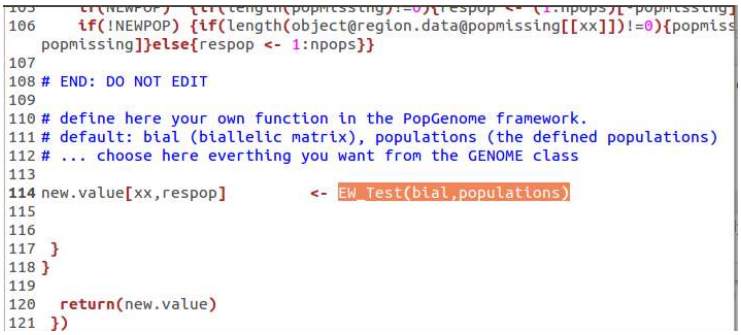

Figure 2: *myFunction.R (population specific)*

```
populations[[2]][1:5]
```

```
 [1]  1  2  3  4  5
```

In this case the first five individuals of the second (`[[2]]`) alignment are present.

Implementation

```
EW_test <- function(bial,populations){

  # Lets create the subsets of the Biallelic Matrix
  pop.bial         <- lapply(populations,function(x){return(bial[x,])})
  # Calculate the haplotype counts without accessing the class GENOME
  # calc.haplotype.counts is your own sub-function
  haplotype.counts <- lapply(pop.bial,calc.haplotype.counts)
  frequencies      <- lapply(haplotype.counts,function(x){return((x/sum(x))^2)})
  EW_values        <- sapply(frequencies,sum)

  return(EW_values)
}
```

Loading/Using the function

```
library(PopGenome)
GENOME.class <- readData("Alignments")
create.PopGenome.method("Ewens.Watterson")
# edit this new function as outlined above...
# load the edited function into R and use it:
source("Ewens.Watterson.R")
EW_values <- Ewens.Watterson(GENOME.class)
```

Now, you can use your function with the full power of the PopGenome framework.

1 Parser for new input formats

PopGenome also supports R-objects as input. You can use this capability to parse any text file and convert it into a special R-object readable by PopGenome. This R-object is a list, which contains a matrix of numerically encoded nucleotides, where each row corresponds to one individual, as well as the positions of the sites of the matrix. The positions are optional, but might be useful in case of SNP data. The nucleotides are coded as follows:

- $T, U \rightarrow 1$
- $C \rightarrow 2$
- $G \rightarrow 3$
- $A \rightarrow 4$
- $unknown \rightarrow 5$
- $- \rightarrow 6$

Example

```
matrix
     [,1] [,2] [,3] [,4]
seq1    1    1    1    2
seq2    1    4    2    1
seq3    1    4    4    1
seq4    1    4    4    1

positions
[1] 25 300 1000 2500

Robject <- list(matrix=matrix, positions=positions)

save(file="Aln", Robject)

# put the the objects in a folder (for example: .../Alignments/Aln.RData)

GENOME.class <- readData("Alignments", format="RData")

In case of SNP data:
GENOME.class <- readData("SNPData",format="RData",FAST=TRUE,SNP.DATA=TRUE)
```

You can split large datasets into chunks (by genomic position) stored in a common folder; Note, these chunks should be labeled with increasing ordered numeric values. (e.g 1.RData, 2.RData,...). PopGenome is able to concatenate them afterwards with the function `concatenate.regions`. We recommend to produce GENOME classes for each chromosome seperately.