Importing data in Hierfstat

Jérôme Goudet*

Dept Ecology & Evolution
and
Swiss institute of Bioinformatics
University of Lausanne,
Lausanne, Switzerland

September 6, 2017

*jerome.goudet@unil.ch

1 Introduction

This vignette documents how to import or enter genotypic data for the `hierfstat` package. Originally this package was written to estimate and test hierarchical F-statistics, but was then further developed and now includes almost all features of the Fstat program, which is no longer maintained, as well as several others.

2 Format required by most functions in Hierfstat

The data types that hierfstat can analyse are haploid or diploid, unphased, multilocus genotypes. Note that each data set must be made of only one ploidy level.

The basic data structure required by most `hierfstat` function is a data frame with the first column containing a population identifier (preferably a number), and the next nl columns the genotype at each of nl loci.

In `hierfstat`, alleles are encoded as 1, 2 or 3 digit numbers, and genotypes are encoded as numbers with the two alleles collated (pasted together). Other type of data can be imported (see below) but for the time being we focus on the primary data type. Thus imagine that you have an individual genotyped at a microsatellite locus with allele length `120` and `124`, the way to encode it for `hierfstat` is either `120124` or `124120`. If the data are SNPs, each allele at a locus could be encoded as `1` and `2`, or you may decide to keep the correspondence between nucleotides and alleles (e.g. `1, 2, 3, 4` for `A, C, G, T`). Thus, if the two alleles at a SNP locus are `A` and `T` and an individual is heterozygote, it could be encoded as `14` or `41`.

Example data sets are included in `hierfstat`. For instance:

```
library(pegas)

## Loading required package:  ape
## Loading required package:  adegenet
## Loading required package:  ade4
##
##    /// adegenet 2.0.1 is loaded ////////////
##
##    > overview:  '?adegenet'
##    > tutorials/doc/questions:  'adegenetWeb()'
##    > bug reports/feature requests:  adegenetIssues()
##
## Attaching package:  'pegas'
## The following object is masked from 'package:ade4':
##
##    amova
## The following object is masked from 'package:ape':
##
##    mst
```

```
library(hierfstat) #load the library

##
## Attaching package:  'hierfstat'
## The following object is masked from 'package:adegenet':
##
##    read.fstat
## The following objects are masked from 'package:ape':
##
##    pcoa, varcomp

data(diploid) # info about this data set with ?diploid
head(diploid)

##   Pop loc-1 loc-2 loc-3 loc-4 loc-5
## 1   1    44    43    43    33    44
## 2   1    44    44    43    33    44
## 3   1    44    44    43    43    44
## 4   1    44    44    NA    33    44
## 5   1    44    44    24    34    44
## 6   1    44    44    NA    43    44
```

The first individual (first row of the diploid data frame) belongs to population 1. Its genotype at `loc-1` is `44`, thus homozygote for allele `4`. It is heterozygote for alleles `3` and `4` at both `loc-2` and `loc-3`, and homozygote for allele `3` at `loc-4` and finally homozygote for allele `4` at `loc-5`. In fact, `loc-1` and `loc-4` are monomorphic, meaning that only one allele is present in all individuals from all populations.

If a genotype is missing, it is encoded as `NA`. For instance, the fourth individual has not been typed at `loc-3`, nor did the 6th individual for the same locus.

The first column of this dataframe contains the identifier of the population to which the individual belongs. We can find how many individuals were typed in each population by using the `table` command:

```
table(diploid[,1])

##
## 1 2 3 4 5 6
## 8 8 5 7 9 7
```

As another example, we look at dataset `cont.isl99`, a data frame where alleles are encoded as 2 digits numbers:

```
data(contisl99)
head(cont.isl99)

##   Pop loc.1 loc.2 loc.3 loc.4 loc.5
```

```
## 1   1  7474  1955  9168  4051  9251
## 2   1  7474  3175  9168  2410  2327
## 3   1   808  3194  9536  9751  9223
## 4   1   874  5294  1876  1310  1292
## 5   1  7484  3875  1010  5107  7712
## 6   1   874  3175  1010  5135  9292
```

The first individual is homozygous for allele `74` at the first locus (`loc.1`) and heterozygous for alleles `19` and `55` at the second locus. The genotype could have been written `5519` instead of `1955`, it does not matter. Note the genotype of the 3rd and fourth individual at the first locus. They both carry allele `8`, which is in fact encoded as `08`. When it comes first, the leading 0 disappears, but it must be present in second position. Hence genotype `874`, `0874` and `7408` are the same, but different from genotype `748` who would be understood by hierfstat as an individual heterozygous for alleles `07` and `48`.

Last point: alleles for all loci to be analysed simultaneously must be encoded with the same number of digits.

3 Importing data files

Often the data to be imported are in a text file. If this is the case, the easiest way to import the file into `R` is via one of the workhorse of `R`, the `read.table` function.

3.1 Importing FSTAT data files

If the data are in the `FSTAT` format, they can be readily imported using the function `read.fstat`:

```
dip<-hierfstat::read.fstat(system.file("extdata","diploid.dat",package="hierfstat"))
head(dip)
```

```
##   Pop loc-1 loc-2 loc-3 loc-4 loc-5
## 1   1    44    43    43    33    44
## 2   1    44    44    43    33    44
## 3   1    44    44    43    43    44
## 4   1    44    44    NA    33    44
## 5   1    44    44    24    34    44
## 6   1    44    44    NA    43    44
```

3.2 Importing from adegenet: genind objects

```
data(nancycats)
head(genind2hierfstat(nancycats)[,1:10]) # only the first 10 loci
```

```
##      pop   fca8  fca23  fca43  fca45  fca77  fca78  fca90  fca96  fca37
## N215 P01     NA 136146 139139 116120 156156 142148 199199 113113 208208
## N216 P01     NA 146146 139145 120126 156156 142148 185199 113113 208208
## N217 P01 135143 136146 141141 116116 152156 142142 197197 113113 210210
## N218 P01 133135 138138 139141 116126 150150 142148 199199  91105 208208
## N219 P01 133135 140146 141145 126126 152152 142148 193199 113113 208208
## N220 P01 135143 136146 145149 120126 150156 148148 193195  91113 208208

#basic.stats(nancycats)
#genet.dist(nancycats)
data(H3N2)
head(genind2hierfstat(H3N2,pop=rep(1,dim(H3N2@tab)[1]))[,1:10]) # only the first 10 pos

##          pop X6 X17 X39 X42 X45 X51 X60 X72 X73
## AB434107   1  1   1   3   2   3   2   3   3   2
## AB434108   1  1   1   3   2   3   2   3   3   2
## AB438242   1 NA  NA  NA  NA  NA  NA   3   3   2
## AB438243   1 NA  NA  NA  NA  NA  NA   3   3   2
## AB438244   1 NA  NA  NA  NA  NA  NA   3   3   2
## AB438245   1 NA  NA  NA  NA  NA  NA   3   3   2

#basic.stats(genind2hierfstat(H3N2,pop=rep(1,dim(H3N2@tab)[1])),diploid=FALSE)
data(eHGDP)
dim(eHGDP$tab)

## [1] 1350 8170

table(eHGDP$other$popInfo$Region)

##
##             AFRICA            AMERICA CENTRAL_SOUTH_ASIA
##                  8                 29                  8
##          EAST_ASIA             EUROPE        MIDDLE_EAST
##                 20                  8                  4
##            OCEANIA
##                  2

HGDP<-genind2hierfstat(eHGDP)
dim(HGDP)

## [1] 1350  679
```

3.3 Importing VCF files

VCF files is a standard file format adopted by many genomic platforms and used by the 1000 genomes project. Several packages offer the possibility to import VCF files and to convert it to different format (e.g. `vcfR`, `SNPRelate` in bioconductor) . We will focus here on the `pegas` package, which has a function to read these files, and from which the conversion to hierfstat format is straightforward.

For this tutorial, two files will be used, one characterising the different individuals in the 1000 genome project and another with the variant calls at chromosome 22 for these individuals.It is necessary to download them if you would like to run the example code.

We first load individual description

```
ind.desc<-read.table("./integrated_call_samples_v3.20130502.ALL.panel",header=TRUE)
head(ind.desc)

##    sample pop super_pop gender
## 1 HG00096 GBR       EUR   male
## 2 HG00097 GBR       EUR female
## 3 HG00099 GBR       EUR female
## 4 HG00100 GBR       EUR female
## 5 HG00101 GBR       EUR   male
## 6 HG00102 GBR       EUR female
```

The column names are self-explanatory. The next two tables give an idea of sample size distribution in the 1000 genomes:

```
table(ind.desc$pop,ind.desc$gender) #sample size per pop and gender

##
##       female male
##   ACB     49   47
##   ASW     35   26
##   BEB     44   42
##   CDX     49   44
##   CEU     50   49
##   CHB     57   46
##   CHS     53   52
##   CLM     51   43
##   ESN     46   53
##   FIN     61   38
##   GBR     45   46
##   GIH     47   56
##   GWD     58   55
##   IBS     53   54
##   ITU     43   59
##   JPT     48   56
```

```
##   KHV     53   46
##   LWK     55   44
##   MSL     43   42
##   MXL     32   32
##   PEL     44   41
##   PJL     48   48
##   PUR     50   54
##   STU     47   55
##   TSI     54   53
##   YRI     56   52
```

```
table(ind.desc$super_pop) # sample size per region
```

```
##
## AFR AMR EAS EUR SAS
## 661 347 504 503 489
```

Around 100 individuals per location, a bit less in American samples, roughly equal proportion of males and females.

```
fn<-"./ALL.chr22.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz"
x.l<-VCFloci(fn)[1:200,] #only a subset
```

```
## Scanning file ./ALL.chr22.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.v
##
 1000 Mb
 2000 Mb
 3000 Mb
 4000 Mb
 5000 Mb
 6000 Mb
 7000 Mb
 8000 Mb
 9000 Mb
 10000 Mb
 11000 Mb
 11223.15 Mb
## Done.
```

```
base<-c("A","T","G","C")
snps<-which(x.l$REF %in%  base & x.l$ALT %in% base)
x<-read.vcf(fn,which.loci=snps)
```

```
##
Reading 100 / 194 loci
Reading 194 / 194 loci.
## Done.
```

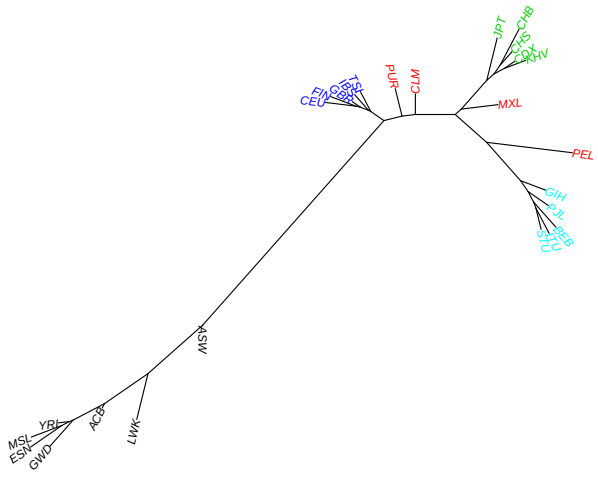
`x` is a object of class `loci` from the `pegas` package. It can be transformed in a hierfstat data frame using first `loci2genind` and then `genind2hierfstat`. Since there is no `pop` slot in the `loci` object we have just created, we need to specify explicitly the `pop` argument to `genind2hierfstat`:

```
dat<-genind2hierfstat(loci2genind(x),pop=ind.desc$pop)
#sanity check
all.equal(rownames(dat),as.character(ind.desc$sample))

## [1] TRUE
```

We can now use this data set for, e.g., drawing a simple neighbor joining tree based on Cavali-Sforza and Edwards chord distance:

```
#Cavalli-Sforza and Edwards chord distance is by default
d<-genet.dist(dat)
#naming rows anb columns
d<-as.matrix(d)
dimnames(d)[[1]]<-dimnames(d)[[2]]<-as.character(levels(dat[,1]))
#preparing colors
my.col<-as.integer(ind.desc$super_pop)
x<-table(ind.desc$pop,my.col)
my.col<-apply(x,1,function(y) which(y>0))
#do the plot
plot(bionj(d),type="unrooted",lab4ut="axial",cex=0.7,tip.color=my.col)
```



It is actually quite surprising that a bit less than 200 SNPs are sufficient to regroup populations based on their continent of origin (for most).

Exercice: Explore the `VCFloci` and `read.VCF` functions capabilities. How many SNPs are on chromosome 22? pick a random set of 1000 SNPs from chromosome 22 and redo the neighbor-joining tree.

3.4 Importing from Quantinemo

Quantinemo is a genetic simulation program for markers and traits. Genotype data generated by `Quantinemo` can be imported using the function `qn2.read.fstat`. The component `$dat` of the object returned by this function contains the genotypes of the individuals simulated:

```
dat<-qn2.read.fstat(system.file("extdata","qn2_sex.dat",package="hierfstat"))
names(dat)

## [1] "dat" "sex" "ped" "W"

head(dat$sex)

## [1] "F" "F" "F" "F" "F" "F"

head(dat$dat[,1:10])

##   Pop trait-1_locus-1 trait-1_locus-2 trait-1_locus-3 trait-1_locus-4
## 1   1             606            1515             101             404
## 2   1             606            1515             101             404
## 3   1             606            1515             101             404
## 4   1             606            1515             101             404
## 5   1             606            1515             101             404
## 6   1             606            1515             101             404
##   trait-1_locus-5 trait-1_locus-6 trait-1_locus-7 trait-1_locus-8
## 1             707             404             303             101
## 2             707             415             303             101
## 3             707             404             303             101
## 4             707             415             303             101
## 5             707             415             303             101
## 6             707            1504             303             101
##   trait-1_locus-9
## 1             808
## 2             808
## 3             808
## 4             808
## 5             808
## 6             808

#sexbias.test(dat[[1]],sex=dat[[2]])
```

3.5 Importing from ms

The program ms of Hudson is commonly used to generate genomic data.

I briefly discussed the `ms` software. Its output looks like this:

```
ms 200 100 -t 20 -I 2 100 100 40 -n 2 0.01
29161
//
segsites: 23
positions: 0.0689 0.2534 0.3219 0.3350 0.3547 0.3768 0.4339 0.4359 0.4388 0.4694 0.5003
00100001100000000000000
00001001000000000001000
```

The first line is the `ms` command line, and it instructs the program to simulate 2 populations, with $\theta = 2N_0\mu = 20$. The 2 populations differ in size and the smallest (the second) is a 100th of the larger one (the first). The two populations exchange $4Nm = 40$ migrants per generation. 100 chromosomes are sampled from each population, and this is repeated a 100 times.

The genetic data itself comes as a series of 0 and 1, collated one to the other. These are the SNP sites, with 0 being the ancestral state and 1 the alternate state.

The function `read.ms` allows reading this either as haplotypes, or as SNPs, into R:

```
msdatH<-read.ms(system.file("extdata","2pops_asspop.txt",package="hierfstat"),what="Hapl


dim(msdatH) # 2nd number is the number of Haplotypes+1

## [1] 200 101

head(msdatH[,1:11]) # fisrt 10 loci

##   Pop loc1 loc2 loc3 loc4 loc5 loc6 loc7 loc8 loc9 loc10
## 1   1   14   26   28    1   24   21   11   17    3    14
## 2   1   10   26    2    1   14    1    5    8   21     3
## 3   1   16   26   17    1    1    1   11   17    3     3
## 4   1    7    1    9   20    1   16    5    8    3     3
## 5   1    1   26   22    1    9    1    4   17    3    15
## 6   1    2   28   25   23   14    2    5    8   19     3

msdatS<-read.ms(system.file("extdata","2pops_asspop.txt",package="hierfstat"),what="SNP"

dim(msdatS)

## [1]  200 3463

head(msdatS[,1:11]) # first 10 loci
```

```
##   Pop loc1 loc2 loc3 loc4 loc5 loc6 loc7 loc8 loc9 loc10
## 1   1    0    0    1    0    0    0    0    1    1     0
## 2   1    0    0    0    0    1    0    0    1    0     0
## 3   1    1    0    0    0    0    0    0    0    0     0
## 4   1    0    0    0    0    1    0    0    1    0     0
## 5   1    0    0    0    0    0    0    0    1    0     0
## 6   1    0    0    0    0    0    0    0    1    0     0
```

```
table(msdatS[,1]) # how many inds per pop
```

```
##
##   1   2
## 100 100
```

Exercice. Install ms. Simulate 20 replicates of 100 haploid individuals in each of 2 populations. These populations exchange 1 migrant per generation, and the 2nd population size of the second population is a tenth of the first. set the scaled mutation rate to 50. Import the data set just created in a suitable format for hierfstat, as haplotype first, and then as SNPs. Compare the allele frequencies in these 2 populations.