

Reading Genetic Data Files Into R with **adegenet** and **pegas**

Emmanuel Paradis

June 9, 2010

adegenet [1] and **pegas** [2] are two packages that complement each other for population genetic analyses in R. Since various computer programs have been used by population geneticists, most of them with their own data file formats, it is crucial that R can read them to ease users to switch to R. The present document explains how to read several file formats commonly used in population genetics. The following formats are considered:

- Text (ASCII) in tabular form,
- FSTAT data format,
- GENETIX data format,
- GENEPOP data format,
- STRUCTURE data format,
- Excel.

Except the last one, these files are stored in text (usually standard ASCII) format the differences being in the layout of the data.

1 Data Structures in **adegenet** and **pegas**

First, let's have a brief look on the way allelic data are stored by our two packages. **adegenet** has the class "**genind**" where individuals are rows and alleles are columns in a matrix named **tab**. This is an S4 class, so the elements are accessed with the **@** operator (e.g., **x@tab**). Additional information are stored in other slots (**@ind.names**, **@pop**, ...) The details can be found with **class?genind**.

pegas has the class "**loci**" which is a simple data frame with a mandatory attribute named "**locicol**" which identifies the columns that are loci; the other columns are additional (individual) variables that may be of any kind. The loci are coded with factors where the different levels are the observed genotypes with the alleles separated with a forward slash, for instance, 'A/A' for a classical genotype, or '132/148' for a microsatellite locus. This is an S3 class.¹ Some examples are given in the next subsection.

¹For details: ape.mpl.ird.fr/pegas/DefinitionDataClassesPegas.pdf.

There is no need to choose between these two data structures: they are used by each package and they can be converted between each other with the functions `genind2loci` (or the generic form `as.loci`) and `loci2genind`. Therefore, it is straightforward to run analyses with both packages on the same data.

2 Reading Genetic Data Files

2.1 Reading Text Tabular Files

It is intuitive to organise allelic data in a tabular form where each row is an individual and the columns are loci and possibly other variables such as population, spatial locations, and so on. A simple example of such a file would be (file ‘toto’):²

`a/a`

This file can be read with the `pegas` function `read.loci`:

```
> library(pegas)
> x <- read.loci("toto", header = FALSE)
> x
```

```
Allelic data frame: 1 individual
                   1 locus
```

```
> print(x, details = TRUE)
```

```
      V1
1 a/a
```

```
> class(x)
```

```
[1] "loci"      "data.frame"
```

Since the file has no label for the column, we used `header = FALSE`. Note that printing a “loci” object displays a very brief account of the data; the option `details = TRUE` allows to print them like a standard data frame. If the same data were formatted with a different allele separator (file ‘titi’):

`a-a`

Then this file would be read with:

```
> y <- read.loci("titi", header = FALSE, allele.sep = "-")
> print(y, details = TRUE)
```

```
      V1
1 a/a
```

```
> identical(x, y)
```

²File contents are printed in blue to distinguish them from R input/output.

```
[1] TRUE
```

Let us have a look on the different options of `read.loci`:

```
> args(read.loci)
```

```
function (file, header = TRUE, loci.sep = "", allele.sep = "/",  
          col.pop = "none", col.loci = NULL, ...)  
NULL
```

We already know `file`, `header`, and `allele.sep`. `loci.sep` is the separator of the columns (not only the loci) which is one or several spaces by default (use `"\t"` if a tabulation). `col.pop` must be an integer giving the index of the column that will be labelled “population” in the returned data; by default there is none. `col.loci` does the same for the loci; by default all columns are treated as loci except if `col.pop` is used. Finally ‘...’ may be any further (named) arguments passed to `read.table` (e.g., `skip` in case there are comments at the top of the file).

Any level of ploidy is accepted and `pegas` checks the order of the alleles so that the returned genotypes have unordered alleles. For instance the file ‘tutu’ is:

```
a/a/A  
A/a/a
```

```
> print(read.loci("tutu", FALSE), TRUE)
```

```
      V1  
1 a/a/A  
2 a/a/A
```

A more realistic example with four columns—an allozyme locus, a microsat locus, a population assignment, and a phenotypic variable—might be (file ‘tata’):

	Adh2	SSR1	pop	size
IndA1	A/A	100/200	A	2.3
IndA2	A/a	100/120	A	2.5
IndB1	A/A	100/100	B	2.1
IndB2	a/a	120/120	B	2.8

which will be read with:

```
> z <- read.loci("tata", loci.sep = "\t", col.loci = 1:2,  
+               col.pop = 3, row.names = 1)  
> z
```

```
Allelic data frame: 4 individuals  
                   2 loci  
                   2 additional variables
```

```
> print(z, details = TRUE)
```

	Adh2	SSR1	population	size
IndA1	A/A	100/200	A	2.3
IndA2	A/a	100/120	A	2.5
IndB1	A/A	100/100	B	2.1
IndB2	a/a	120/120	B	2.8

Note `row.names` which is passed with the ‘...’ argument. To make sure that only the first and the second columns are treated as loci, let us extract the alleles from this data set:

```
> getAlleles(z)

$Adh2
[1] "a" "A"

$SSR1
[1] "100" "120" "200"
```

We may check that the attribute "locicol" has been set correctly, but usually the user does not need:

```
> attr(z, "locicol")

[1] 1 2
```

Finally we display the internal structure of the data to see that the additional variables are treated as they should be:

```
> str(z)

Classes 'loci' and 'data.frame':      4 obs. of  4 variables:
 $ Adh2      : Factor w/ 3 levels "a/a","A/a","A/A": 3 2 3 1
 $ SSR1      : Factor w/ 4 levels "100/100","100/120",...: 3 2 1 4
 $ population: Factor w/ 2 levels "A","B": 1 1 2 2
 $ size      : num  2.3 2.5 2.1 2.8
 - attr(*, "locicol")= int  1 2
```

2.2 Importing Fstat, Genetix, Genepop, and Structure Data Files

These four programs have their own data format. Roughly, these formats have the same idea: they store the genotypes of individuals from different populations. So, they store genotypes at several loci and an individual categorical variable. Additionally, the GENETIX and STRUCTURE formats allow for individual labels.

`adegenet` includes four data files in each of these format of the same microsatellite data set. These files can be displayed in the R console with:

```
> file.show(system.file("files/nancycats.dat", package="adegenet"))
> file.show(system.file("files/nancycats.gtx", package="adegenet"))
> file.show(system.file("files/nancycats.gen", package="adegenet"))
> file.show(system.file("files/nancycats.str", package="adegenet"))
```

If you want to copy these files into the working directory to further display or edit them with your favourite editor, use these commands:

```
> file.copy(system.file("files/nancycats.dat", package = "adegenet"),
+   getwd())
[1] TRUE

> file.copy(system.file("files/nancycats.gtx", package = "adegenet"),
+   getwd())
[1] TRUE

> file.copy(system.file("files/nancycats.gen", package = "adegenet"),
+   getwd())
[1] TRUE

> file.copy(system.file("files/nancycats.str", package = "adegenet"),
+   getwd())
[1] TRUE
```

adegenet provides four functions to read these formats. Reading the first three formats is straightforward:

```
> A <- read.fstat("nancycats.dat", quiet = TRUE)
> B <- read.genetix("nancycats.gtx", quiet = TRUE)
> C <- read.genepop("nancycats.gen", quiet = TRUE)
```

Reading a STRUCTURE file is slightly more complicated because the function needs some extra information. This can be done interactively (the default), or by specifying the appropriate options in which case we will use `ask = FALSE`:

```
> D <- read.structure("nancycats.str", onerowperind = FALSE,
+   n.ind = 237, n.loc = 9, col.lab = 1, col.pop = 2,
+   ask = FALSE, quiet = TRUE)
```

All four data sets are identical (we only compare the tab slots):

```
> identical(A@tab, C@tab)
[1] TRUE

> identical(B@tab, D@tab)
[1] TRUE
```

Once the data have been read into R, they can be analysed with `adegenet` or with `pegas` after eventually converting them with `as.loci`. We now delete the data files:

```
> unlink(c("nancycats.dat", "nancycats.gtx", "nancycats.gen",
+   "nancycats.str"))
```

Finally, `pegas` has the function `read.gtx` to read a GENETIX data file and return an object of class "loci". This function has no option.

2.3 Importing Excel Files

Excel is widely used for trivial data management, but clearly these data must be exported to other programs for most analyses. This also applies to the free spreadsheet editors such as OpenOffice's Calc or Gnumeric. Several solutions to get such data into R are given below. I assume that the allelic data in the spreadsheet are in a tabular form similar to what we have seen in §2.1, so the objective is to have them in R as a "loci" object.

1. The simplest solution is to save the spreadsheet as a text file using either the tab-delimited or comma-separated-variable (csv) format. This can be done with any spreadsheet editor since Calc or Gnumeric can import Excel files. Once the text file is created, `read.loci` can be used with the option `loci.sep = "\t"` or `loci.sep = ","`, as well as any other that may be needed.
2. If the "Save as..." solution does not work, it is possible to save a sheet, or part of it, in a text file by following these steps:
 - Open the file, again this may be done with any program.
 - Select the cells you want to export; this can be done by clicking once on the top-left cell, and then clicking a second time on the bottom-right cell after pressing the Shift key (this could avoid you a tunnel syndrome and is much easier if many cells must be selected).
 - Copy the selected cells in the clipboard (usually Ctrl-C).
 - Open a text editor (do not use a word processor), paste the content of the clipboard (usually Ctrl-V), and save the file.

The text file can now be read with `read.loci(..., loci.sep = "\t")`.

3. If Perl is installed on your computer (this is true for almost all Linux distributions), you can use the function `read.xls` from the package `gdata` (available on CRAN) to read directly an Excel file into R (the Perl program actually does the same job than the user does manually in the "Save as..." solution above). By default the first sheet is used, but this can be changed with the `sheet` option. The returned object is a data frame and can be converted as a "loci" object with `as.loci`. In that case, the same options that in `read.loci` can be used (see `?as.loci`).

In my experience, `read.xls` works well with small to moderate size Excel files but can be very slow with bigger files (> 10~MB).

3 Editing and Writing Genetic Data Files

After the data have been read into R, they can be manipulated in the standard way. This is straightforward for the class "loci" since it is a direct extension of data frames. `pegas` has a few method functions adapted to "loci": `rbind`, `cbind`, and the indexing operator `[]`. Some other functions, such as `subset`, `split`, `rownames`, `colnames`, or the `$` operator, can be used without problem since they respect additional attributes. Others, such as `transform`, drop the attributes and so will return a simple data frame.

adegenet allows to edit an object of class "**genind**", but since this is an S4 class, the elements are accessed with the `@` operator. The help page `?genind` describes them in details. A few functions are provided to ease the manipulation of "**genind**" objects because setting all elements by hand may be tedious: **seploc** splits the data with respect to each locus, **seppop** does the same with respect to each population, and **repool** allows to do the opposite operation.

It is also possible to select a part of a "**genind**" object with the usual indexing operator `[]`, the indices will apply to the rows and/or columns of the `@tab` slot, and the other slots will be modified accordingly. Some care should be paid when using numerical indexing on columns because something like `x[, 1]` will only select one allele column, eventually excluding other alleles of the same locus. It is safer to use the option `nloc` which specifies the loci to be selected, and so will select the appropriate allele columns. Further details are available in `?pop` as well as information on some other functions.

In addition to standard data editing, **pegas** allows to edit a "**loci**" object with **edit** or **fix**. The command `edit(x)` opens the data in R's spreadsheet-like data editor. Here are a few points about this procedure:

- It is possible to change the row and column labels (`rownames` and `colnames`).
- It is possible to add new rows (individuals): if some columns are not filled they will be given `NA`.
- You can add new genotypes and/or alleles to a locus column: the levels of the corresponding factor will be adjusted and a warning message will inform you of that.
- New columns may be added, but they can only be numerical or character vectors.
- Like most R functions, **edit** returns its results in the console if no assignment has been done, so you may prefer to call the editor with `x<-~edit(x)` or `fix(x)`.

If you forgot the assignment and don't want to lose all the changes you did, after you have closed the editor you can save the modified data with (only if you don't do any other operation after **edit**):

```
x <- .Last.value
```

References

- [1] T. Jombart. **adegenet**: a R package for the multivariate analysis of genetic markers. *Bioinformatics*, 24:1403–1405, 2008.
- [2] E. Paradis. **PEGAS**: an R package for population genetics with an integrated-modular approach. *Bioinformatics*, in revision:000–000, 2010.