Migration from poppr version 1.1 to 2.1.1

Zhian N. Kamvar[1] and Niklaus J. Grünwald[1,2]

1) Department of Botany and Plant Pathology, Oregon State University, Corvallis, OR

2) Horticultural Crops Research Laboratory, USDA-ARS, Corvallis, OR

March 15, 2016

Abstract

In 2015, *adegenet* and *poppr* both went through major, breaking changes. In this short vignette, we will demonstrate the `old2new_` family of functions that will convert your data and provide a guide of functions that have changed name or functionality.

POPPR
Population Genetics in R

Contents

1 Introduction **2**
1.1 Color Schemes 2
1.2 Major changes in *adegenet* 2
1.3 Major changes in *poppr* 3

2 Migrating data **3**
2.1 migrating genind data 3
2.2 migrating genclone data 4

3 Hierarchy functions/accessors **6**
3.1 Additional warning 7
3.1.1 Why should I use the accessors? 7

1 Introduction

In March of 2015, a population genetics in R hackathon was held at NESCent in Durham, North Carolina, USA. Details can be found here: https://github.com/NESCent/r-popgen-hackathon. This hackthon identified several issues concerning population genetics in R, one of them being the need for package efficiency. Thus, many packages, including *adegenet* and *poppr* received major updates [1]. The side effect of these updates is that backwards compatibility has been broken in making *adegenet* and *poppr* more efficient.

1.1 Color Schemes

Because this vignette will discuss both code that will work and code that will is not recommended, we will use two color schemes to distinguish the two. Note that all code that will fail is wrapped in a `try()` command.

- **Solarized-light theme for code that works**:

```
rnorm(10) # Works!

##  [1]  0.69088717 -0.86326396 -0.40833433 -0.40701859 -1.44034223
##  [6] -1.00630491  0.69886895 -0.45829552  0.09558282  1.57140844
```

- **Solarized-dark theme for code that is not recommended**:

```
try(rnorm("A")) # Throws a warning and error.

## Warning in rnorm("A"):  NAs introduced by coercion
```

1.2 Major changes in *adegenet*

Major breaking changes involve the structure of the *adegenet*'s GENIND object. The `@tab` slot has been changed from containing allelic frequency data to counts of alleles, thus reducing the size of the data by half. Several redundant slots have also disappeared including `@pop.names`, `@loc.names`, and `@ind.names`. The function `na.replace()` has been removed and modification of the data set directly is discouraged.

1.3 Major changes in *poppr*

The function `splitcombine()` has officially been removed (it was deprecated in version 1.1). Version 1.1 introduced the `@hierarchy` slot to contain population hierarchies. This slot was moved to the GENIND object and renamed to `@strata`. To make things particularly complicated, a slot called `@hierarchy` was also added to the GENIND object, but it only contains a formula. All methods associated with the previous hierarchy slot (e.g. `splithierarchy()` or `setpop()`) still exist in *poppr* as a wrapper to the true methods.

2 Migrating data

Migrating data from 1.1 to 2.0 is simple, all you have to do is use the function `old2new_genind()` or `old2new_genclone()` depending on the type of data you have. Below is an example of using these functions on old data.

2.1 migrating genind data

We will use the `partial_clone` data set from *poppr* version 1.1.5 for demonstration. First, we'll load the data.

```
library("poppr")
data("old_partial_clone", package = "poppr") # From version 1.1.5
data("partial_clone", package = "poppr")     # From version 2.0.0
```

Now, let's examine the names of the slots and compare it with the current version.

```
names(attributes(old_partial_clone)) # Has pop.names, ind.names, etc.

##  [1] "tab"       "loc.names" "loc.fac"   "loc.nall"  "all.names"
##  [6] "call"      "ind.names" "pop"       "pop.names" "ploidy"
## [11] "type"      "other"     "class"

names(partial_clone)                 # Has strata slot.

##  [1] "tab"       "loc.fac"   "loc.n.all" "all.names" "ploidy"
##  [6] "type"      "other"     "call"      "pop"       "strata"
## [11] "hierarchy"

# This is ultimately what we want
partial_clone

## /// GENIND OBJECT /////////
##
##  // 50 individuals; 10 loci; 35 alleles; size: 21.2 Kb
##
##  // Basic content
##    @tab:  50 x 35 matrix of allele counts
##    @loc.n.all: number of alleles per locus (range: 3-5)
##    @loc.fac: locus factor for the 35 columns of @tab
##    @all.names: list of allele names for each locus
##    @ploidy: ploidy of each individual  (range: 2-2)
##    @type:  codom
##    @call: old2new_genind(object = x, donor = new(class(x)))
##
##  // Optional content
##    @pop: population of each individual (group size range: 12-13)
```

We can try printing the old object, but it won't work and will throw an error:

```
try(old_partial_clone)

## /// GENIND OBJECT /////////
##
##  // 50 individuals; 10 loci; 35 alleles; size: 35 Kb
##
##  // Basic content
##    @tab:  50 x 35 matrix of allele counts

## Error in .nextMethod(x = x):  no slot of name "loc.n.all" for this object of class "genind"
```

To correct this we should use the function `old2new_genind()`.

```
opc <- old2new_genind(old_partial_clone)
opc # It prints!

## /// GENIND OBJECT /////////
##
##  // 50 individuals; 10 loci; 35 alleles; size: 20.7 Kb
##
##  // Basic content
##    @tab:  50 x 35 matrix of allele counts
##    @loc.n.all: number of alleles per locus (range: 3-5)
##    @loc.fac: locus factor for the 35 columns of @tab
##    @all.names: list of allele names for each locus
##    @ploidy: ploidy of each individual  (range: 2-2)
##    @type:  codom
##    @call: old2new_genind(object = old_partial_clone)
##
##  // Optional content
##    @pop: population of each individual (group size range: 12-13)
```

Be careful, though. **Do not use old2new_genind more than once!**.

2.2 migrating genclone data

This procedure is almost exactly the same as migrating genind data. We will use the `Pinf` data set for our demonstration.

```
data("old_Pinf", package = "poppr") # From version 1.1.5
data("Pinf", package = "poppr")     # From version 2.0.0
names(attributes(old_Pinf)) # No strata slot

##  [1] "mlg"       "hierarchy" "tab"       "loc.names" "loc.fac"
##  [6] "loc.nall"  "all.names" "call"      "ind.names" "pop"
## [11] "pop.names" "ploidy"    "type"      "other"     "class"
## [16] "strata"

names(Pinf)                 # Has strata slot
```

```
##  [1] "mlg"       "tab"       "loc.fac"   "loc.n.all" "all.names"
##  [6] "ploidy"    "type"      "other"     "call"      "pop"
## [11] "strata"    "hierarchy"

Pinf # What we want

##
## This is a genclone object
## -------------------------
## Genotype information:
##
##    72 multilocus genotypes
##    86 tetraploid individuals
##    11 codominant loci
##
## Population information:
##
##     2 strata - Continent Country
##     2 populations defined - South America North America
```

Again, we can just use `old2new_genclone()` to convert this.

```
opi <- old2new_genclone(old_Pinf)
opi # It prints!

##
## This is a genclone object
## -------------------------
## Genotype information:
##
##    72 multilocus genotypes
##    86 tetraploid individuals
##    11 codominant loci
##
## Population information:
##
##     2 strata - Continent Country
##     2 populations defined - South America North America
```

3 Hierarchy functions/accessors

Since the `@hierarchy` slot was moved to the `@strata` slot, the methods also had to change. Below is a table to guide you from the old to the new functions.

old version	new version
`sethierarchy()`	`strata()`
`gethierarchy()`	`strata()`
`splithierarchy()`	`splitStrata()`
`addhierarchy()`	`addStrata()`
`namehierarchy()`	`nameStrata()`
`setpop()`	`setPop()`

Table 1: Function conversion from version 1.1 to 2.0

While all of the previous hierarchy functions still exist, they exist in a deprecated fashion and will inform you how to properly use them. For example, if we wanted to use the old function `gethierarchy()`, it would throw a warning:

```
head(gethierarchy(Pinf, ~Continent/Country)) # Throws warning

## Warning:  'gethierarchy' has been deprecated, moved to the adegenet package, and renamed
to 'strata'.
##
##  Please use:
##  strata(Pinf, ~Continent/Country)

##        Continent                Country
## 20 South America South America_Colombia
## 21 South America South America_Colombia
## 22 South America South America_Colombia
## 23 South America South America_Colombia
## 24 South America South America_Colombia
## 25 South America  South America_Ecuador
```

```
head(strata(Pinf, ~Continent/Country))       # No warning :)

##        Continent                Country
## 20 South America South America_Colombia
## 21 South America South America_Colombia
## 22 South America South America_Colombia
## 23 South America South America_Colombia
## 24 South America South America_Colombia
## 25 South America  South America_Ecuador
```

Note, the function `setpop()` is also affected:

```
setpop(Pinf) <- ~Country # Ambiguous warning

## Warning:  'setpop' has been deprecated, moved to the adegenet package, and renamed to 'setPop'.
##
##  Please use:
##  setPop()
```

```
setPop(Pinf) <- ~Country # No warning
```

For many of the functions where you use the assignment, it can't tell what arguments you supply, so it will simply suggest the new function.

3.1 Additional warning

Please read this section if you have code that directly accesses the hierarchy or strata slot directly. An example is below:

```
myData@hierarchy <- myData@hierarchy[-2] # wat
```

If you have done such a thing, then migrating to poppr 2.0 will be a bit more difficult. The hierarchy slot now has a completely different meaning, as it will now simply contain a formula specifying which levels of the strata are hierarchical.

My recommendation for this situation is to go through your code and replace all lines where `@hierarchy` is used with the proper accessors. This is much better than other solutions [1].

3.1.1 Why should I use the accessors?

While it may seem reasonable to modify a slot directly, using the accessors is ultimately the best way to modify your data because the accessors will always verify the incoming data. This is shown in the adegenet basics tutorial (See: `adegenetTutorial("basics")`). I will show a short example here with the Pinf data set, first I will show what happens when you don't use the accessors:

```
newPinf        <- Pinf
the_strata     <- head(newPinf@strata)
newPinf@strata <- the_strata     # Only setting strata for six samples!
newPinf@strata                   # How is this allowed?

##        Continent  Country
## 20 South America Colombia
## 21 South America Colombia
## 22 South America Colombia
## 23 South America Colombia
## 24 South America Colombia
## 25 South America  Ecuador

try(setPop(newPinf) <- ~Country) # Oh.
```

```
## Error in `pop<-`(`*tmp*`, value = structure(c(1L, 1L, 1L, 1L, 1L, 2L), .Label = c("Colombia",
:
##  wrong length for population factor
```

If we had used accessors, the error would have been caught from the start!

```
the_strata          <- head(strata(Pinf))
try(strata(Pinf) <- the_strata)
```

[1] For example: a hackish way to fix your situation is to do a search and replace through your code, replacing all instances of `@hierarchy` with `@strata`

```
## Error :  Number of rows in data frame not equal to number of individuals in object.
```

References

[1] Zhian N Kamvar, Jonah C Brooks, and Niklaus J Grünwald. Novel R tools for analysis of genome-wide population genetic data with emphasis on clonality. *Frontiers in Genetics*, 6:208, 2015.