polysat version 0.1 Tutorial Manual

Lindsay V. Clark <lvclark@ucdavis.edu>
UC Davis Department of Plant Sciences
http://openwetware.org/wiki/Polysat

June 23, 2010

Contents

1 Introduction

The R package `polysat` is intended to provide useful tools for working with microsatellite data of any ploidy level, including populations of mixed ploidy. It can convert genotype data between different formats, including Applied Biosystems GeneMapper®, binary presence/absence data, ATetra, Tetra/Tetrasat, GenoDive, SPAGeDi, and Structure. It can also calculate pairwise genetic distances between samples, assist the user in estimating ploidy based on allele number, and calculate some simple population statistics. Due to the versatility of the R programming environment and the simplicity of how genotypes are stored by polysat, the user may find many ways to interface other R functions with this package, such as Principal Coordinate Analysis or AMOVA.

This manual is written to be accessible to beginning users of R. If you are a complete novice to R, it is recommended that you read through *An Introduction to R* (`http://cran.r-project.org/manuals.html`) before reading this manual or at least have both open at the same time. If you have the console open while reading the manual you can also look at the help files for base R functions (for example by typing `?dimnames` or `?%in%`) and also get more detailed information on polysat functions (e.g. `?read.GeneMapper`).

2 Obtaining and installing polysat

The R console and base system can be obtained at `http://www.r-project.org/` . Once installed, polysat and other packages (note that `combinat` is required for the `Bruvo.distance` function) can be installed and loaded from

a drop-down menu while connected to the internet. Alternatively, you can type

```
> install.packages("combinat")
> install.packages("polysat")
> library("polysat")
```

into the R console. If you instead download the package from CRAN using your web browser, you can then install it at your operating system's command prompt using `R CMD INSTALL` or in R using the function `install.packages`, and load it in R using the function `library`. (See `http://cran.r-project.org/doc/manuals/R-intro.html#Packages` for more information; the procedure is the same for any R package.) If you quit and restart R you will not have to re-install the package but you might need to load it again (using the `library` function as shown above).

3 Notes on autopolyploids vs. allopolyploids

Although polysat is equally able to store autopolyploid and allopolyploid data, it does not distinguish between the two. The user should take assumptions of the analysis into consideration depending on the inheritance pattern of the loci. In an autopolyploid, all alleles for a given microsatellite marker truly belong to one locus. In an allopolyploid, a microsatellite marker represents two or more loci (barring mutations that cause the primers to only amplify in one genome), and there is no simple way to know which alleles came from which locus without genotyping progeny of every individual. Recent software for allotetraploids [12, 6, 8] use iterative processes to assign alleles to one disomic genome or the other and calculate population statistics from there. At this time, no functions in polysat perform this type of partitioning. In particular, the functions `Bruvo.distance` and `estimate.freqs` make more sense in autopolyploids than allopolyploids. They may still produce biologically meaningful results when run on allopolyploid data, although the user should keep in mind that allopolyploid data violate the assumptions that all alleles for a given marker belong to one interbreeding pool and can be closely related to each other through mutation.

4 How genotypes are stored in polysat

The object that is used universally in polysat to store genotypes is a two-dimensional list of vectors. Samples are the first dimension of the list, and the names of the first dimension are the names of the samples. Loci are the second dimension of the list, and likewise the names of the second dimension are the names of the loci. Each vector contains all unique alleles for a given sample at a given locus. It is generally assumed that the alleles are integers, although some polysat functions may still work on numeric and character vectors. It is also assumed that copy number is never known for partially heterozygous genotypes, and so a vector of alleles contains each allele only once and is no longer than needed to contain the alleles in this way. This allows polysat the flexibility of processing samples of any ploidy, storing samples of mixed ploidy in the same project, and estimating ploidy when it is unknown. Missing data are represented by vectors of length 1, containing a symbol of the user's choosing, `-9` by default. The user may find ways of manipulating the genotype object that are not provided in the polysat package, since the object can be created and accessed using the base R package.

If you want to follow along with the examples below, first type

```
> data(testgenotypes)
> data(FCRinfo)
```

into your console to load the objects `testgenotypes` and `FCRinfo`. All of the commands in this manual can also be found in a file called "sample-session.R" in the "demo" directory of the polysat package. This file can be opened with a text editor, or if you use Emacs Speaks Statistics you can open the file there and send it to R one line at a time.

Alternatively, skip ahead to the chapter on importing data from files so that you can work with your own data.

4.1 Examples of how to view and index genotype data

If you type the name of the genotype object (either from the example data or your own) into the console in order to view it, you will get something like this (sample names in this example start with `FCR` and locus names start with `RhCBA`):

```
> testgenotypes
```

```
      RhCBA15   RhCBA23   RhCBA28
FCR1  207       Integer,2 Integer,6
FCR2  Integer,2 Integer,6 Integer,6
FCR3  208       Integer,3 Integer,8
FCR4  Integer,4 Integer,2 Integer,3
FCR5  207       Integer,3 Integer,8
FCR6  208       Integer,3 Integer,8
FCR7  Integer,4 Integer,3 Integer,5
FCR8  Integer,4 Integer,2 Integer,3
FCR9  Integer,4 Integer,2 Integer,3
FCR10 Integer,3 98        Integer,3
FCR11 Integer,4 Integer,2 Integer,3
FCR12 Integer,4 Integer,2 Integer,3
FCR13 Integer,4 Integer,2 Integer,3
FCR14 Integer,4 Integer,2 Integer,3
FCR15 Integer,4 Integer,3 182
FCR16 Integer,4 Integer,2 -9
FCR17 Integer,4 Integer,2 Integer,3
FCR18 Integer,4 Integer,2 Integer,3
FCR19 Integer,4 Integer,2 Integer,3
FCR20 Integer,4 Integer,2 Integer,3
```

As you can see, this does not allow you to view the alleles, except for genotypes that only have one allele. However, you can see the sample names and locus names that are used to index the genotype data. You can also see that the alleles are all stored as integers, and you can see how many alleles each genotype has.

Say you just wanted to view the genotypes for RhCBA23:

```
> testgenotypes[, "RhCBA23"]

$FCR1
[1] 102 111

$FCR2
[1] 102 104 106 111 123 125

$FCR3
```

```
[1] 102 106 115

$FCR4
[1]  98 125

$FCR5
[1] 102 106 115

$FCR6
[1] 102 106 115

$FCR7
[1]  98 106 126

$FCR8
[1]  98 127

$FCR9
[1]  98 127

$FCR10
[1] 98

$FCR11
[1]  98 127

$FCR12
[1]  98 127

$FCR13
[1]  98 127

$FCR14
[1]  98 127

$FCR15
[1]  98 108 117
```

```
$FCR16
[1]  98 125

$FCR17
[1]  98 126

$FCR18
[1]  98 126

$FCR19
[1]  98 126

$FCR20
[1]  98 126
```

At the locus RhCBA23, FCR1 has the alleles 102 and 111, FCR2 has the alleles 102, 104, 106, 111, 123, and 125, and so on with the other individuals.

Say you just wanted to view the genotypes for FCR14:

```
> testgenotypes["FCR14", ]

$RhCBA15
[1] 197 207 212 218

$RhCBA23
[1]  98 127

$RhCBA28
[1] 164 174 176
```

You may just want to analyze a subset of your data:

```
> myloci <- c("RhCBA23", "RhCBA28")
> mysamples <- c("FCR1", "FCR2", "FCR3", "FCR4", "FCR5", "FCR6",
+     "FCR7", "FCR8", "FCR9", "FCR10")
> subgenotypes <- testgenotypes[mysamples, myloci]
> subgenotypes
```

```
      RhCBA23   RhCBA28
FCR1  Integer,2 Integer,6
FCR2  Integer,6 Integer,6
FCR3  Integer,3 Integer,8
FCR4  Integer,2 Integer,3
FCR5  Integer,3 Integer,8
FCR6  Integer,3 Integer,8
FCR7  Integer,3 Integer,5
FCR8  Integer,2 Integer,3
FCR9  Integer,2 Integer,3
FCR10 98        Integer,3
```

When providing your genotype data as an argument for any of the functions in polysat, you may choose to just index a subset of the data in the argument rather than making a separate object to contain the subset beforehand. For example,

```
> testdist <- meandistance.matrix(testgenotypes[mysamples, myloci])
```

Many of the functions also have optional `samples` and `loci` arguments, which work exactly the same way for choosing a subset of the data. For example,

```
> testdist <- meandistance.matrix(testgenotypes, samples = mysamples,
+     loci = myloci)
```

would give exactly the same results as the similar assignment above.

You might just want to exclude a few samples without having to type the full names of all other samples:

```
> to.exclude <- c("FCR11", "FCR12")
> all.samples <- dimnames(testgenotypes)[[1]]
> to.analyze <- all.samples[!all.samples %in% to.exclude]
> to.analyze

 [1] "FCR1"  "FCR2"  "FCR3"  "FCR4"  "FCR5"  "FCR6"  "FCR7"  "FCR8"  "FCR9"
[10] "FCR10" "FCR13" "FCR14" "FCR15" "FCR16" "FCR17" "FCR18" "FCR19" "FCR20"

> testgenotypes[to.analyze, ]
```

```
      RhCBA15   RhCBA23   RhCBA28
FCR1  207       Integer,2 Integer,6
FCR2  Integer,2 Integer,6 Integer,6
FCR3  208       Integer,3 Integer,8
FCR4  Integer,4 Integer,2 Integer,3
FCR5  207       Integer,3 Integer,8
FCR6  208       Integer,3 Integer,8
FCR7  Integer,4 Integer,3 Integer,5
FCR8  Integer,4 Integer,2 Integer,3
FCR9  Integer,4 Integer,2 Integer,3
FCR10 Integer,3 98        Integer,3
FCR13 Integer,4 Integer,2 Integer,3
FCR14 Integer,4 Integer,2 Integer,3
FCR15 Integer,4 Integer,3 182
FCR16 Integer,4 Integer,2 -9
FCR17 Integer,4 Integer,2 Integer,3
FCR18 Integer,4 Integer,2 Integer,3
FCR19 Integer,4 Integer,2 Integer,3
FCR20 Integer,4 Integer,2 Integer,3
```

Note: a common error to get when indexing is a `subscript out of bounds` error. This means that you used a character string that is not actually found in the `dimnames` of the object, like `FRC15` or `FCR21` in this example.

4.2 Missing data

If a genotype is missing at a particular sample and locus, its vector will be of length one, containing a missing data symbol of the user's choice. This symbol is `-9` by default. Any polysat function that deals with missing data has an argument called `missing` (or sometimes `missingin` and `missingout` if there are options for both input and output) that can be used to indicate that the genotype object uses a different symbol for missing data, or that a different symbol should be used in the object to be created.

The function `find.missing.gen` can be used to search a genotype object and list the locus and sample name of any genotype that is missing. For example:

```
> find.missing.gen(testgenotypes)
```

```
    Locus Sample
1 RhCBA28  FCR16
```

4.3 Examples of how to edit genotype data in R

Since all of the loci in the above example have similar names, you may want to shorten their names.

```
> dimnames(testgenotypes)[[2]] <- c("C15", "C23", "C28")
```

There may be errors in the genotypes that you want to fix:

```
> testgenotypes[["FCR5", "C15"]] <- 208
> testgenotypes[["FCR19", "C23"]]

[1]  98 126

> testgenotypes[["FCR19", "C23"]] <- c(98, 125)
> testgenotypes[["FCR2", "C23"]]

[1] 102 104 106 111 123 125

> testgenotypes[["FCR2", "C23"]][4] <- 112
> testgenotypes["FCR5", ]

$C15
[1] 208

$C23
[1] 102 106 115

$C28
[1] 146 148 157 159 166 170 172 174

> testgenotypes["FCR19", ]

$C15
[1] 197 207 211 218

$C23
```

```
[1]  98 125

$C28
[1] 163 174 176
```

```
> testgenotypes["FCR2", ]
```

```
$C15
[1] 206 207

$C23
[1] 102 104 106 112 123 125

$C28
[1] 146 155 157 159 168 175
```

Why am I using double brackets to index single genotypes, while I use single brackets to index larger subsets of the data? Compare the output of the two methods when used to access a single genotype:

```
> testgenotypes["FCR7", "C28"]
```

```
[[1]]
[1] 164 174 176 179 181
```

```
> testgenotypes[["FCR7", "C28"]]
```

```
[1] 164 174 176 179 181
```

The first is a list of length one containing the vector. The second is the vector itself. In most cases you want the latter. Note that you cannot use double brackets to index multiple elements of the list:

```
> testgenotypes[["FCR7",]]
```

```
Error in testgenotypes[["FCR7", ]] : invalid subscript type 'symbol'
```

4.4 Editing genotype data using spreadsheet software

If you have a lot of edits to make or simply don't like the command-line approach to genotype editing, you may prefer to export the data to a tab-delimited text file, edit it in spreadsheet software or a text editor, then import it back into R. polysat can export and import data in a variety of formats (see chapters on exporting and importing). Personally, I think that the easiest for viewing and editing data is the Applied Biosystems GeneMapper®format, which is a simple table that stores each allele in its own cell.

```
> write.GeneMapper(testgenotypes, "genotypestoedit.txt")
```

Now open the file "genotypestoedit.txt" in your favorite spreadsheet program. If you don't know where your home directory is and can't find the file, try again but type in the full file path that you want to write to, e.g. `"C:\\Users\\lvclark\\Documents\\genotypestoedit.txt"`.

Save the file once you've finished editing it, then import it back:

```
> testgenotypes <- read.GeneMapper("genotypestoedit.txt")
```

or

```
> testgenotypes <-read.GeneMapper(
+     "C:\\Users\\lvclark\\Documents\\genotypestoedit.txt")
```

4.5 Merging genotype objects

If you have multiple genotype objects that you would like to combine into one, there are a couple methods that you could use.

If both genotype objects have the same set of samples, but different loci, the `cbind` function can combine them. If you are concerned that the samples might be in a different order or that one object has a few samples (that you want to get rid of) that the other object doesn't have, you can subscript both objects with the same vector of sample names.

```
> mygenotypes1 <- array(list(1,2,3,4), dim=c(2,2),
+     dimnames=list(c("ind1", "ind2"), c("loc1", "loc2")))
> mygenotypes2 <- array(list(5,6,7,8), dim=c(2,2),
+     dimnames=list(c("ind1", "ind2"), c("loc3", "loc4")))
```

```
> mygenotypes1

     loc1 loc2
ind1 1    3
ind2 2    4

> mygenotypes2

     loc3 loc4
ind1 5    7
ind2 6    8

> mysamples <- c("ind1", "ind2")
> mygenotypes <- cbind(mygenotypes1[mysamples, ], mygenotypes2[mysamples,
+     ])
> mygenotypes

     loc1 loc2 loc3 loc4
ind1 1    3    5    7
ind2 2    4    6    8
```

Likewise, if the two objects have the same set of loci, but different samples, they can be combined using `rbind`.

```
> mygenotypes2 <- array(list(9,10,11,12), dim=c(2,2),
+     dimnames=list(c( "ind3", "ind4"), c("loc1", "loc2")))

> mygenotypes2

     loc1 loc2
ind3 9    11
ind4 10   12

> myloci <- c("loc1", "loc2")
> mygenotypes <- rbind(mygenotypes1[, myloci], mygenotypes2[, myloci])
> mygenotypes

     loc1 loc2
ind1 1    3
ind2 2    4
ind3 9    11
ind4 10   12
```

If the situation is more complicated (e.g. you would use `cbind` but one object has a few samples that the other doesn't, and you want to preserve these samples in the final object), you can export the objects to separate GeneMapper files, then import them together into one object.

```
> write.GeneMapper(mygenotypes1, "mygenotypes1.txt")
> write.GeneMapper(mygenotypes2, "mygenotypes2.txt")
> mygenotypes <- read.GeneMapper(c("mygenotypes1.txt", "mygenotypes2.txt"))
```

4.6 Creating a genotype object from scratch

If you are storing your genotype data in a format that is not already read by one of the functions in polysat (see next chapter), you may want to write your own function to read your genotype data, or you may want to make a genotype object manually. All of the functions in polysat that produce a genotype object first establish the structure of the list and fill it with missing data symbols:

```
> missing <- -9
> samples <- c("ind1", "ind2", "ind3")
> loci <- c("loc1", "loc2")
> gendata <- array(list(missing), dim = c(length(samples), length(loci)),
+     dimnames = list(samples, loci))
> gendata

     loc1 loc2
ind1 -9   -9
ind2 -9   -9
ind3 -9   -9
```

Then the list is filled with genotype data. You can do this one genotype at a time, although it would take awhile for any reasonably sized dataset:

```
> gendata[["ind1", "loc1"]] <- c(100, 102, 104)
```

Or you can write a loop structure to fill the whole list at once:

```
> for(L in loci){
>   for(s in samples){
>     # Insert code here that would find the genotype of sample
>     # s at locus L in your data structure and convert it to a
>     # vector called thesealleles.
>     gendata[[s,L]] <- thesealleles
>   }
> }
```

If you are going to do this, some useful functions to look into are `read.table`, `readLines`, `substring`, and `strsplit`. However, hopefully you will be able to import your data using one of the functions below!

5 Importing data from files

Each of these functions creates a genotype object as described in the previous chapter. If the file format also contains information about which samples belong to which populations (`read.ATetra`, `read.Tetrasat`, `read.GenoDive`), the function also produces a vector, as long as the number of samples and with the sample names as the vector names, containing the population number for each sample. `read.Structure` and `read.SPAGeDi` can optionally produce data frames of population and other information contained in their respective file types.

Before you try any of these functions, you should make sure that you know in general how to read files into R in your operating system. You should be able to get `read.table` to work on a simple spreadsheet-like text file (tab-delimited or CSV). The file path is always a character string in quotes. Note that because backslash is an escape character in R, and is used in Windows file paths, you will have to write the backslash twice each time if you are using Windows. For example, `"C:\\Users\\lvclark\\Documents\\mygenotypedata.txt"`.

All of the data import functions are summarized below, but more information on each can be found in their respective help files.

5.1 Arguments universal to the functions

The file path to be read is a character string as in other R functions for reading files. This is the first argument, and the only required argument, for

all of the functions below except for `dominant.to.codominant` (see below for more details on this function). `read.GeneMapper` allows a character vector of any length, because it can read multiple files and combine them into one genotype object.

Most of the functions allow the user to specify which symbol is used to represent missing data in the genotype object produced. This is `-9` by default. Since the ATetra format does not allow for missing data, there is no such argument for `read.ATetra`. The argument is called `missing` for `read.GeneMapper`, `dominant.to.codominant`, `read.Tetrasat`, `read.SPAGeDi`, and `read.Genodive`, and `missingout` for `read.Structure`.

5.2 Function summaries

read.GeneMapper

This function reads one or multiple files in the Applied Biosystems GeneMapper®Genotypes Table format. `dimnames` for the genotype object are derived from the "Sample Name" and "Marker" columns. There can be as many "Allele" columns as needed to contain the data. Missing data should be indicated by deleting the row (if the function does not find a row for a particular sample-locus combination, it fills in the missing data symbol) or by filling in the missing data symbol in the first allele position. Alleles are kept in whatever format they are found in the file (fragment length in nucleotides if using the default naming scheme of GeneMapper). It is recommended that all alleles be converted to integers before reading the file.

Usage: `read.GeneMapper(infiles, missing = -9)`

read.ATetra

This function reads a file formatted for the software ATetra [12]. The data should be tetraploid, and no missing data are allowed. The function returns a list containing two elements: `PopData` and `Genotypes`. The first is a vector, with names equal to the sample names, containing the population number of each sample. The second is the genotype object. Alleles are converted to integers but otherwise stay the same as they are found in the file.

Usage: `read.ATetra(infile)`

read.Tetrasat

This function reads a file formatted for the software Tetrasat [8] or Tetra [6]. The data should be tetraploid, and missing data are recognized as a column completely made up of white space. A list is returned in the same format as that returned by `read.ATetra`. The two-digit alleles from the file are converted to integers in the genotype object. Completely homozygous genotypes are written as the same allele four times in the file, but the allele is only stored once in the genotype object.

Usage: `read.Tetrasat(infile, missing = -9)`

read.GenoDive

This function reads a file formatted for the software GenoDive [9] (or more recently `http://www.bentleydrummer.nl/software/software/GenoDive.html`). The format is similar to that for GenePop, but any ploidy is allowed. Missing data are recognized as genotypes consisting only of zeros. Alleles are converted to integers but otherwise kept the same. A list is returned in the same format as that returned by `read.ATetra` and `read.Tetrasat`.

Usage: `read.GenoDive(infile, missing = -9)`

read.Structure

This function reads a file formatted for the software Structure [3, 2, 11, 5]. At this time the ONEROWPERIND option is not supported by polysat, so each microsatellite locus must have only one column in the file, and each individual must have n rows, where n is the ploidy of the file. Otherwise, the function is flexible as to whether or not sample names and marker names are present in the file, how many rows and columns are used in addition to those containing genotype data, the character used to delimit fields, the symbol used to indicate missing data, and the ploidy of the file. If marker names are used, they must be aligned with their corresponding columns (so that the data would look right if opened in a spreadsheet program). If there are extra columns present, there is the option to return the data in these as a data frame. For each sample and locus, repeated alleles are ignored, and missing data symbols are ignored unless the entire genotype is missing.

Usage: `read.Structure(infile, missingin = -9, missingout = -9, sep = "", markernames = TRUE, labels = TRUE, extrarows = 1, extracols = 0, ploidy = 4, getexcols = FALSE)`

read.SPAGeDi

This function reads files formatted for the software SPAGeDi [4]. The user can specify whether a character is used to delimit alleles and which character it is, although this must be the same across all genotypes in the file. The function otherwise interprets codominant genotypes the same way that the software does. By default the function will only return a genotype object, but there are also options to return a vector containing the ploidy of each sample, or a data frame containing the categories and spatial coordinates of all samples.

Usage: `read.SPAGeDi(infile, allelesep = "/", returncatspatcoord = FALSE, returnploidies = FALSE, missing = -9)`

dominant.to.codominant

This function does not read a file directly, but instead takes data in the form of an array or matrix of binary allele presence/absence data. A file containing this type of data in a table-like format can be read by `read.table` and converted to a matrix by `as.matrix`, then processed by `dominant.to.codominant`. The function also requires information about which columns represent which alleles and which loci. This information can be contained in the column names themselves, with the locus name and allele number separated by a period (or other character as specified by the user). The user can alternatively provide the function with a data frame listing loci and alleles in the same order that they are found in the columns of the genotype array. The symbol that indicates the presence of an allele is 1 by default but can be changed by the user. If the function does not find any alleles present for a particular sample and locus, it inserts the missing data symbol into that position in the genotype object.

Usage: `dominant.to.codominant(domdata, colinfo = NULL, samples = dimnames(domdata)[[1]], missing = -9, allelepresent = 1, split="\.")`

5.3 Examples of usage

In the polysat package there is a folder called "extdata" which contains sample input files in all of the above formats. Open the files with formats of interest to you to see what they look like. To run these examples, you should figure

out the file path to this folder and edit the examples below accordingly. (Note that I use `paste` to simplify the creation of multiple similar file paths, but this is not a necessary part of using these functions. Try `paste` by itself or look at its help file if you are confused about what it does.)

```
> folderpath <- "C:\\Users\\lvclark\\R\\win-library\\2.11\\polysat\\extdata\\"
> ATdata <- read.ATetra(paste(folderpath,"ATetraExample.txt",sep=""))
> ATdata

$PopData
 CMW1  CMW2  CMW3  CMW4  CMW5  FCR4  FCR7 FCR14 FCR15 FCR16 FCR17
    1     1     1     1     1     2     2     2     2     2     2

$Genotypes
      CBA15     CBA23
CMW1  Integer,4 Integer,4
CMW2  Integer,4 Integer,2
CMW3  Integer,4 Integer,2
CMW4  Integer,4 Integer,4
CMW5  Integer,4 Integer,3
FCR4  Integer,4 Integer,2
FCR7  Integer,4 Integer,3
FCR14 Integer,4 Integer,2
FCR15 Integer,4 Integer,3
FCR16 Integer,4 Integer,2
FCR17 Integer,4 Integer,2

> Tetdata <- read.Tetrasat(paste(folderpath,"tetrasatExample.txt",sep=""))
> Tetdata

$PopData
 BCRHE1 BCRHE10  BCRHE2  BCRHE3  BCRHE4  BCRHE5  BCRHE6  BCRHE7  BCRHE8  BCRHE9
      1       1       1       1       1       1       1       1       1       1
    BR1    BR10     BR2     BR3     BR4     BR5     BR6     BR7     BR8     BR9
      2       2       2       2       2       2       2       2       2       2

$Genotypes
        A1_Gtype  A10_Gtype B1_Gtype  D7_Gtype  D9_Gtype  D12_Gtype
```

```
BCRHE1  Integer,2 4         Integer,2 2         3         Integer,2
BCRHE10 Integer,2 4         7         2         Integer,2 Integer,2
BCRHE2  4         4         Integer,2 2         Integer,3 Integer,2
BCRHE3  4         4         2         Integer,2 3         Integer,2
BCRHE4  4         4         Integer,2 Integer,2 3         Integer,3
BCRHE5  4         4         Integer,2 2         3         Integer,3
BCRHE6  Integer,2 4         Integer,2 2         3         7
BCRHE7  Integer,2 4         Integer,2 -9        3         7
BCRHE8  Integer,2 4         Integer,2 Integer,2 3         Integer,2
BCRHE9  Integer,2 4         Integer,2 2         3         Integer,2
BR1     Integer,2 4         5         2         3         Integer,2
BR10    Integer,3 4         Integer,2 2         3         Integer,2
BR2     Integer,3 4         7         2         3         9
BR3     Integer,3 4         7         2         3         9
BR4     Integer,3 4         7         Integer,2 3         10
BR5     Integer,3 4         7         2         3         10
BR6     Integer,2 4         Integer,2 Integer,2 3         10
BR7     Integer,2 4         Integer,2 2         3         Integer,3
BR8     Integer,3 4         7         2         3         Integer,3
BR9     Integer,2 4         7         2         3         7
```

```
> GDdata <- read.GenoDive(paste(folderpath,"genodiveExample.txt",sep=""))
> GDdata

$PopData
  John   Paul George  Ringo   Yoko
     1      1      2      2      1

$Genotypes
       loc1      loc2
John   Integer,2 Integer,2
Paul   2         -9
George 1         Integer,2
Ringo  Integer,3 Integer,2
Yoko   Integer,2 Integer,2

> GMdata <- read.GeneMapper(paste(folderpath,"GeneMapperCBA",
+      c("15.txt","23.txt","28.txt"), sep=""))
> GMdata
```

```
      RhCBA15   RhCBA23   RhCBA28
FCR1  207       Integer,2 Integer,6
FCR2  Integer,2 Integer,6 Integer,6
FCR3  208       Integer,3 Integer,8
FCR4  Integer,4 Integer,2 Integer,3
FCR5  207       Integer,3 Integer,8
FCR6  208       Integer,3 Integer,8
FCR7  Integer,4 Integer,3 Integer,5
FCR8  Integer,4 Integer,2 Integer,3
FCR9  Integer,4 Integer,2 Integer,3
FCR10 Integer,3 98        Integer,3
FCR11 Integer,4 Integer,2 Integer,3
FCR12 Integer,4 Integer,2 Integer,3
FCR13 Integer,4 Integer,2 Integer,3
FCR14 Integer,4 Integer,2 Integer,3
FCR15 Integer,4 Integer,3 182
FCR16 Integer,4 Integer,2 Integer,3
FCR17 Integer,4 Integer,2 Integer,3
FCR18 Integer,4 Integer,2 Integer,3
FCR19 Integer,4 Integer,2 Integer,3
FCR20 Integer,4 Integer,2 Integer,3
```

```
> Structdata <- read.Structure(paste(folderpath, "structureExample.txt",
+     sep=""), extracols = 1, ploidy = 8, getexcols = TRUE)
> Structdata

$ExtraCol
      V1
WIN1B  1
MCD1   2
MCD2   2
MCD3   2

$Genotypes
      RhCBA15   RhCBA23   RhCBA28   RhCBA14   RUB126    RUB262    RhCBA6
WIN1B Integer,4 Integer,2 Integer,4 Integer,3 Integer,4 208       Integer,3
MCD1  208       Integer,6 Integer,6 Integer,4 Integer,3 Integer,6 151
MCD2  208       Integer,4 Integer,7 Integer,4 Integer,2 Integer,4 150
```

```
MCD3  197       Integer,2 Integer,2 Integer,2 Integer,2 213       Integer,2
      RUB26
WIN1B 99
MCD1  Integer,3
MCD2  Integer,3
MCD3  Integer,2
```

```
> Spagdata <- read.SPAGeDi(paste(folderpath, "spagediExample.txt", sep=""),
+                          returnploidies = TRUE)
> Spagdata
```

```
$Indploidies
i1 i2 i3 i4 i5 i6 i7 i8
 4  4  2  4  2  4  4  4

$Genotypes
   locA      locB
i1 Integer,3 Integer,2
i2 Integer,4 Integer,3
i3 8         Integer,2
i4 Integer,3 22
i5 Integer,2 Integer,2
i6 Integer,4 Integer,2
i7 -9        Integer,4
i8 Integer,4 Integer,2
```

```
> Spagdata$Genotypes[,"locA"]
$i1
[1]  4 16 17

$i2
[1] 10 13 18 19

$i3
[1] 8

$i4
[1]  6 12 14
```

```
$i5
[1] 12 15

$i6
[1] 10 16 17 19

$i7
[1] -9

$i8
[1]  9 12 14 17
```

```
> Domdata <- as.matrix(read.table(paste(folderpath, "dominantExample.txt",
+     sep=""), header=TRUE,sep="\t",row.names=1))
> Domdata
```

```
     ABC1.123 ABC1.126 ABC1.129 ABC1.132 ABC1.135 ABC2.201 ABC2.203 ABC2.205
ind1        1        0        0        0        1        0        1        1
ind2        0        1        1        0        1        1        1        1
ind3        0        0        0        0        0        0        0        1
     ABC2.207 ABC2.209
ind1        0        0
ind2        1        0
ind3        0        1
```

```
> ConvDomdata <- dominant.to.codominant(Domdata)
> ConvDomdata
```

```
     ABC1      ABC2
ind1 Numeric,2 Numeric,2
ind2 Numeric,3 Numeric,4
ind3 -9        Numeric,2
```

6 Exporting genotype data to files

If you just want to save a copy of the genotype object to be opened again in R later, it is probably simplest to use the `save` function to make an .RData

file, then use `load` when you want to restore the object. Under normal circumstances the object should be saved in your R workspace anyway, but you may want a backup copy or a copy that you can open on another computer. Use the functions below if you want to be able to open the genotype data in a text editor, spreadsheet program, or other population genetic software.

Polysat has functions to write data files formatted for GenoDive, Structure, SPAGeDi, ATetra, and Tetrasat/Tetra. It can also write files in a similar format to GeneMapper genotypes tables, as well as create matrices of binary allele presence/absence data that can either be further manipulated in R or written to files using `write.table`.

For more information and examples of the use of these functions beyond what is provided in this chapter, see their respective help files.

6.1 Arguments universal to the functions

All of the functions below take as the first argument (and the only required argument) a genotype object as described earlier in this manual ("How genotypes are stored in polysat"). This argument is referred to in the help files as `gendata`. If only a subset of `gendata` is to be used, the samples and loci to use can either be specified by subscripting or by the arguments `samples` and `loci`. All functions except for `codominant.to.dominant` have a `file` argument for specifying the path to which to write the file. The file will be written to the console if no file path is specified (although see `sink` for writing all output from the console to a file).

All of the functions allow the user to specify the symbol that is used in `gendata` to represent missing data, which is `-9` by default. This is the argument `missing` in `write.ATetra`, `write.Tetrasat`, `write.GenoDive`, `write.SPAGeDi`, and `write.Structure`, and `missingin` in `codominant.to.dominant`. The Tetrasat/Tetra, SPAGeDi, and GenoDive programs have specific ways of representing missing data, which are followed by their corresponding functions. ATetra does not allow missing data, although `write.ATetra` will simply leave the corresponding allele slots blank and print a warning if there is missing data in `gendata`. In `write.Structure` and `write.GeneMapper`, the same missing data symbol is used in the output as input. In `codominant.to.dominant`, the missing data symbol to be used in the binary matrix that is produced can be specified by `missingout`.

Some file formats contain information about which samples belong to which populations. `write.ATetra`, `write.Tetrasat`, `write.SPAGeDi`, and

`write.GenoDive` have an argument called `popinfo`, which should be an integer vector (or in some cases a character vector is acceptable) containing a population number for each sample. If `popinfo` is unnamed it is assumed to be in the same order as `samples` (or `dimnames(gendata)[[1]]` by default), or it can be named with sample names. In the latter case it is okay for `samples` to be a subset of `names(popinfo)`, but not vice versa. ATetra and GenoDive also use population names, and so `write.ATetra` and `write.GenoDive` have an argument called `popnames` that takes a character vector of names for the populations, ordered by the numbers used to represent them in `popinfo`. Because Structure will take other optional information for each sample in addition to population identity, the argument `extracols` for `write.Structure` is a two dimensional array, where the first dimension is indexed by sample name and the second dimension represents the columns to be placed before the locus columns. Similarly, `write.SPAGeDi` has an argument called `spatcoord` that takes a data frame of spatial coordinates to include in the file.

Both `write.Structure` and `write.SPAGeDi` require information about the ploidy of each sample, which is specified by the integer vector `indploidies`. Like `popinfo`, this can either be named using sample names or should be in the same order as `samples`.

A first line containing comments about the data is allowed in ATetra, Tetrasat/Tetra, and GenoDive formats, and so the argument `commentline` is used in the corresponding functions to specify the character string to use in this line.

polysat allows alleles to be stored as fragment lengths rather than repeat numbers, and so alleles must be converted to repeat numbers for Tetrasat/Tetra, SPAGeDi, and GenoDive formats. The argument `usatnts` is an integer vector containing the length of the repeat for each locus (2 for dinucleotide repeats, 3 for trinucleotide repeats, etc.) Note that if the alleles in the genotype object are already stored as repeat numbers rather than length in nucleotides, the value of `usatnts` for that locus should be 1. This vector should also be named using the locus names.

6.2 Function summaries

write.Structure

This function creates a file to be read by the program Structure [3]. The user may specify the ploidy of each sample as well as the overall ploidy (number of rows per sample) of the file. Locus names and sample names are taken from the `dimnames` of the genotype object. A RECESSIVEALLELES row is automatically inserted under the row of marker names and contains the missing data symbol, which signifies to the program that allele copy number is unknown. An array containing PopData or any other columns to be used in the file can also be supplied to the function. An especially useful aspect of the function is that it duplicates or randomly removes alleles as necessary to get to the right ploidy. Sample names and other sample info are also duplicated for each row as required by Structure. Because `write.table` is used by the function to create the file, the user must manually delete a few fields in the upper left corner of the file before importing the data into Structure.

Usage: `write.Structure(gendata, ploidy, file="", samples=dimnames(gendata)[[1]], loci=dimnames(gendata)[[2]], indploidies=rep(ploidy,times=length(samples)), extracols=NULL, missing=-9)`

write.ATetra

This function writes a file to be read by the software ATetra [12]. If missing data is found in the genotype object, a warning is printed and all allele fields for that particular sample and locus are left blank. If genotypes with more than four alleles are found, a warning is printed and four alleles are chosen at random to be included in the file.

Usage: `write.ATetra(gendata, samples = dimnames(gendata)[[1]], loci = dimnames(gendata)[[2]], popinfo = rep(1, length(samples)), popnames = "onebigpop", commentline = "insert data info here", missing = -9, file = "")`

write.Tetrasat

This function writes a file to be read by the software Tetrasat [8] or Tetra [6]. If missing data is encountered, the genotype field for that particular sample and locus is left blank. If a genotype has more than four alleles, a warning is printed and four alleles are chosen at random to be included in the

file. Alleles are also converted to repeat numbers by dividing by the number supplied in `usatnts`, and if necessary a multiple of 10 is subtracted from all alleles at a locus so that all alleles can be represented by two digits. If a genotype has only one allele, the allele is repeated four times to represent a fully homozygous genotype. If the missing data symbol is encountered in the genotype object, that particular genotype is represented in the file by white space.

Usage: `write.Tetrasat(gendata, commentline = "insert data description here", samples = dimnames(gendata)[[1]], loci = dimnames(gendata)[[2]], popinfo = rep(1, length(samples)), usatnts = rep(2, length(loci)), file = "", missing = -9)`

write.GenoDive

This function writes a file to be read by the software GenoDive [9] (or more recently `http://www.bentleydrummer.nl/software/software/GenoDive.html`). Alleles can be represented by either two or three digits as specified by the user. Alleles are converted to repeat numbers by dividing by the numbers supplied in `usatnts`, and if necessary a multiple of 10 is subtracted from all alleles at a locus so that all alleles can be represented by the specified number of digits. If a missing data symbol is found in the genotype object, zeros are written in the corresponding position in the file.

Usage: `write.GenoDive(gendata, popnames = "onebigpop", commentline = "file description goes here", digits = 2, file = "", samples = dimnames(gendata)[[1]], loci = dimnames(gendata)[[2]], popinfo = rep(1, times = length(samples)), usatnts = rep(2, times = length(loci)), missing=-9)`

write.GeneMapper

This function writes a file in a similar format to the Genotypes Tables exported from Applied Biosystems GeneMapper®. This format is not read by any other population genetic software (to the best of my knowledge) but may be convenient for viewing and editing the data. The same missing data symbol is used in the file as is encountered in the genotype object. The file produced is a tab-delimited table containing columns for Sample Name, Marker, and Alleles.

Usage: `write.GeneMapper(gendata, file = "", samples = dimnames(gendata)[[1]], loci = dimnames(gendata)[[2]])`

write.SPAGeDi

This function writes a file readable by the software SPAGeDi [4]. Population identities as specified in `popinfo` will be put into a column labeled "Cat" for categories to be used by the program. Any number of spatial coordinates can be used, and are given to the function as a data frame with sample names as row names. (As with `popinfo` and `indploidies`, the data frame will be assumed to be in the same order as `samples` if unnamed, or if named the data frame will be automatically subscripted by `samples` before being used by the function.) By default, a category column and two spatial coordinate columns are written to the file, to later be edited by the user with spreadsheet software. Alleles are converted from fragment length to repeat number similarly to the conversion in `write.GenoDive`. Since ploidy is reflected in how the genotypes are written for SPAGeDi, there is an `indploidies` argument similar to that for `write.Structure`. The first line of the file is generated automatically from the data provided. If latitude and longitude are used for spatial coordinates, the user will have to manually change the third number in the first line from 2 to -2. There is currently not an option to write distance classes to the second line of the file using this function, but the user can easily make this modification after the file is written.

Usage: `write.SPAGeDi(gendata, samples = dimnames(gendata)[[1]], loci = dimnames(gendata)[[2]], indploidies = rep(4, length(samples)), popinfo = rep(1, length(samples)), allelesep = "/", digits = 2, file = "", spatcoord = data.frame(X = rep(1, length(samples)), Y = rep(1, length(samples)), row.names = samples), usatnts = rep(2, length(loci)), missing = -9)`

codominant.to.dominant

This function creates a matrix of binary data indicating the presence or absence of each allele in each sample. The matrix can then be saved to a text file using the `write.table` or `write` function. The matrix may also be of use directly in R, for example in an AMOVA analysis using the `ade4` package. The user has control over which symbols are used to represent missing data (both in the input and the output) and the presence or absence of alleles

(in the output). In addition to the matrix of genotype data, a data frame can optionally be returned containing locus names and allele numbers in the same order as the columns of the matrix.

Usage: `codominant.to.dominant(gendata, makecolinfo = FALSE, allelepresent = 1, alleleabsent = 0, missingin = -9, missingout = -9, loci = dimnames(gendata)[[2]], samples = dimnames(gendata)[[1]])`

6.3 Examples of usage

```
> mypopinfo <- FCRinfo$Species
> names(mypopinfo) <- row.names(FCRinfo)
> mypopinfo

 FCR1  FCR2  FCR3  FCR4  FCR5  FCR6  FCR7  FCR8  FCR9 FCR10 FCR11 FCR12 FCR13
    1     1     1     2     1     1     2     2     2     2     2     2     2
FCR14 FCR15 FCR16 FCR17 FCR18 FCR19 FCR20
    2     3     2     2     2     2     2

> mypopnames <- c("A", "B", "C")
> write.GeneMapper(testgenotypes, file = "GMout.txt")
> write.Structure(testgenotypes, ploidy = 8, file = "Structout.txt",
+     indploidies = c(8, 8, 8, 4, 8, 8, rep(4, 14)), extracols = array(mypopinfo
+         dim = c(20, 1), dimnames = list(names(mypopinfo), "PopData")))
> write.GenoDive(testgenotypes, mypopnames, file = "GDout.txt",
+     popinfo = mypopinfo)
> tetrasamples <- names(mypopinfo)[mypopinfo != 1]
> tetrasamples

 [1] "FCR4"  "FCR7"  "FCR8"  "FCR9"  "FCR10" "FCR11" "FCR12" "FCR13" "FCR14"
[10] "FCR15" "FCR16" "FCR17" "FCR18" "FCR19" "FCR20"

> write.ATetra(testgenotypes, popinfo = mypopinfo, popnames = mypopnames,
+     file = "ATout.txt", samples = tetrasamples)

More than 4 alleles: FCR7 C28
Missing data: FCR16 C28

> write.Tetrasat(testgenotypes, popinfo = mypopinfo[tetrasamples],
+     file = "TSout.txt", samples = tetrasamples)
```

```
Alleles randomly removed: FCR7 C28

> write.SPAGeDi(testgenotypes, file = "SpagOut.txt", popinfo = mypopinfo,
+     spatcoord = data.frame(Lat = c(rep(43.943, 6), rep(43.957,
+         14)), Long = c(rep(-122.768, 6), rep(-122.755, 14))),
+     indploidies = c(8, 8, 8, 4, 8, 8, rep(4, 14)))

Alleles randomly removed to get to ploidy: C28 FCR7

> Domdata <- codominant.to.dominant(testgenotypes)
> write.table(Domdata, file = "Domout.txt")
```

7 Individual-level statistics

7.1 Create a matrix of pairwise distances

Typically, this will be done using the function `meandistance.matrix`, which calls `distance.matrix.1locus`, which in turn calls a function to calculate a distance given two genotypes (`Bruvo.distance` by default). `meandistance.matrix` has its own arguments and can also pass arguments on to the functions that it calls. These arguments are:

`gendata` The only required argument. A genotype object in the standard two-dimensional list of vectors format.

`samples` A character vector of samples to analyze. This must be a subset of `dimnames(gendata)[[1]]`.

`loci` A character vector of loci to analyze. This must be a subset of `dimnames(gendata)[[2]]`.

`all.distances` Boolean. If `TRUE`, a three-dimensional array of pairwise distances by locus will be produced in addition to the mean distance matrix.

`distmetric` The function to be used to calculate genetic distance. This is `Bruvo.distance` by default, which incorporates a stepwise mutation model [1]. A distance metric included in polysat that uses an infinite allele model [7] is `Lynch.distance`.

`usatnts` An integer vector containing the repeat length of each locus, if this information is used by `distmetric`. For example, 2 would indicate dinucleotide repeats, 3 would indicate trinucleotide repeats, and 1 would indicate mononucleotide repeats. Note that if the alleles are already stored in terms of repeat number rather than nucleotide length, 1 should be used! The names of the vector are the same as the names of loci and the second dimension of `gendata`.

`missing` The symbol that is used to indicate missing data in the genotype object (`-9` by default). In the array of pairwise distances by locus, `NA` will be inserted into positions where either sample has missing data. When these distances are averaged to create the mean distance matrix, any `NA` values will be ignored.

`progress` Boolean. If `TRUE`, print sample names after each pairwise distance calculation is performed. If evaluation is expected to take a long time, this can be useful for monitoring the progress.

`maxl` If distmetric is `Bruvo.distance`, and two genotypes both contain more than this number of alleles, the calculation will be skipped and `NA` will be written to that position in the array instead (similarly to when there is missing data). This is `9` by default, and is intended to save processing time by skipping a few rare distance calculations that would be too computationally intensive.

7.1.1 Examples of creating a mean distance matrix

To create two matrices based on the two measures of distance supplied with polysat:

```
> Bmatrix <- meandistance.matrix(testgenotypes, progress = FALSE)
> Lmatrix <- meandistance.matrix(testgenotypes, distmetric = Lynch.distance,
+     progress = FALSE)
```

The symmetrical distance matrix that is produced by `meandistance.matrix` can be immediately used by other R functions. For example, to view a histogram of all distances,

```
> hist(as.vector(Bmatrix))
> hist(as.vector(Lmatrix))
```

should produce plots that could be useful for determining distance thresholds between clones, populations, and species.

To do a principal coordinate analysis:

```
> Bprcomp <- cmdscale(Bmatrix)
> Lprcomp <- cmdscale(Lmatrix)
> plot(Bprcomp[,1],Bprcomp[,2], col=FCRinfo$Plot.color,
      pch=FCRinfo$Plot.symbol)
> plot(Lprcomp[,1],Lprcomp[,2], col=FCRinfo$Plot.color,
      pch=FCRinfo$Plot.symbol)
```

`write.table` and `write` can be used to export the distance matrix for use in other software.

```
> write.table(Bmatrix, file = "Bmatrix.txt")
```

`meandist.from.array` can take a three-dimensional array such as that produced when `all.distances=TRUE` and recalculate a mean distance matrix from it. This could be useful, for example, if you want to try omitting loci from your analysis. If `Bruvo.distance` skips some calculations because `maxl` is exceeded, you may also want to estimate these distances and fill them into the array manually, then recalculate the mean distance matrix. See the help file for `meandist.from.array` for some additional functions that can help to locate missing values in the three-dimensional distance array.

To experiment with excluding loci:

```
> Larray <- meandistance.matrix(testgenotypes, progress = FALSE,
+     distmetric = Lynch.distance, all.distances = TRUE)[[1]]
> mdist15.23 <- meandist.from.array(Larray, loci = c("C15", "C23"))
> mdist23.28 <- meandist.from.array(Larray, loci = c("C23", "C28"))
> mdist15.28 <- meandist.from.array(Larray, loci = c("C15", "C28"))
```

And from there you might want to do principal coordinate analyses on these three matrices as before, in order to visualize the effects of excluding loci.

7.2 Estimate the ploidy of samples

The function `estimate.ploidy` calculates the maximum number of alleles and mean number of alleles for each sample across all loci. Its only required argument is a genotype object in the standard two-dimensional list of vectors format. Optional arguments include `samples` and `loci` for specifying a subset of the data to be used.

Using the sample data provided in the package, we would write:

```
> myploidies <- as.data.frame(estimate.ploidy(testgenotypes))
> myploidies

      max.alleles mean.alleles
FCR1            6     3.000000
FCR2            6     4.666667
FCR3            8     4.000000
FCR4            4     3.000000
FCR5            8     4.000000
FCR6            8     4.000000
FCR7            5     4.000000
FCR8            4     3.000000
FCR9            4     3.000000
FCR10           3     2.333333
FCR11           4     3.000000
FCR12           4     3.000000
FCR13           4     3.000000
FCR14           4     3.000000
FCR15           4     2.666667
FCR16           4     2.333333
FCR17           4     3.000000
FCR18           4     3.000000
FCR19           4     3.000000
FCR20           4     3.000000

> myploidies[[3]] <- myploidies$max.alleles
> names(myploidies)[3] <- "ploidy"

> myploidies <- edit(myploidies)
```

This opens up a Data Editor. In the `ploidy` column, you can now edit the ploidy based on what you know about the organism, for example if you were expecting tetraploid, hexaploid, and octoploid individuals. You might also want to make a character vector containing species or phenotypic information for the samples, and make this another column of the data frame to assist with ploidy editing.

8 Population-level statistics

The population statistics in polysat allow for mixed ploidy populations. Because of this, population sizes are measured in number of genomes rather than number of individuals. For example, a tetraploid individual makes twice as much of a contribution to allele frequency as a diploid individual does, if both have the same two alleles at a locus. When allele frequencies are averaged for the calculation of H_T, or an average H_S value is calculated between two populations, the averages are weighted by the number of genomes in each population.

8.1 Estimating allele frequencies

In partially heterozygous polyploid genotypes, allele copy number is assumed to be unknown (as in all polysat functions), so allele frequencies can only be estimated and not truly calculated from the data. The function `estimate.freq` assumes that in a partially heterozygous genotype, all alleles have an equal probability of being present in more than one copy. This is of course not true, because some alleles are more common in the population. The result is that this function should underestimate the frequencies of common alleles and overestimate the frequencies of rare alleles. If these allele frequencies are used to calculate F_{ST}, then, F_{ST} will be underestimated. However, this should still be useful for looking at relative amounts of population structure within one study system.

The first and only required argument for `estimate.freq` is a genotype object. The default is that all individuals are in one population and tetraploid, although it is likely that the user will want to adjust this using the `popinfo` and `indploidies` arguments. These arguments are named vectors similar to their counterparts in some of the functions for exporting data. The names of both are sample names. Each element of `popinfo` is the population number

or population name for that particular sample. The `indploidies` argument should contain an integer for each sample indicating the ploidy of the sample. `missing`, `samples`, and `loci` arguments are also provided and are used in the same way as in other polysat functions to specify the symbol used to represent missing data, a subset of samples to be used, and a subset of loci to be used, respectively.

`estimate.freq` produces a data frame with one row per population. The first column is called `Genomes` and contains the number of genomes in each population. All remaining columns represent alleles (one column per allele) and contain allele frequencies. The column names are the locus name and allele name separated by a period. Within one population and locus, all allele frequencies will total to 1. The frequencies are the estimated number of copies of the allele in the population divided by the total number of genomes in the population. If a sample has missing data at a locus, the number of genomes in that population is reduced accordingly for that locus in the calculation.

Using the data provided in the package:

```
> mypopinfo <- FCRinfo$Species
> names(mypopinfo) <- row.names(FCRinfo)
> myploidies <- c(8, 8, 8, 4, 8, 8, rep(4, 14))
> names(myploidies) <- row.names(FCRinfo)
> freqtable <- estimate.freq(testgenotypes, popinfo = mypopinfo,
+     indploidies = myploidies)
> freqtable[, 1:10]

  Genomes   C15.197    C15.199 C15.206   C15.207    C15.208    C15.209
1      40 0.0000000 0.00000000     0.1 0.3000000 0.60000000 0.00000000
2      56 0.2321429 0.02380952     0.0 0.1785714 0.05357143 0.02380952
3       4 0.2500000 0.00000000     0.0 0.0000000 0.25000000 0.00000000
    C15.211   C15.212   C15.218
1 0.0000000 0.0000000 0.0000000
2 0.1071429 0.1607143 0.2202381
3 0.2500000 0.2500000 0.0000000
```

In addition to calculating F_{ST} (see below), see the stats function `dist` for other measures of distance that can be calculated from this type of numerical data. The software SPAGeDi [4] can also calculate a variety of distances between populations.

8.2 Calculating pairwise F_{ST}

The function `calcFst` will calculate pairwise F_{ST} [10] values between populations based on a data frame of genomes per population and allele frequencies per population. The data frame produced by `estimate.freq` can be passed directly to `calcFst`. If only a subset of populations or loci from the data frame should be used, these can be specified by vectors with the arguments `pops` and `loci`.

Continuing the above example:

```
> testfsts <- calcFst(freqtable)
> testfsts

           1             2          3
1 0.00000000  1.215551e-01 0.07300686
2 0.12155509 -1.481901e-16 0.04495806
3 0.07300686  4.495806e-02 0.00000000
```

9 How to cite polysat

We are submitting an article to Molecular Ecology Resources:

Clark, L and Jasieniuk, M. POLYSAT: an R package for polyploid microsatellite analysis. *Molecular Ecology Resources* (in review).

References

[1] BRUVO, R., MICHIELS, N. K., D'SOUZA, T. G. and SCHULENBURG, H. 2004. A simple method for the calculation of microsatellite genotype distances irrespective of ploidy level. *Molecular Ecology*, 13, 2101-2106.

[2] FALUSH, D., STEPHENS, M. and PRITCHARD, J. K. 2003. Inference of population structure using multilocus genotype data: Linked loci and correlated allele frequencies. *Genetics*, 164, 1567-1587.

[3] FALUSH, D., STEPHENS, M. and PRITCHARD, J. K. 2007. Inference of population structure using multilocus genotype data: dominant markers and null alleles. *Molecular Ecology Notes*, 7, 574-578.

[4] HARDY, O. J. and VEKEMANS, X. 2002. SPAGEDi: a versatile computer program to analyse spatial genetic structure at the individual or population levels. *Molecular Ecology Notes*, 2, 618-620.

[5] HUBISZ, M. J., FALUSH, D., STEPHENS, M. and PRITCHARD, J. K. 2009. Inferring weak population structure with the assistance of sample group information. *Molecular Ecology Resources*, 9, 1322-1332.

[6] LIAO, W. J., ZHU, B. R., ZENG, Y. F. and ZHANG, D. Y. 2008. TETRA: an improved program for population genetic analysis of allotetraploid microsatellite data. *Molecular Ecology Resources*, 8, 1260-1262.

[7] LYNCH, M. 1990. THE SIMILARITY INDEX AND DNA FINGERPRINTING. *Molecular Biology and Evolution*, 7, 478-484.

[8] MARKWITH, S. H., STEWART, D. J. and DYER, J. L. 2006. TETRASAT: a program for the population analysis of allotetraploid microsatellite data. *Molecular Ecology Notes*, 6, 586-589.

[9] MEIRMANS, P. G. and VAN TIENDEREN, P. H. 2004. GENOTYPE and GENODIVE: two programs for the analysis of genetic diversity of asexual organisms. *Molecular Ecology Notes*, 4, 792-794.

[10] NEI, M. 1973. Analysis of gene diversity in subdivided populations. *Proceedings of the National Academy of Sciences of the United States of America* 70, 3321-3323.

[11] PRITCHARD, J. K., STEPHENS, M. and DONNELLY, P. 2000. Inference of population structure using multilocus genotype data. *Genetics*, 155, 945-959.

[12] VAN PUYVELDE, K., VAN GEERT, A. and TRIEST, L. 2010. ATETRA, a new software program to analyse tetraploid microsatellite data: comparison with TETRA and TETRASAT. *Molecular Ecology Resources*, 10, 331-334.